Georgia Institute of Technology

ECE 4803: Fundamentamentals of Machine Learning (FunML)

Spring 2022

Homework Assignment # 4

Due: Friday, March 11, 2022 @8PM

Please read the following instructions carefully.

- The entire homework assignment is to be completed on this `ipython` notebook. It is designed to be used with `Google Colab`, but you may use other tools (e.g., Jupyter Lab) as well.
- Make sure that you execute all cells in a way so their output is printed beneath the corresponding cell. Thus, after successfully executing all cells properly, the resulting notebook has all the questions and your answers.
- Print a PDF copy of the notebook with all its outputs printed and submit the **PDF** on `Canvas` under Assignments.
- Make sure you delete any scratch cells before you export this document as a PDF. Do not change the order of the questions and do not remove any part of the questions. Edit at the indicated places only.
- Rename the PDF according to the format: ***LastName_FirstName_ECE_4803_sp22_assignment_#.pdf***
- It is encouraged for you to discuss homework problems amongst each other, but any copying is strictly prohibited and will be subject to Georgia Tech Honor Code.
- Late homework is not accepted unless arranged otherwise and in advance.
- Comment on your codes.
- Refer to the tutorial and the supplementary/reading materials that are posted on `Canvas` for lectures 11, 12, 13 to help you with this assignment.
- **IMPORTANT:** Start your solution with a **BOLD RED** text that includes the words *solution* and the part of the problem you are working on. For example, start your solution for Part (c) of Problem 2 by having the first line as:
  Solution to Problem 2 Part (c). Failing to do so may result in a *20% penalty* of the total grade.

Assignment Objectives:

- Understand the intuition behind various clustering algorithms discussed in class
- Connect concepts related to different clustering algorithms
- Implement and evaluate clustering techniques
- Implement clustering algorithms on real world datasets

Guide for Exporting Ipython Notebook to PDF:

Here is a video summarizes how to export Ipythin Notebook into PDF.

- **[Method1: Print to PDF]**
  After you run every cell and get their outputs, you can use **[File] -> [Print]** and then choose **[Save as PDF]** to export this Ipython Notebook to PDF for submission.
  Note: Sometimes figures or texts are splited into different pages. Try to tweak the layout by adding empty lines to avoid this effect as much as you can.
- **[Method2: colab-pdf script]**
  The author of that video provided an alternative method that can generate better layout PDF. However, it only works for Ipythin Notebook without embedded images.
  How to use: Put the script below into cells at the end of your Ipythin Notebook. After you run the fisrt cell, it will ask for google drive permission. Executing the second cell will generate the PDF file in your google drive home directory. Make sure you use the correct path and file name.

```
## this will link colab with your google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
%%capture
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
```

```
from colab_pdf import colab_pdf
colab_pdf('LastName_FirstName_ECE_4803_sp22_assignment_#.ipynb') ## change path and file name
```

- **[Method3: GoFullPage Chrome Extension] (most recommended)**
  Install the extension and generate PDF file of the Ipython Notebook in the browser.

Problem 1: K-Means and Gaussian Mixture Models (GMMs) on a Toy Example (20pts)

Suppose we are given a dataset with 5 training examples, each with two features as shown below.

Datapoint ($\mathbf{x}_i$)	Feature 1 Value ($\mathbf{x}_{i1}$)	Feature 2 Value ($\mathbf{x}_{i2}$)
1	0	0
2	2.5	1.5
3	0.5	0.5
4	0.75	1.5
5	0	1

In this problem, you will be running the K-means and the GMM clustering algorithms on this dataset step by step to gain an insight into the algorithms.

(a)

In this part, you will run the K-means clustering algorithm on the data above for two iterations. Fill out the tables below for each iteration.

Follow the steps highlighted in Lecture 11 (21-Feb-2022) Page 34 of the PDF where five steps are listed. The difference is that you do not have a convergence criterion but instead you will stop after the second iteration. Use $k = 2$ clusters. Initialize the centroids using the following mean values, $\mathbf{u}_1 = [0, 0]^T$ and $\mathbf{u}_2 = [0.75, 1.5]^T$ respectively. You may do the intermediate calculations on scratch paper using either a calculator or a computer program, but do your best to understand every step.

Write down the values for the distances of each of the datapoint from the means in each iteration and the resulting cluster assingments in the tables, respectively.

The point distances are calculated using the following formula:

$$||\mathbf{x}_i - \mathbf{u}_k||_2^2$$

The cluster assingments are obtained as below:

$$\underset{k}{\operatorname{argmin}} \|\mathbf{x}_i - \mathbf{u}_k\|_2^2$$

Provide also the means after each iteration. (all numbers round to at least 4 decimals.)

(b)

With the dataset above, you will use GMM in this part to determine the clustering assignment.

Use $K = 2$ clusters. Use the same initializations for the means, $\mathbf{u}_1 = [0, 0]^T$ and $\mathbf{u}_2 = [0.75, 1.5]^T$, respectively. Run 2 iterations of the GMM algorithm. Assume the initial priors to be equal, i.e., $p(\mathbf{u}_k, \Sigma_k) = 0.5$. Assume that the initial covariance matrices Σ_k to be 2×2 identity matrices. Refer to Lecture 13 (28-MArch-2022) Page 18-19.

Fill in the table below after for each iteration. You may do the intermediate calculations on scratch paper using either a calculator or a computer program, but please show how you get the numbers. Provide the class assignments, in addition to the means and covariance matrices for the two mixtures in each iteration.

The posterior for the datapoint $\mathbf{x}_i$ is obtained using the following formula:

$$p(\mathbf{u}_k, \Sigma_k \,|\mathbf{x}_i) = \frac{p(\mathbf{u}_k, \Sigma_k) \times \mathcal{N}(\mathbf{x}_i; \mathbf{u}_k, \Sigma_k)}{\sum_{k=1}^{K} p(\mathbf{u}_k, \Sigma_k) \times \mathcal{N}(\mathbf{x}_i; \mathbf{u}_k, \Sigma_k)}$$

where $p(\mathbf{u}_k, \Sigma_k)$ is the prior for k-th mixture, $\mathcal{N}(\mathbf{x}_i; \mathbf{u}_k, \Sigma_k)$ is the multivariate normal distribution given as following:

$$\mathcal{N}(\mathbf{x}_i; \mathbf{u}_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)^2 |\Sigma_k|}} \exp(-\frac{1}{2}(\mathbf{x}_i - \mathbf{u}_k)^T \Sigma_k^{-1} (\mathbf{x}_i - \mathbf{u}_k))$$

(all numbers round to at least 4 decimals.)

