Spring 2022 CS4641/CS7641 A Homework 2

Instructor: Dr. Mahdi Roozbahani

Deadline: Thursday, March 3rd, 11:59 pm AOE

- No unapproved extension of the deadline is allowed. Late submission will lead to 0 credit.
- Discussion is encouraged on Ed as part of the Q/A. However, all assignments should be done individually.
- Plagiarism is a **serious offense**. You are responsible for completing your own work. You are not allowed to copy and paste, or paraphrase, or submit materials created or published by others, as if you created the materials. All materials submitted must be your own.
- All incidents of suspected dishonesty, plagiarism, or violations of the Georgia Tech Honor Code will be subject to the institute's Academic Integrity procedures. If we observe any (even small) similarities/plagiarisms detected by Gradescope or our TAs, **WE WILL DIRECTLY REPORT ALL CASES TO OSI**, which may, unfortunately, lead to a very harsh outcome. **Consequences can be severe, e.g., academic probation or dismissal, grade penalties, a 0 grade for assignments concerned, and prohibition from withdrawing from the class.**

Instructions for the assignment

- This assignment consists of both programming and theory questions.
- Unless a theory question explicitly states that no work is required to be shown, you must provide an explanation, justification, or calculation for your answer.
- To switch between cell for code and for markdown, see the menu -> Cell -> Cell Type
- You can directly type Latex equations into markdown cells.
- If a question requires a picture, you could use this syntax `<img src="" style="width: 300px;"/>` to include them within your ipython notebook.
- Your write up must be submitted in PDF form. You may use either Latex, markdown, or any word processing software. We will **NOT** accept handwritten work. Make sure that your work is formatted correctly, for example submit $\sum_{i=0} x_i$ instead of \text{sum_{i=0} x_i}
- When submitting the non-programming part of your assignment, you must correctly map pages of your PDF to each question/subquestion to reflect where they appear. **Improperly mapped questions may not be graded correctly and/or will result in point deductions for the error.**
- All assignments should be done individually, and each student must write up and submit their own answers.
- **Graduate Students**: You are required to complete any sections marked as Bonus for Undergrads

Using the autograder

- You will find three assignments on Gradescope that correspond to HW2: "Assignment 2 Programming", "Assignment 2 - Non-programming" and "Assignment 2 Programming - Bonus for all".

- You will submit your code for the autograder in the Assignment 2 Programming sections. Please refer to the Deliverables and Point Distribution section for what parts are considered required, bonus for undergrads, and bonus for all.
- We provided you different .py files and we added libraries in those files please DO NOT remove those lines and add your code after those lines. Note that these are the only allowed libraries that you can use for the homework.
- You are allowed to make as many submissions until the deadline as you like. Additionally, note that the autograder tests each function separately, therefore it can serve as a useful tool to help you debug your code if you are not sure of what part of your implementation might have an issue.
- **For the "Assignment 2 - Non-programming" part, you will download your Jupyter Notebook as html and submit it as a PDF on Gradescope. To download the notebook as html, click on "File" on the top left corner of this page and select "Download as > html". Then, open the html file and print to PDF.** Please refer to the Deliverables and Point Distribution section for an outline of the non-programming questions.
- **When submitting to Gradescope, please make sure to mark the page(s) corresponding to each problem/sub-problem.**

Deliverables and Points Distribution

Q1: KMeans Clustering & DBScan [55 pts + 10 pts Bonus for Undergrad]

Deliverables: kmeans.py, dbscan.py, and Written Report

- **pairwise_dist** [5 pts] - *programming*
- **KMeans Implementation** [35pts] - *programming*
  - _init_centers [5pts]
  - _update_assignment [10pts]
  - _update_centers [10pts]
  - _get_loss function [5pts]
  - _test_centers (Additional Autograder Tests - no need to implement) [5pts]
- **Elbow Method** [15 pts]
  - find_num_optimal_clusters [10pts] - *programming*

    - written questions [5pts] - *non-programming*
- **DBScan** [10 pts] - *programming* **BONUS FOR UNDERGRAD**
    - regionQuery [2pts]
    - expandClusters [4pts]
    - fit [4pts]

Q2: EM Algorithm [20pts]

Deliverables: Written Report

- **2.1 Performing EM Update** [20 pts] - *non-programming*
    - 2.1.1 [4pts] - *non-programming*
    - 2.1.2 [4pts] - *non-programming*
    - 2.1.3 [8pts] - *non-programming*
    - 2.1.4 [4pts] - *non-programming*

Q3: GMM implementation [55pts; 5pts Bonus for All]