Problem 1 (a) Solution

First iteration:

Point Distances $\|\mathbf{x}_i - \mathbf{u}_k\|_2^2$			Class Assignments $\underset{k}{\operatorname{argmin}} \|\mathbf{x}_i - \mathbf{u}_k\|_2^2$		Class Means $\mathbf{u}_k$	
Datapoint ($\mathbf{x}_i$)	**Distance from $\mathbf{u}_1$**	**Distance from $\mathbf{u}_2$**	**Datapoint ($\mathbf{x}_i$)**	**Cluster Assignment (1 or 2)**	**K-Means Parameters**	
1	0	2.8125	1	1	Mean Cluster 1 ($\mathbf{u}_1$)	[0.25, 0.25]

Datapoint ($\mathbf{x}_i$)	Distance from $\mathbf{u}_1$	Distance from $\mathbf{u}_2$
2	8.5	3.0625
3	0.5	1.0625
4	2.8125	0
5	1	0.8125

Datapoint ($\mathbf{x}_i$)	Cluster Assignment (1 or 2)
2	2
3	1
4	2
5	2

K-Means Parameters	
Mean Cluster 2 ($\mathbf{u}_2$)	[1.0833, 1.3333]

Second iteration:

Point Distances $\|\mathbf{x}_i - \mathbf{u}_k\|_2^2$

Datapoint ($\mathbf{x}_i$)	Distance from $\mathbf{u}_1$	Distance from $\mathbf{u}_2$
1	0.125	2.9514
2	6.625	2.0347
3	0.125	1.0347
4	1.8125	0.1389
5	0.625	1.2847

Class Assignments $\underset{k}{\operatorname{argmin}} \|\mathbf{x}_i - \mathbf{u}_k\|_2^2$

Datapoint ($\mathbf{x}_i$)	Cluster Assignment (1 or 2)
1	1
2	2
3	1
4	2
5	1

Class Means $\mathbf{u}_k$

K-Means Parameters	
Mean Cluster 1 ($\mathbf{u}_1$)	[0.1667, 0.5]
Mean Cluster 2 ($\mathbf{u}_2$)	[1.625, 1.5]

```
# Problem 1(a)
print("1st Iteration:")
x = np.array([[0,0],[2.5,1.5],[0.5,0.5],[0.75,1.5],[0,1]])
u1 = np.array([0,0])
u2 = np.array([0.75, 1.5])
dist1 = np.linalg.norm(x - u1, axis=1) ** 2
print("Distance 1: \n", dist1)
dist2 = np.linalg.norm(x - u2, axis=1) ** 2
print("Distance 2: \n", dist2)

u1_new = np.array([(x[0,:]+x[2,:])/2])
print("Mean Cluster 1: \n",u1_new)
u2_new = np.array([(x[1,:]+x[3,:]+x[4,:])/3])
print("Mean Cluster 2: \n",u2_new)
print("\n")
print("2nd Iteration:")
dist1 = np.linalg.norm(x - u1_new, axis=1) ** 2
print("Distance 1: \n", dist1)
dist2 = np.linalg.norm(x - u2_new, axis=1) ** 2
print("Distance 2: \n", dist2)

u1_new = np.array([(x[0,:]+x[2,:]+x[4,:])/3])
```

```
print("Mean Cluster 1: \n",u1_new)
u2_new = np.array([(x[1,:]+x[3,:])/2])
print("Mean Cluster 2: \n",u2_new)
```

```
1st Iteration:
Distance 1:
 [0.     8.5    0.5    2.8125 1.    ]
Distance 2:
 [2.8125 3.0625 1.0625 0.     0.8125]
Mean Cluster 1:
 [[0.25 0.25]]
Mean Cluster 2:
 [[1.08333333 1.33333333]]


2nd Iteration:
Distance 1:
 [0.125  6.625  0.125  1.8125 0.625 ]
Distance 2:
 [2.95138889 2.03472222 1.03472222 0.13888889 1.28472222]
Mean Cluster 1:
 [[0.16666667 0.5       ]]
Mean Cluster 2:
 [[1.625 1.5  ]]
```

Problem 1 (b) Solution

First iteration:

Point Posteriors $p(\mathbf{u}_k, \Sigma_k | \mathbf{x}_i)$

Datapoint	Posterior Mixture 1	Posterior Mixture 2
1	0.8032	0.1968
2	0.0619	0.9381
3	0.5699	0.4301
4	0.1968	0.8032
5	0.4766	0.5234

Cluster Assignment $\underset{k}{\operatorname{argmax}}\, p(\mathbf{u}_k, \Sigma_k | \mathbf{x}_i)$

Datapoint ($\mathbf{x}_i$)	Cluster Assignment (1 or 2)
1	1
2	2
3	1
4	2
5	2

GMM Parameters ($\mathbf{u}_k, \Sigma_k$)

GMM Parameters	
Cluster 1 prior	0.4217
Mean Cluster 1 ($\mathbf{u}_1$)	[0.2786, 0.5453]
Covariance Cluster 1 (Σ_1)	[0.2260, 0.1308; 0.1308, 0.2724]
Cluster 2 prior	0.5783
Mean Cluster 2 ($\mathbf{u}_2$)	[1.0938, 1.1586]
Covariance Cluster 2 (Σ_2)	[1.0248, 0.2990; 0.2990, 0.2306]

Second iteration:

Point Posteriors $p(\mathbf{u}_k, \Sigma_k | \mathbf{x}_i)$

Datapoint	Posterior Mixture 1	Posterior Mixture 2
1	0.9414	0.0586
2	0.0000	0.9999
3	0.7485	0.2515
4	0.3421	0.6879
5	0.4549	0.5451

Cluster Assignment $\underset{k}{\operatorname{argmax}}\, p(\mathbf{u}_k, \Sigma_k | \mathbf{x}_i)$

Datapoint ($\mathbf{x}_i$)	Cluster Assignment (1 or 2)
1	1
2	2
3	1
4	2
5	2

GMM Parameters $(\mathbf{u}_k, \Sigma_k)$

GMM Parameters	
Cluster 1 prior	0.4974
Mean Cluster 1 ($\mathbf{u}_1$)	[0.2537,0.5397]
Covariance Cluster 1 (Σ_1)	[0.0884, 0.0931; 0.0931,0.2763]
Cluster 2 prior	0.0938
Mean Cluster 2 ($\mathbf{u}_2$)	[1.2412,1.2566]
Covariance Cluster 2 (Σ_2)	[1.1187,0.2522; 0.2522,0.1475]

```
x = np.array([[0,0],[2.5,1.5],[0.5,0.5],[0.75,1.5],[0,1]])
u1 = np.array([0,0])
u2 = np.array([0.75, 1.5])
p1 = 0.5
p2 = 0.5
sigma = np.eye(2)

print("First Iteration: ")
N1 = np.zeros((5,1))
N2 = np.zeros((5,1))
for i in range(5):
  N1[i,0]=(np.exp(-0.5* np.reshape((x[i,:]-u1),[1,-1]) @ np.linalg.inv(sigma) @ np.transpose(np.reshape((x[i,:]-u1),[1,-1]))) / np.sqr
  N2[i,0]=(np.exp(-0.5* np.reshape((x[i,:]-u2),[1,-1]) @ np.linalg.inv(sigma) @ np.transpose(np.reshape((x[i,:]-u2),[1,-1]))) / np.sqr

post_1=p1*N1/(p1*N1+p2*N2)
post_2=p2*N2/(p1*N1+p2*N2)

print("Posterior 1: \n", post_1.T)
print("Posterior 2: \n", post_2.T)
p1 = np.mean(post_1)
print("Prior 1: ", p1)
p2 = np.mean(post_2)
print("Prior 2: ", p2)
u1 = np.array([np.sum(post_1*np.reshape(x[:,0],[-1,1]))/np.sum(post_1), np.sum(post_1*np.reshape(x[:,1],[-1,1]))/np.sum(post_1)])
print("Mean 1: ", u1)
u2 = np.array([np.sum(post_2*np.reshape(x[:,0],[-1,1]))/np.sum(post_2), np.sum(post_2*np.reshape(x[:,1],[-1,1]))/np.sum(post_2)])
print("Mean 2: ", u2)
```

```
sig1 = np.array([[0,0],[0,0]])
for i in range(5):
  arr = np.transpose((np.reshape(x[i,:],[1,-1])-u1)) @ (np.reshape(x[i,:],[1,-1])-u1)
  sig1 = sig1 + post_1[i]*arr/np.sum(post_1)
print("Sigma 1: \n", sig1)

sig2 = np.array([[0,0],[0,0]])
for i in range(5):
  arr = np.transpose((np.reshape(x[i,:],[1,-1])-u2)) @ (np.reshape(x[i,:],[1,-1])-u2)
  sig2 = sig2 + post_2[i]*arr/np.sum(post_2)
print("Sigma 2: \n", sig2)



print("\nSecond Iteration: ")
N1 = np.zeros((5,1))
N2 = np.zeros((5,1))
for i in range(5):
  N1[i,0]=(np.exp(-0.5* np.reshape((x[i,:]-u1),[1,-1]) @ np.linalg.inv(sig1) @ np.transpose(np.reshape((x[i,:]-u1),[1,-1]))) / np.sqrt
  N2[i,0]=(np.exp(-0.5* np.reshape((x[i,:]-u2),[1,-1]) @ np.linalg.inv(sig2) @ np.transpose(np.reshape((x[i,:]-u2),[1,-1]))) / np.sqrt

post_1=p1*N1/(p1*N1+p2*N2)
post_2=p2*N2/(p1*N1+p2*N2)
print("Posterior 1: \n", post_1.T)
print("Posterior 2: \n", post_2.T)

p1 = np.mean(post_1)
print("Prior 1: ", p1)
p2 = np.mean(post_2)
print("Prior 2: ", p2)
u1 = np.array([np.sum(post_1*np.reshape(x[:,0],[-1,1]))/np.sum(post_1), np.sum(post_1*np.reshape(x[:,1],[-1,1]))/np.sum(post_1)])
print("Mean 1: ", u1)
u2 = np.array([np.sum(post_2*np.reshape(x[:,0],[-1,1]))/np.sum(post_2), np.sum(post_2*np.reshape(x[:,1],[-1,1]))/np.sum(post_2)])
print("Mean 2: ", u2)

sig1 = np.array([[0,0],[0,0]])
for i in range(5):
  arr = np.transpose((np.reshape(x[i,:],[1,-1])-u1)) @ (np.reshape(x[i,:],[1,-1])-u1)
  sig1 = sig1 + post_1[i]*arr/np.sum(post_1)
print("Sigma 1: \n", sig1)

sig2 = np.array([[0,0],[0,0]])
```

```
for i in range(5):
  arr = np.transpose((np.reshape(x[i,:],[1,-1])-u2)) @ (np.reshape(x[i,:],[1,-1])-u2)
  sig2 = sig2 + post_2[i]*arr/np.sum(post_2)
print("Sigma 2: \n", sig2)
```

```
First Iteration:
Posterior 1:
 [[0.8031738  0.06187599 0.56985265 0.1968262  0.47657965]]
Posterior 2:
 [[0.1968262  0.93812401 0.43014735 0.8031738  0.52342035]]
Prior 1:  0.42166165764029523
Prior 2:  0.5783383423597048
Mean 1:  [0.27853419 0.54525198]
Mean 2:  [1.09374179 1.15864382]
Sigma 1:
 [[0.22593353 0.13078544]
 [0.13078544 0.27240954]]
Sigma 2:
 [[1.02478074 0.2989791 ]
 [0.2989791  0.23062963]]

Second Iteration:
Posterior 1:
 [[9.41403262e-01 4.83736941e-05 7.48491912e-01 3.42113080e-01
  4.54943313e-01]]
Posterior 2:
 [[0.05859674 0.99995163 0.25150809 0.65788692 0.54505669]]
Prior 1:  0.4973999882726206
Prior 2:  0.5026000117273793
Mean 1:  [0.25369993 0.53977945]
Mean 2:  [1.24116523 1.25649362]
Sigma 1:
 [[0.08837617 0.093127  ]
 [0.093127   0.27636215]]
Sigma 2:
 [[1.11873526 0.25219025]
 [0.25219025 0.1474754 ]]
```

Problem 2: K-Means vs GMMs for Modeling Non-Spherical Distributions (15pts)

K-means is good when finding clusters of data sampled from gaussian distributions with zero correlation (and ideally equal variances in all feature directions). In cases where this is not true (i.e., gaussian distributions have correlated features and/or unequal variances), GMMs tend to perform superior to K-means, as you will hopefully see in this question.

As seen in Problem 1 above, running both K-means and GMMs requires the setup of an $N\times K$ dimensional table for each iteration, storing the point distances to the indiviudal cluster means for the former and point posteriors for the latter. N refers to the number of training examples while K is the number of clusters. The **un-normalized** posterior for the i-th datapoint having mean and covariance $\mathbf{u}_{k}$ and Σ_{k} is given as:

\begin{align} p(\mathbf{u}_{k},\Sigma_{k}\,|\mathbf{x}_{i}) &= \overbrace{p(\mathbf{u}_{k},\Sigma_{k})}^{\text{prior for mixture }k}\times \overbrace{p(\mathbf{x}_{i}|\mathbf{u}_{k},\Sigma_{k})}^{\text{likelihood for }\mathbf{x}_{i}\text{ given mxiture }k} \tag{4} \\ \\ p(\mathbf{u}_{k},\Sigma_{k}\,|\mathbf{x}_{i}) &= p(\mathbf{u}_{k},\Sigma_{k}) \times \mathcal{N}(\mathbf{x}_{i};\mathbf{u}_{k},\Sigma_{k}), \tag{5} \end{align}

where $\mathcal{N}(\mathbf{x}_{i};\mathbf{u}_{k},\Sigma_{k})$ is the multivariate normal distribution characterized by mean $\mathbf{u}_{k}$ and covariance Σ_{k} and

$$\mathcal{N}(\mathbf{x}_{i};\mathbf{u}_{k},\Sigma_{k}) = \frac{1}{\sqrt{(2\pi)^{P}|\Sigma|}}\text{exp}(-\frac{1}{2}(\mathbf{x} - \mathbf{u})^{T}\Sigma^{-1}(\mathbf{x} - \mathbf{u})) \tag{6}$$

(a)

If we want to normalize Equation 4, what is the formula of denominator? After normalization, what is the range for normalized posterior?

(b)

As we have seen in Regression, it is often more convenient to compute these values in the natural log scale. In this part, take the log of both sides of Equation 4. Write down the resulting equation below.

(c)

Plug Equation 6 into the equation you derived in part (b). Simplify and write down a fully expanded expression for $\text{log}\,p(\mathbf{u}_{k},\Sigma_{k}\,|\mathbf{x}_{i})$.

(d)

Considering the expression in part (c), it is possible to simplify the expression to represent K-means. In other words, K-means can be a special case of GMM. Explain exactly under what constraints and changes, if any, on the prior, means, covariance matrices, and the update process does this statement become true, i.e., K-means is a sepcial case of GMM.

(e)

The code below generates some data sampled from two $2-D$ gaussian distributions with different means and covariance matrices.

Using the `sklearn.cluster.KMeans` class for K-means and `sklearn.mixture.GaussianMixture` class for GMMs, set up a class object for each to run for 2 clusters on this data. Describe the results. Which one of K-means and GMMs captures the original distribution better? How do you relate this to what you explained in part (d)?

Note: To discount the effect of random initialization, you might have to execute the cell multiple times to get a consistent idea of the results.

Problem 2 (a) Solution

- Denominator = $\sum_{k=1}^{K} p(\mathbf{u}_k, \Sigma_k) \times p(\mathbf{x}_i; \mathbf{u}_k, \Sigma_k)$ for equation 4
- The range of normalized postirior will be in 0 ~ 1.