Deliverables: gmm.py and Written Report

- 3.1 Helper Functions [15pts] - *programming & non-programming*
    - 3.1.1. softmax [5pts]
    - 3.1.2. logsumexp [3pts + 2pts] - *programming & non-programming*
    - 3.1.3. normalPDF [5pts] - *for CS4641 students only*
    - 3.1.3. multinormalPDF [5pts] - *for CS7641 students only*
- 3.2 GMM Implementation [30pts] - *programming*
    - 3.2.1. init_components [5pts]
    - 3.2.2. _ll_joint [10pts]
    - 3.2.3. Setup iterative steps for EM algorithm [15pts]
- 3.3 Image Compression and Pixel clustering [10pts] - *non-programming*
- 3.4 Compare Full Convariance Matrix with Diagonal Covariance Matrix [5pts Bonus for All]

Q4: Cleaning Super Duper Messy data with semi-supervised learning [30pts Bonus for All]

Deliverables: semisupervised.py and Written Report

- 4.1: KNN [10pts] - *programming*
    - 4.1.a. complete, *incomplete*, unlabeled_ [3pts]
    - 4.1.b. CleanData __call__ [7pts]
- 4.2: Getting acquainted with semi-supervised learning approaches [5pts] - *non-programming*
- 4.3: Implementing the EM algorithm [10pts] - *programming*
    - _init_components [5pts]
    - SemiSupervised __call__ [5pts]
- 4.4: Demonstrating the performance of the algorithm [5pts] - *programming*
    - accuracy_semi_supervised [2.5pts]
    - accuracy_GNB [2.5pts]

0 Set up

This notebook is tested under python 3.6.8 (https://www.python.org/downloads/release/python-368/), and the corresponding packages can be downloaded from miniconda (https://docs.conda.io/en/latest/miniconda.html). You may also want to get yourself familiar with several packages:

- jupyter notebook (https://jupyter-notebook.readthedocs.io/en/stable/)
- numpy (https://docs.scipy.org/doc/numpy-1.15.1/user/quickstart.html)
- matplotlib (https://matplotlib.org/users/pyplot_tutorial.html)

You can create a python conda environment with the necessary packages using the instructions in the `environment/environment_setup.md` file.

Please implement the functions that have "raise NotImplementedError", and after you finish the coding, please delete or comment "raise NotImplementedError".

In [1]:

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from __future__ import absolute_import
from __future__ import print_function
from __future__ import division

%matplotlib inline

import sys
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from tqdm import tqdm

print('Version information')

print('python: {}'.format(sys.version))
print('matplotlib: {}'.format(matplotlib.__version__))
print('numpy: {}'.format(np.__version__))

# Load image
import imageio

%load_ext autoreload
%autoreload 2
```

```
Version information
python: 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)]
matplotlib: 3.4.3
numpy: 1.20.3
```

1. Rob's Balloon Commission - KMeans Clustering [55 pts + 10 pts Bonus for Undergrad]

Rob Boss, after being stuck in his dorm for multiple nights finishing Homework 1 for his Machine Learning class, decides to get some well deserved fresh air. He plans on visiting the colorful birds festival downtown this weekend with a few friends. Once there, he takes many pictures and ultimately decides that his favorite is the picture attached below (please compliment Rob on his photography and machine learning skills the next time you see him) and wants to hang it up in his dorm. He decides to commission his artist friend to convert his picture into a painting but finds out his friend charges him for each color used when painting. Being the cash-strapped college student he is, help Rob build a KMeans clustering algorithm that can help him decide how many colors he wants to commission for his painting so that he can see the feather colors without spending too much money.

KMeans is trying to solve the following optimization problem:

$$\arg\min_{S} \sum_{i=1}^{K} \sum_{x_j \in S_i} \|x_j - \mu_i\|^2$$

where one needs to partition the N observations into K clusters: $S = \{S_1, S_2, \ldots, S_K\}$ and each cluster has μ_i as its center.

1.1 pairwise distance [5pts]

In this section, you are asked to implement pairwise_dist function.

Given $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ and $\boldsymbol{Y} \in \mathbb{R}^{M \times D}$, obtain the pairwise distance matrix $\boldsymbol{dist} \in \mathbb{R}^{N \times M}$ using the euclidean distance metric, where $\boldsymbol{dist}_{i,j} = \|\boldsymbol{X}_i - \boldsymbol{Y}_j\|_2$.

DO NOT USE FOR LOOPs in your implementation, **using for-loops or while-loops will result in 0 credit for this portion.** Use array broadcasting instead (https://numpy.org/doc/stable/user/basics.broadcasting.html).

Test Case:

In [8]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################

# Set random seed so output is all same
np.random.seed(1)

from kmeans import pairwise_dist

x = np.random.randn(2, 2)
y = np.random.randn(3, 2)

print("*** Expected Answer ***")
print("""==x==
[[ 1.62434536 -0.61175641]
 [-0.52817175 -1.07296862]]
==y==
[[ 0.86540763 -2.3015387 ]
 [ 1.74481176 -0.7612069 ]
 [ 0.3190391  -0.24937038]]
==dist==
[[1.85239052 0.19195729 1.35467638]
 [1.85780729 2.29426447 1.18155842]]""")


print("\n*** My Answer ***")
print("==x==")
print(x)
print("==y==")
print(y)
print("==dist==")
print(pairwise_dist(x, y))
```

```
*** Expected Answer ***
==x==
[[ 1.62434536 -0.61175641]
 [-0.52817175 -1.07296862]]
==y==
[[ 0.86540763 -2.3015387 ]
 [ 1.74481176 -0.7612069 ]
 [ 0.3190391  -0.24937038]]
==dist==
[[1.85239052 0.19195729 1.35467638]
 [1.85780729 2.29426447 1.18155842]]

*** My Answer ***
==x==
[[ 1.62434536 -0.61175641]
 [-0.52817175 -1.07296862]]
==y==
[[ 0.86540763 -2.3015387 ]
 [ 1.74481176 -0.7612069 ]
 [ 0.3190391  -0.24937038]]
==dist==
[[1.85239052 0.19195729 1.35467638]
 [1.85780729 2.29426447 1.18155842]]
```

1.2 KMeans Implementation [30pts]

In this section, you are asked to implement _init_centers [5pts], _update_assignment [10pts], _update_centers [10pts] and _get_loss function [5pts].

For the function signature, please see the corresponding doc strings.

HINT: In _update_centers(), If you need to reduce the number of clusters when there are 0 points for a center, then do so.

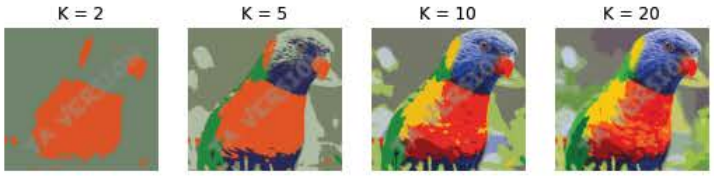
In [9]:

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from kmeans import KMeans

def image_to_matrix(image_file, grays=False):
    """
    Convert .png image to matrix
    of values.
    params:
    image_file = str
    grays = Boolean
    returns:
    img = (color) np.ndarray[np.ndarray[np.ndarray[float]]]
    or (grayscale) np.ndarray[np.ndarray[float]]
    """
    img = plt.imread(image_file)
    # in case of transparency values
    if len(img.shape) == 3 and img.shape[2] > 3:
        height, width, depth = img.shape
        new_img = np.zeros([height, width, 3])
        for r in range(height):
            for c in range(width):
                new_img[r, c, :] = img[r, c, 0:3]
        img = np.copy(new_img)
    if grays and len(img.shape) == 3:
        height, width = img.shape[0:2]
        new_img = np.zeros([height, width])
        for r in range(height):
            for c in range(width):
                new_img[r, c] = img[r, c, 0]
        img = new_img
    return img

def update_image_values(k):
    cluster_idx, centers, loss = KMeans()(image_values, k)
    updated_image_values = np.copy(image_values)

    # assign each pixel to cluster mean
    for i in range(0,k):
        indices_current_cluster = np.where(cluster_idx == i)[0]
        updated_image_values[indices_current_cluster] = centers[i]

    updated_image_values = updated_image_values.reshape(r,c,ch)
    return updated_image_values

def plot_image(img_list, title_list, figsize=(9, 12)):
    fig, axes = plt.subplots(1, len(img_list), figsize=figsize)
    for i, ax in enumerate(axes):
        ax.imshow(img_list[i])
        ax.set_title(title_list[i])
        ax.axis('off')
```

In [10]:

```
################################
### DO NOT CHANGE THIS CELL ###
################################

#Note that because of a different file structure, students' paths will be different

image_values = image_to_matrix('./data/images/bird.png')

r = image_values.shape[0]
c = image_values.shape[1]
ch = image_values.shape[2]
# flatten the image_values
image_values = image_values.reshape(r*c,ch)

print('Loading...')