Problem 2 (b) Solution

+ Code + Text

$$\log p(\mathbf{u}_k, \Sigma_k \,|\mathbf{x}_i) = \log(p(\mathbf{u}_k, \Sigma_k) \times p(\mathbf{x}_i; \mathbf{u}_k, \Sigma_k)) = \log(p(\mathbf{u}_k, \Sigma_k)) + \log(p(\mathbf{x}_i; \mathbf{u}_k, \Sigma_k))$$

Problem 2 (c) Solution

$$\log p(\mathbf{u}_k, \Sigma_k \,|\mathbf{x}_i) = \log(p(\mathbf{u}_k, \Sigma_k)) + \log(\mathcal{N}(\mathbf{x}_i; \mathbf{u}_k, \Sigma_k)) = \log(p(\mathbf{u}_k, \Sigma_k)) + \log(\frac{1}{\sqrt{(2\pi)^P |\Sigma|}} \exp(-\frac{1}{2}(\mathbf{x}-\mathbf{u})^T \Sigma^{-1} (\mathbf{x}-\mathbf{u})))$$

Therefore,

$$\log p(\mathbf{u}_k, \Sigma_k \,|\mathbf{x}_i) = \log(p(\mathbf{u}_k, \Sigma_k)) - 0.5 * (\log((2\pi)^P) + \log|\Sigma|) - \frac{1}{2}(\mathbf{x}-\mathbf{u})^T \Sigma^{-1} (\mathbf{x}-\mathbf{u})$$

Problem 2 (d) Solution

Problem 2 (d) Solution

- KMeans is a hard-assignment version of GMM that has an identity covariance since it is only looking at the fix distances between points.
- GMM weight the distances by multiplying the Gaussian distribution of different covariances and thus will provide a more accurate prediction.

Problem 2 (e) Solution

From the plot below we can see that GMM has a better performance of outlining the correct Gaussian distribution. This matches our observation at Problem 2(d) of KMeans having a fix covariance and will be less accurate.

```
#### Do not change this cell. This a helper cell. Please execute it.

# imports and utlity functions
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import torch
from itertools import cycle
from sklearn import metrics
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.datasets import load_iris, load_wine
import random
from skimage import data, color


# generate colors for clustering via python generator
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k']
color_generator = cycle(colors)

# helper functions
```

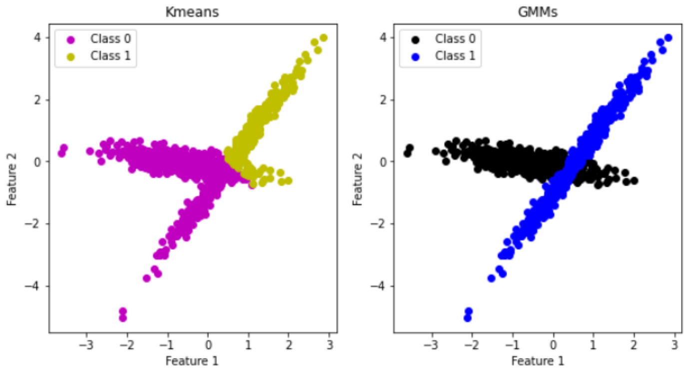
```
# evaluation metrics

# Dunn Index
# from the resource: https://en.wikipedia.org/wiki/Cluster_analysis
# the distance between two clusters can be any of the measurements, i.e. distance between centroids or any points.

def delta(ck, cl):
    values = np.ones([len(ck), len(cl)]) * np.finfo(np.float32).max
    for i in range(len(ck)):
        for j in range(len(cl)):
            values[i, j] = np.linalg.norm(ck[i] - cl[j])

    return np.min(values)


def big_delta(ci):
    values = np.zeros([len(ci), len(ci)])

    for i in range(0, len(ci)):
        for j in range(0, len(ci)):
            values[i, j] = np.linalg.norm(ci[i] - ci[j])

    return np.max(values)

# Dunn Index
def dunn_index(X, cluster_labels):
    # A list containing a numpy array for each cluster
    # k_list[k] is np.array([N, p]) (N : number of samples in cluster k, p : sample dimension)
    k_list = []
    for k in np.unique(cluster_labels):
        k_list.append(X[cluster_labels == k])

    deltas = np.ones([len(k_list), len(k_list)]) * np.finfo(np.float32).max
    big_deltas = np.zeros([len(k_list), 1])
    l_range = list(range(0, len(k_list)))

    for k in l_range:
        for l in (l_range[0:k] + l_range[k + 1:]):
            deltas[k, l] = delta(k_list[k], k_list[l])

        big_deltas[k] = big_delta(k_list[k])

    di = np.min(deltas) / np.max(big_deltas)
```

```python
    di = np.min(deltas) / np.max(big_deltas)

    return di


def delta_fast(ck, cl, distances):
    values = distances[np.where(ck)][:, np.where(cl)]
    values = values[np.nonzero(values)]

    return np.min(values)


def big_delta_fast(ci, distances):
    values = distances[np.where(ci)][:, np.where(ci)]

    return np.max(values)


def dunn_index_fast(X, cluster_labels):
    """Dunn Index - fast(using sklearn pairwise euclidean_distance function
    X: np.array
    np.array([N,p] of all samples
    cluster_labels: np.array
    np.array([N,]) labels of all samples
    """
    distances = euclidean_distances(X)
    ks = np.sort(np.unique(cluster_labels))

    deltas = np.ones([len(ks), len(ks)]) * np.finfo(np.float32).max
    big_deltas = np.zeros([len(ks), 1])
    l_range = list((range(0, len(ks))))

    for k in l_range:
        for l in (l_range[0:k] + l_range[k+1:]):
            deltas[k, l] = delta_fast((cluster_labels==ks[k]), (cluster_labels==ks[l]), distances)

        big_deltas[k] = big_delta_fast(cluster_labels == ks[k], distances)

    di = np.min(deltas) / np.max(big_deltas)
    return di


# Cluster Purity
```

```
def purity_score(labels_true, labels_pred):
    # compute contingency matrix
    contingency_matrix = metrics.cluster.contingency_matrix(labels_true, labels_pred)

    return np.sum(np.amax(contingency_matrix, axis=0)) / np.sum(contingency_matrix)


# for Problem 2 (e)

from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture

mu_1 = np.array([0,0])
mu_2 = np.array([0,0])
cov1 = np.array([[1.95, 1],[1, 0.5]])
theta = np.radians(30)
c, s = np.cos(theta), np.sin(theta)
R = np.array(((c, -s), (s, c)))
X_1 = np.random.multivariate_normal(mu_1, cov1, 500)
X_2 = X_1.dot(R) - np.array([2,0]).reshape(1,-1)
X = np.concatenate((X_1, X_2), axis=0)
X = (X - X.mean(axis=0)) / X.std(axis=0)

#-----------------Don't change anything above------------------------#
# print(np.shape(X))
model_kmeans = KMeans(n_clusters=2, random_state=0) ##TODO
model_gmms = GaussianMixture(n_components=2, random_state=0) ##TODO
y_pred_kmeans = model_kmeans.fit_predict(X)
y_pred_gmms = model_gmms.fit_predict(X)
#-----------------Don't change anything below------------------------#

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))

for cluster_id, color in zip(range(2), color_generator):
  data_x = X[y_pred_kmeans==cluster_id,0]
  data_y = X[y_pred_kmeans==cluster_id,1]
  ax1.scatter(data_x, data_y, color = color, label='Class {}'.format(cluster_id))
  ax1.legend()
  ax1.set_xlabel('Feature 1')
  ax1.set_ylabel('Feature 2')
  ax1.set_title('Kmeans')


for cluster_id, c in zip(range(2), color_generator):
```

```
for cluster_id, c in zip(range(2), color_generator):
  data_x = X[y_pred_gmms==cluster_id,0]
  data_y = X[y_pred_gmms==cluster_id,1]
  ax2.scatter(data_x, data_y, color = c, label='Class {}'.format(cluster_id))
  ax2.legend()
  ax2.set_xlabel('Feature 1')
  ax2.set_ylabel('Feature 2')
  ax2.set_title('GMMs')


plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: RuntimeWarning: covariance is not positive-semidefinite.
  if sys.path[0] == '':
```



Problem 3: Implement K-Means (20pts)

In this problem, you are going to design a python class that implements the k-means algorithm. You are provided with a class template called `MyKmeans` that you have to fill in to implement various stages in the k-means workflow. Follow the steps outlined in the parts below to answer the question.

(a)

The initialization function in the class stores the training data (`X_train`), the number of clusters (`k`), and the number of iterations to run the k-means algorithm for (`num_iter`). The first step in the algorithm entails randomly selecting $k$$k$ data points in `X_train` to be the centroids, one for each cluster. Fill in code for the class function `__init_means()` below to do this. The output should be stored into the `self.means` variable to be used later in the fitting stage. Pay attention to the shape of this array.

(b)

Implement the `fit_predict()` function in the class definition by writing code to execute the Assignment and Update steps, as described in Page 34 of lecture 11. For each iteration, the assignment step involves computing the euclidean distance of each data point in `X_train` to each of the $k$$k$ centroids selected in part(a) above. The result is essentially an $N\times K$$N\times K$ dimensional table where each column stores the euclidean distance of all data points to the centroid corresponding to the column. This is used to compute the cluster label for each datapoint by choosing the centroid (where centroid label $\in[1,K]$$\in[1,K]$) corresponding to the smallest distance, resulting in a $N\times 1$$N\times 1$ array.

(c)

Next, implement the Update step, where the $N\times 1$$N\times 1$ label vector just created in part (b) is used to recompute the means for each of the $k$$k$ centroids, and thus update the `self.means` structure. Refer to Page 34 in Lecture 11 (21-Feb-2022) for the details.

Execute the cell once you have completed all of the above to classify the `Iris` dataset you used in homework 1 using **two** preselected features. You may have to run the cell multiple times to discount the effect of random initialization and get consistent results.

Plot the clustering results when using features $0$$0$ and $2$$2$.

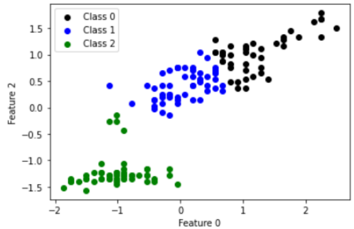
Problem 3 (a)(b)(c) Solution

```
from sklearn.metrics import davies_bouldin_score, mutual_info_score, adjusted_rand_score

class MyKMeans:
  def __init__(self, X_train, k, num_iter=20):
    """
```

```
    Parameters
    ----------
    X_train: ndarray of shape (number of samples, num of features).
      Training data array.

    k: int,
      number of clusters.

    num_iter:int
      number of steps to run algorithm for.
    """

    self.X_train = X_train
    self.k = k
    self.num_iter = num_iter

  def __init_means(self):
    """initialize means as an ndarray of shape (k, num of features)."""
    ## part (a)

    ind = random.sample(range(0, self.X_train.shape[0]), self.k)
    # print(ind)
    self.means = self.X_train[ind]

    ##TODO

  def fit_predict(self):
    """Runs the k means algorithm.

    Returns
    -------
    y_pred: ndarray of shape (num of samples, 1)
      array of predicted cluster labels for each data point.
    """

    self.__init_means()  # initialize means

    for iteration in range(self.num_iter):      # begin the algorithm

      # assignment step
      ## part (b)
      ##TODO
```

```
      label = np.zeros(np.shape(self.X_train)[0])
      for n in range(np.shape(self.X_train)[0]):
        x = np.reshape(self.X_train[n,:],[1,-1])
        dist = np.linalg.norm(x - self.means, axis=1)
        label[n] = np.argmin(dist)

      # update means step
      ## part (c)
      ##TODO
      label = label.astype(int)
      count = np.zeros((1,self.k))
      k = self.k

      for id in range(k):
        count[0,id] = np.sum(label==id)
      new_means = []

      for j in range(self.k):
        new_means.append(np.sum(self.X_train[label==j, :], axis = 0) / count[0,j])
      self.means = np.asarray(new_means)



    # Final class assignments
    ## part (c)
    label = np.zeros(np.shape(self.X_train)[0])
    for n in range(np.shape(self.X_train)[0]):
      x = np.reshape(self.X_train[n,:],[1,-1])
      dist = np.linalg.norm(x - self.means, axis=1)
      label[n] = np.argmin(dist)
    y_pred = label.astype(int) ##TODO


    return y_pred

#-----------------Don't change anything below-----------------------#

# k means parameters
k = 3
num_iter = 20
feature_nums = [0,2]  # features to use
```

```
# load and preprocess the dataset
dataset = load_iris()
X, y = dataset.data, dataset.target
X = (X - X.mean(axis=0)) / X.std(axis=0)
X = X[:,feature_nums]

# fit the model
kmeans = MyKMeans(X, k=k, num_iter=num_iter)
y_pred = kmeans.fit_predict()


# plot data
fig, ax = plt.subplots()

for cluster_id in range(k):
  data_x = X[y_pred==cluster_id,0]
  data_y = X[y_pred==cluster_id,1]
  ax.scatter(data_x, data_y, color = next(color_generator), label='Class {}'.format(cluster_id))
  ax.legend()

plt.xlabel('Feature {}'.format(feature_nums[0]))
plt.ylabel('Feature {}'.format(feature_nums[1]))
plt.show()

print('Davies Bouldin Score: {:0.4f}'.format(davies_bouldin_score(X, y_pred)))
print('Dunn Index: {:0.4f}'.format(dunn_index(X, y_pred)))
print('Mutual Information Score: {:0.4f}'.format(mutual_info_score(y, y_pred)))
print('Rand Index: {:0.4f}'.format(adjusted_rand_score(y, y_pred)))
print('Purity Score: {:0.4f}'.format(purity_score(y, y_pred)))
```




```
Davies Bouldin Score: 0.6373
Dunn Index: 0.0672
Mutual Information Score: 0.6401
Rand Index: 0.5634
Purity Score: 0.8067
```

Problem 4: Implementing Gaussian Mixture Models (20pts)

In this problem, you will create a `MyGMMs` class that implements clustering using Gaussian Mixture Models (GMM). The class intialization function stores the training data array, `X_train`, the prespecified number of mixtures, `k`, and the number of iterations, `num_iteratons`. The means and covariances of the $k$$k$-th cluster and the posteriors structure have already been initalized in the `__init_params()` function. Read every question very carefully before you start your solution.