image_2 = update_image_values(2).reshape(r, c, ch)
image_5 = update_image_values(5).reshape(r, c, ch)
image_10 = update_image_values(10).reshape(r, c, ch)
image_20 = update_image_values(20).reshape(r, c, ch)

plot_image([image_2, image_5, image_10, image_20], ['K = 2', 'K = 5', 'K = 10', 'K = 20'])
```

```
Loading...
```

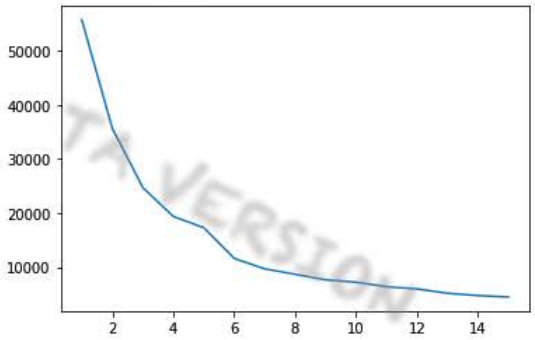


1.3 Elbow Method [10 pts Programming + 5 pts Written Questions]

One of the biggest drawbacks of KMeans is that we need to know the number of clusters beforehand. Let's see if we can help Rob's paint optimization problem by implementing the elbow method to find the optimal number of clusters in the function find_optimal_num_clusters below.

In [7]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################

from kmeans import find_optimal_num_clusters

find_optimal_num_clusters(image_values)
```

Out[7]:
```
[55710.66015625,
 35642.8603515625,
 24710.19384765625,
 19367.06298828125,
 17287.696166992188,
 11604.470275878906,
 9666.754577636719,
 8666.904907226562,
 7630.714752197266,
 7206.045715332031,
 6354.473876953125,
 5955.548156738281,
 5156.260559082031,
 4732.635360717773,
 4453.780471801758]
```



Written Questions (5 pts):

1) Approximately what value does the elbow method give?

2) Rob is not quite sure if he should use the loss graph to select a K that minimizes loss as close to 0 as possible or to use the elbow method. Which is a better option and why? Does your elbow method elbow method answer reflect this in the resulting images produced by kmeans?

Answers:

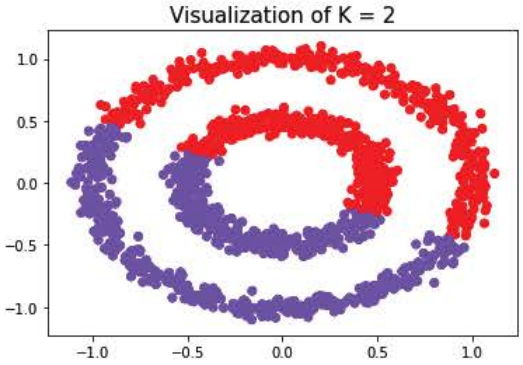
1.
2.

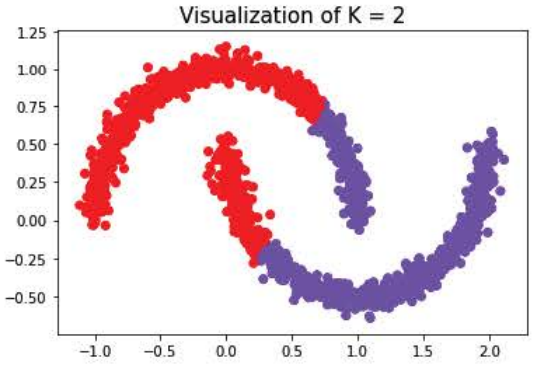
Limitation of K-Means

One of the limitations of K-Means Clustering is that it dependes largely on the shape of the dataset. A common example of this is trying to cluster one circle within another (concentric circles). A K-means classifier will fail to do this and will end up effectively drawing a line which crosses the circles. You can visualize this limitation in the cell below.