(a)

Expectation Step: The analogue of the $N\times K$$N\times K$ dimesnional cluster distances table created in Problem 1 (b) above is the posteriors structure in GMMs. The entries $\gamma_{ik}$$\gamma_{ik}$ in this table store the (unnormalized) posterior probabilities of the parameters of the $k$$k$-th mixture given a datapoint $\mathbf{x}_{i}$$\mathbf{x}_{i}$ (i.e., $\gamma_{ik} = p(\mathbf{u}_{k},\Sigma_{k}\,|\mathbf{x}_{i})$$\gamma_{ik} = p(\mathbf{u}_{k},\Sigma_{k}\,|\mathbf{x}_{i})$). Using the expression you derived in problem 2 (b), fill in the `fit_predict()` function below to compute the expectation step. We work with natural logs because they are numerically easier to deal with from the computer's point of view. Remember also that since you are working with logs, so you need to take the exponent of the final value and normalize before storing it as a posterior.

(b)

Maximization Step: Having completed the expectation step in the E-M algorithm you studied in class, you now have a complete $N\times K$$N\times K$ dimensional table of posterior values. We now move on to the maximization step where we are required to update the priors

($p(\mathbf{u}_{k},\Sigma_{k})$), the means ($\mathbf{u}_{k}$), and the covariances ($\Sigma_{k}$) for each of the K clusters.

(i) Using page 19 of Lecture 13 (28-Feb-2022) as a reference, write down the expression for the mean of the k-th mixture in terms of γ_{ik} we computed in (a).

(ii) Again using page 19 as the reference, write down the expression for the covariance of the k-th mixture in terms of γ_{ik} and $\mathbf{u}_{k}$ we computed above.

(iii) Once again using page 19 as a reference, write down the expression for the new prior of the j-th mixture in terms of γ_{ik}.

(iv) Code steps (i - iii) into the `fit_predict()` function in the class template below. This completes the maximization step of the iteration.

Further, complete the code to calculate the final assignments using the latest posterior table. Do not forget to update the `self.parameter` and `self.priors` variables before the function returns the labels.

Execute the cell multiple times until you are able to get an idea of what the consistent results look like.

Plot the clustering results when using features 0 and 2.

Practical Tips:

1. The priors in equation 1 can underflow to zero in the log expression, resulting in nans. You may want to add a small value e.g., $1e-4$ to compensate for that.
2. In case the covariance matrix turns out to be singular or badly conditioned, taking the inverse will result in an error. You may want to use `numpy.linalg.pinv` rather than `numpy.linalg.inv`. This will calcualte the pseudo inverse.
3. You can improve the conditioning of the covariance matrices in the maximization step by adding an identity matrix scaled by a small number e.g. 1e-4

Problem 4 (a) Solution

Complete code cell below.

Problem 4 (b)(i) Solution

Problem 4 (b)(i) Solution

$$\mu_k = \frac{\sum_{i=1}^N \gamma_{ik} x_i}{\sum_{i=1}^N \gamma_{ik}}$$

Problem 4 (b)(ii) Solution

$$\pi_k = \frac{\sum_{i=1}^N \gamma_{ik}}{N}$$

Problem 4 (b)(iii) Solution

$$\Sigma_k = \frac{\sum_{i=1}^N \gamma_{ik} (x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{i=1}^N \gamma_{ik}}$$

Problem 4 (b)(iv) Solution

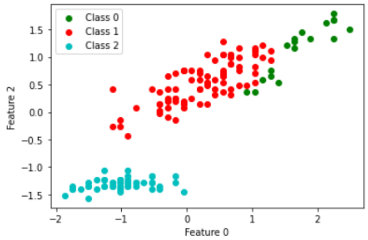
```
# for problem 4
from sklearn.metrics import davies_bouldin_score, mutual_info_score, adjusted_rand_score

class MyGMMs:
  def __init__(self, X_train, k, num_iter=20):
    """
    Parameters
    ----------
    X_train: ndarray of shape (number of samples, num of features).
      Training data array.

    k: int,
      number of clusters.

    num_iter: int
      number of iteration to run E-M algorithm for.
    """

    self.X_train = X_train
```

```
# expectation step to compute NxK dimensional array storing posterior values

  for mixture_id in range(k):
    ## part (a)
    ##TODO
    log_prior = np.log(self.priors[mixture_id])
    u = np.asarray(self.parameters[mixture_id][0])
    sigma = np.asarray(self.parameters[mixture_id][1])
    pi = np.pi
    second_term = np.log(np.sqrt(4*pi*pi)*np.linalg.det(sigma))
    # second_term = 0.5*np.log(4*pi*pi) + 0.5*np.log(np.linalg.det(sigma))
    X_train = self.X_train

    for n in range(np.shape(X_train)[0]):
      arr = np.reshape(X_train[n,:],[-1,1])- u
      third_term = np.transpose(arr)@np.linalg.pinv(sigma)@arr
      log_L = log_prior - second_term - 0.5*third_term
      Likelihood = np.exp(log_L)
      self.posteriors[n,mixture_id] = Likelihood

    self.posteriors = self.posteriors/np.sum(self.posteriors,axis=1)[:,None]

  posteriors = self.posteriors
  label = np.argmax(self.posteriors, axis=1)
  label = label.astype(int)
  count = np.bincount(label)
  new_parameters = new_parameters = [[] for i in range(k)]
  new_priors = np.sum(self.posteriors, axis = 0)/np.shape(self.posteriors)[0]
  # new_sigmas = []

  for mixture_id in range(k):
    ## part (b)
    ##TODO
    Denominator = np.sum(posteriors[:,mixture_id])

    u_nom = np.transpose(np.reshape(np.matmul(posteriors[:,mixture_id],X_train),[1,-1]))

    new_parameters[mixture_id].append(u_nom/Denominator)

    sigma_nom = 0

    for n in range(np.shape(X_train)[0]):
```

```
        self.k = k
        self.num_iter = num_iter


    def __init_params(self):
        """Function initializes the means, covariances, and posteriors structure for
        the E-M algorithm"""


        # extract k and data matrices
        k = self.k
        X_train = self.X_train

        # Initialize priors as uniform
        self.priors = 1/k * np.ones((k,1))

        # intialize means and covariances
        self.parameters = [[] for i in range(k)]
        for i in range(k):
            self.parameters[i].append(np.random.randn(X_train.shape[1],1))  # initialize random means
            temp = np.random.randn(X_train.shape[1],X_train.shape[1])
            self.parameters[i].append(temp.T.dot(temp)+1e-4*np.eye(X_train.shape[1]))  # initialize random covariances

        # set up posterior structure
        self.posteriors = np.zeros((X_train.shape[0], k))


    def fit_predict(self):
        """ Returns predicted cluster classes.

        Returns
        -------
        y_pred: ndarray of shape (number of samples, 1).
          Predicted classes for each data point in X_train.
        """

        self.__init_params()
        # print(self.parameters[0][1])
        k = self.k

        # begin the E-M Algorithm
        for iteration in range(self.num_iter):
```

```
      arr = np.reshape(X_train[n,:],[-1,1])- u_nom/Denominator
      square_arr = arr@np.transpose(arr)
      sigma_nom = sigma_nom + posteriors[n,mixture_id]*square_arr


    new_parameters[mixture_id].append(sigma_nom/Denominator+(1e-4)*np.eye(np.shape(sigma_nom/Denominator)[0]))


  self.priors = new_priors
  self.parameters = new_parameters



# compute final class predictions
label = np.argmax(self.posteriors, axis=1)
label = label.astype(int)
y_pred = label   ##TODO

# store distribution paramteres
new_parameters = new_parameters = [[] for i in range(k)]
new_priors = np.sum(self.posteriors, axis = 0)/np.shape(self.posteriors)[0]
# new_sigmas = []

for mixture_id in range(k):
  ## part (b)
  ##TODO
  Denominator = np.sum(posteriors[:,mixture_id])

  np.transpose(np.reshape(np.matmul(posteriors[:,mixture_id],X_train),[1,-1]))

  new_parameters[mixture_id].append(u_nom/Denominator)

  sigma_nom = 0

  for n in range(np.shape(X_train)[0]):
    arr = np.reshape(X_train[n,:],[-1,1])- u_nom/Denominator
    square_arr = arr@np.transpose(arr)
    sigma_nom = sigma_nom + posteriors[n,mixture_id]*square_arr

  new_parameters[mixture_id].append(sigma_nom/Denominator)
self.priors = new_priors ##TODO
self.parameters = new_parameters ##TODO
```

```
    return y_pred

#-----------------Don't change anything below------------------------#

# k means parameters
k = 3                     # num of mixtures
num_iter = 20
feature_nums = [0,2]  # features to use

# load and preprocess the dataset
dataset = load_iris()
X, y = dataset.data, dataset.target
X = (X - X.mean(axis=0)) / X.std(axis=0)
X = X[:, feature_nums]
# print(np.shape(X))
gmm = MyGMMs(X, k, num_iter=num_iter)
y_pred = gmm.fit_predict()

# plot data
fig, ax = plt.subplots()

for cluster_id in range(k):
  data_x = X[y_pred==cluster_id,0]
  data_y = X[y_pred==cluster_id,1]
  ax.scatter(data_x, data_y, color = next(color_generator), label='Class {}'.format(cluster_id))
  ax.legend()

plt.xlabel('Feature {}'.format(feature_nums[0]))
plt.ylabel('Feature {}'.format(feature_nums[1]))
plt.show()

print('Davies Bouldin Score: {:0.4f}'.format(davies_bouldin_score(X, y_pred)))
print('Dunn Index: {:0.4f}'.format(dunn_index(X, y_pred)))
print('Mutual Information Score: {:0.4f}'.format(mutual_info_score(y, y_pred)))
print('Rand Index: {:0.4f}'.format(adjusted_rand_score(y, y_pred)))
print('Purity Score: {:0.4f}'.format(purity_score(y, y_pred)))
```




```
Davies Bouldin Score: 0.6441
Dunn Index: 0.0409
Mutual Information Score: 0.6514
Rand Index: 0.5389
Purity Score: 0.7200
```

Problem 5: Implementing Image Segmentation via Unsupervised Clustering on Kaggle Competition (20pts)

After designing your very own classes implementing the popular K-means and GMM algorithms for clustering, we are now going to test them out on image data for segmentation of different structures therein. The first part of this question is designed to guide you in a step-by-step process to convert a simple, gray-scale image in a form that can be processed by the clustering classes you designed above before converting the result back in a spatiotemporal form for visualization of the segmented structures.

(a)

Run the code cell below to load an image from `sklearn`'s `digits` dataset representing images of numbers $0 - 9$$0 - 9$. Complete the function template in the cell to reshape the features in the form of an $8\times 8$$8\times 8$ image that can be displayed using `matplotlib`'s `imshow` function. (*Hint: You may find it helpful to use* `np.reshape` *function.*)

(b)

Execute the cell below that takes a digit example as input to each of the clustering classes (K-means and GMM) you designed above to output a segmentation result. Remember that in this case, each pixel in the image is going to be a 'training example' from the perspective of the clustering algorithm, with the 'feature' being the gray scale value itself. Fill in the function template reshaping the digit example into the form needed for your custering classes.

(c)

For a simple problem like above with only gray-scale images, you learnt to process images in a form they could be used to train clustering algorithms (with each individual pixel being a 'training example'). For an RGB image, each training example would have at least 3 features (the Red, Green, and Blue values for the pixel). In addition, one could add the spatial positions of the pixel as another set of features. We are now going test what you have learnt by means of a Kaggle competition wherein you are asked to segment two RGB images. Your results will be submitted to a Kaggle leaderboard to be graded accordingly. Try different feature combinations, image processing techniques to get the best looking results

We provide you with two images: a simple image consisting of geometric shapes and another one containing more complicated objects, both in RGB. The images can be downloaded by running the following snippet of code:

```
!gdown --id 1ZAvUJktJ0aojeXWJnuxuYqHlLslCZ9T-
!gdown --id 1Qh2HppgVSAniVqWxRsbJ7cZFhUpRdUpI
```

You can then load the two images as shown below:

```
import imageio

im1 = imageio.imread('img1.png')
im2 = imageio.imread('img2.png')
```

Since this is an open-ended question, you should try to design your own features to achieve better classification results. By submitting your result to Kaggle, you will see your multi-class classification accuracy and ranking on the leaderborad. 10 pts will depend on your ranking (top 10% get 10pts, top 11%-20% get 9pts, etc.), and another 5 pts depend on your method explanation and clear, well-labeled plots and images.

[Note:]

- After having your ideal result, please save your result in `result_img1` and `result_img2` in the cell below.
- We have defined the label for each class in each image. Make sure you use the same settings as us and feel free to use the provided swapping label code to swap labels if needed.
    - for `img1.png`, use k=3, background labels as 0, rectangle labels as 1 and triangle labels as 2
    - for `img2.png`, use k=2, background labels as 0, dog labels as 1
- After your `result_img1` and `result_img2` are ready, run the cell below to create a `submission.csv`. Please download it from this notebook and submit it in our Kaggle competition.
- Please remember to note your Kaggle competition nickname in this notebook. We will use your ranking to grade.

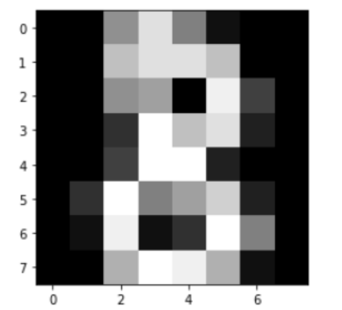
- You have 10 submission quota each day to submit your result and get your multi-class classification accuracy and ranking.
- We calculate the multi-class classification accuracy with ground-turth hand-crafted labels in both images by

$$\text{Accuracy} = \frac{\text{# of correctly labeled pixels}}{\text{total # of pixels}}$$$$\text{Accuracy} = \frac{\text{# of correctly labeled pixels}}{\text{total # of pixels}}$$

Problem 5 (a) Solution

```
## for problem 5 (a)

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits

# function to reshape
def image_reshape(x):
  """Function reshapes a training example from the Digits dataset into an image

  Parameters
  ----------
  x: ndarray of shape (1, num_of_features)
    flattened image example from the digits dataset

  Returns
  -------
  img: ndarray of shape (8,8)
    ndarray containing reshaped image for visualization
  """

  return np.reshape(x, (8,8)) ##TODO

#----------------Don't change anything below-----------------------#

# load data and extract a digit example
X, _ = load_digits(return_X_y=True)
digit = X[8].reshape(1, -1)

# reshape image
```

```
reshaped_img = image_reshape(digit)

# visualize
plt.imshow(reshaped_img, cmap='gray')
plt.show()
```

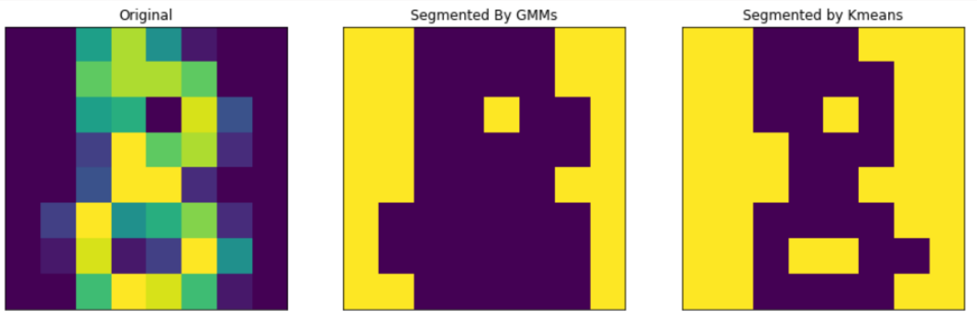


Problem 5 (b) Solution

```
## for problem 5 (b)

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits

def feature_reshape(digit):
  """function reshapes a digit training example into a form acceptable for use with
  clustering classes

  Parameters:
  -----------
  digit: ndarray of shape (1, num_of_features)

  Returns:
  --------
```

```
  reshaped_digit: ndarray of shape (num_of_features, 1)
  """

  return digit.reshape([-1,1]) ##TODO


#-----------------Don't change anything below-----------------------#


# load data and extract a digit example
X, _ = load_digits(return_X_y=True)
digit = X[8].reshape(1, -1)

# number of mixtures/clusters
k = 2

reshaped_digit = feature_reshape(digit)
# print(np.shape(reshaped_digit))

# cluster with model of choice
model_1 = MyGMMs(reshaped_digit, k=k, num_iter=20)
model_2 = MyKMeans(reshaped_digit, k=k, num_iter=20)
y_pred_1 = model_1.fit_predict()
y_pred_2 = model_2.fit_predict()

# visualize results
segmented_im1_1 = y_pred_1.reshape(8, 8)
segmented_im1_2 = y_pred_2.reshape(8, 8)

fig, (ax1, ax2, ax3) = plt.subplots(1,3, figsize=(15,9))
ax1.imshow(image_reshape(digit))
ax1.set_title('Original')
ax1.set_xticks([])
ax1.set_yticks([])
ax2.imshow(segmented_im1_1)
ax2.set_title('Segmented By GMMs')
ax2.set_xticks([])
ax2.set_yticks([])
ax3.imshow(segmented_im1_2)
ax3.set_title('Segmented by Kmeans')
ax3.set_xticks([])
```

```
ax3.set_yticks([])
plt.show()
```



Problem 5 (c) Solution

```
!gdown --id 1ZAvUJktJ0aojeXWJnuxuYqHlLslCZ9T-
!gdown --id 1Qh2HppgVSAniVqWxRsbJ7cZFhUpRdUpI
```

```
Downloading...
From: https://drive.google.com/uc?id=1ZAvUJktJ0aojeXWJnuxuYqHlLslCZ9T-
To: /content/img1.png
100% 8.57k/8.57k [00:00<00:00, 7.26MB/s]
Downloading...
From: https://drive.google.com/uc?id=1Qh2HppgVSAniVqWxRsbJ7cZFhUpRdUpI
To: /content/img2.png
100% 34.1k/34.1k [00:00<00:00, 39.0MB/s]
```

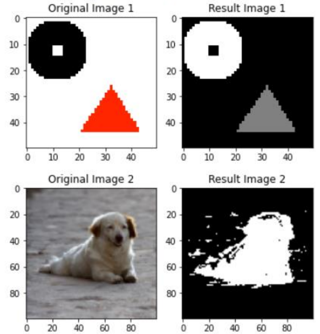
```
import imageio

im1 = imageio.imread('img1.png') # use k=3, background labels as 0, rectangle labels as 1 and triangle labels as 2
im2 = imageio.imread('img2.png') # use k=2, background labels as 0, dog labels as 1

# for Problem 2 (e)
```

```
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture

X1 = np.concatenate((np.reshape(im1[:,:,0], [-1,1]),
                    np.reshape(im1[:,:,1], [-1,1]),
                    np.reshape(im1[:,:,2], [-1,1])), axis=1)
X2 = np.concatenate((np.reshape(im2[:,:,0], [-1,1]),
                    np.reshape(im2[:,:,1], [-1,1]),
                    np.reshape(im2[:,:,2], [-1,1])), axis=1)
# print(np.shape(X1))
# print(np.shape(X2))


model_GMM_im1 = GaussianMixture(3,random_state=0) ##TODO
model_GMM_im2 = GaussianMixture(2,random_state=0) ##TODO

y_pred_GMM_im1 = model_GMM_im1.fit_predict(X1)
y_pred_im2 = model_GMM_im2.fit_predict(X2)

model_KM_im2 = KMeans(n_clusters=2, random_state=0) ##TODO

y_pred_im2 = y_pred_im2 + model_KM_im2.fit_predict(X2)

for i in range(2):
  model_GMM_im2 = GaussianMixture(2,random_state=0) ##TODO
  y_pred_im2 = y_pred_im2 + model_GMM_im2.fit_predict(X2)


y_pred_im2 = y_pred_im2/4


y_pred_im2[y_pred_im2>=0.5]=1
y_pred_im2[y_pred_im2<0.5]=0

# print(np.shape(np.reshape(y_pred_kmeans_im1,[-1,1])))

import matplotlib.pyplot as plt
new_im1 = np.reshape(np.reshape(y_pred_GMM_im1,[-1,1]), [50,50])
new_im2 = np.reshape(np.reshape(y_pred_im2,[-1,1]), [100,100])
# figure()

new_im2[0:15,:] = 0
```

```
new_im2[83:100,:] = 0
new_im2[:,0:5] = 0
new_im2[:,95:100] = 0
new_im2[30:40,70:80] = 1

result_im2 = np.zeros((100,100))
for i in np.arange(5,95):
  for j in np.arange(5,95):
    result_im2[i,j] = (new_im2[i+1,j]+ new_im2[i-1,j]+
                       new_im2[i,j+1]+ new_im2[i,j-1]+
                       new_im2[i,j]+ new_im2[i+1,j+1]+
                       new_im2[i-1,j-1]+new_im2[i+1,j-1]+
                       new_im2[i-1,j+1])/9
    if (new_im2[i,j] >= 0.45):
      result_im2[i,j] = 1
    else:
      result_im2[i,j] = 0

# plt.imshow(new_im1, 'gray')
plt.figure()
plt.subplot(121)
plt.title('Original Image 1')
plt.imshow(im1)
plt.subplot(122)
plt.title('Result Image 1')
plt.imshow(new_im1, 'gray')
plt.figure()
plt.subplot(121)
plt.title('Original Image 2')
plt.imshow(im2)
plt.subplot(122)
plt.title('Result Image 2')
plt.imshow(result_im2, 'gray')
```

-

```
<matplotlib.image.AxesImage at 0x7faab6e00b90>
```




```
## make sure img1 has shape (50, 50) and img2 has shape (100, 100)
result_img1 = new_im1  ##TODO
result_img2 = result_im2  ##TODO


## -----------------swap labels if needed-----------------------------##
## Here is the template code for you, if you need to swap label 0 and 1

result_img1[result_img1==1] = -1
result_img1[result_img1==2] = 1
result_img1[result_img1==-1] = 2

#-----------------Don't change anything below-----------------------#
## After running this cell, you should be able to download sumission.csv file on the lefe side bar.

cat_data = np.concatenate((result_img1.reshape(-1, 1), result_img2.reshape(-1, 1)), axis=0)
np.savetxt('/content/submission.csv', np.concatenate((np.arange(12500).reshape(-1, 1), cat_data), axis=1), delimiter=',', header="Id,C
```

* My Kaggle Nickname is: tyu304