In [6]:

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

# visualize limitation of kmeans
from sklearn.datasets import (make_circles, make_moons)

X1, y1 = make_circles(factor=0.5, noise=0.05, n_samples=1500)
X2, y2 = make_moons(noise=0.05, n_samples=1500)

def visualise(X, C, K=None):# Visualization of clustering. You don't need to change this function
    fig, ax = plt.subplots()
    ax.scatter(X[:, 0], X[:, 1], c=C,cmap='rainbow')
    if K:
        plt.title('Visualization of K = '+str(K), fontsize=15)
    plt.show()
    pass

cluster_idx1, centers1, loss1 = KMeans()(X1, 2)
visualise(X1, cluster_idx1, 2)

cluster_idx2, centers2, loss2 = KMeans()(X2, 2)
visualise(X2, cluster_idx2, 2)
```





1.5 Autograder test to find centers for data points [5 pts]

To obtain these 5 points, you need to be pass the tests set up in the autograder. These will test the centers created by your implementation. Be sure to upload the correct files to obtain these points.

1.6 DBSCAN [10 pts BONUS FOR UNDERGRAD]

Let us try to solve these limitations using another clustering algorithm: DBSCAN. As mentioned in lecture, DBSCAN tries to find dense regions in the data spce, separated by regions of lower density. DBSCAN is parameterized by two parameters (eps and minPts):

- ϵ: Maximum radius of neighborhood
- $MinPts$: Minimum number of points in Eps-neighborhood of a point to be considered 'dense'.

The algorithm fits on a dataset as follows:

For each unvisited point in dataset:

1. Set point as visited
2. Extract the Eps-neighborhood $Neighbors$ of this point using regionQuery() (all points which are within ϵ distance of this neighborhood)
3. If the size of this neighbohood is greater than $minPts$, then we run expandCluster(), which does the following for cluster C:

A. Set up a queue ***expand*** with P in it
B. Add P to cluster C
C. While the ***expand*** queue is not empty:
    a. Pop a P' from the queue and mark it as visited
    b. Set point P' as visited
    c. For each point P'' in P''s neighborhood ***Neighbors***:
        i. Extract the Eps-neighborhood ***Neighbors'*** of P'' using regionQuery()
        ii. If the size of ***Neighbors'*** is greater than or equal to minPts, we add any unvisited points in ***Neighbors'*** to our current ***expand***
        iii. Add P'' to cluster C if P'' is unvisited.
D. Move onto next cluster $C+1$

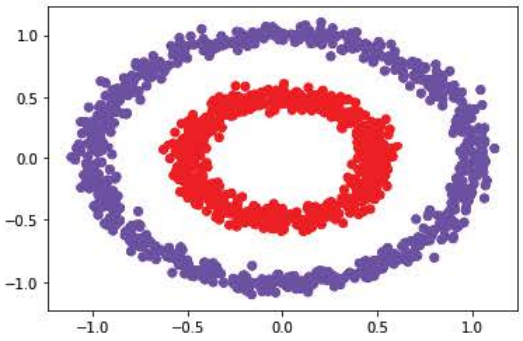
Using the above description, complete fit(), expandCluster(), and regionQuery() in dbscan.py.

HINT: You might find it easier to implement expandCluster() before attempting to implement fit().

Then, test your code by running the cell below. You should be able to get a perfect clustering for the two circles dataset, which you can observe quantitatively by checking whether the clusters returned by cluster_idx and the ground truth clusters are the same and qualitatively by visualizing the clusters.

In [7]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################
BEST_EPS = 0.11
BEST_POINTS = 3
from dbscan import DBSCAN
dbscan = DBSCAN(BEST_EPS, BEST_POINTS, X1)
cluster_idx = dbscan.fit()
## Note that one of the two cells should print True for a correct implementation
print(np.array_equal(y1, cluster_idx)) #Checks if y1 == cluster_idx
print(np.array_equal(y1, 1-cluster_idx)) ## Checks if y1 is the exact opposite of cluster_idx (1s instead of
visualise(X1, cluster_idx)
```

```
True
False
```



2. EM algorithm [20 pts]

2.1 Performing EM Update [20 pts]

SOLUTIONS CANNOT BE HANDWRITTEN

A univariate Gaussian Mixture Model (GMM) has two components, both of which have their own mean and standard deviation. The model is defined by the following parameters:

$$z \sim Bernoulli(\theta) = \begin{cases} \theta & \text{if } z = 0 \\ 1-\theta & \text{if } z = 1 \end{cases}$$
$$p(x|z=0) \sim \mathcal{N}(\mu, \sigma^2)$$
$$p(x|z=1) \sim \mathcal{N}(2\mu, 3\sigma^2)$$

For a dataset of N datapoints, find the following:

2.1.1. Write the marginal probability of x, i.e. $p(x)$ [4pts]
-- Express your answers in terms of $\mathcal{N}(a,b)$ where a represents the mean and b represents the variance of a normal distribution, and θ
-- HINT: Start with the Sum Rule

2.1.2. E-Step: Compute the posterior probability, i.e, $p(z_i = k|x_i)$, where k = {0,1} [4pts]
-- Express your answers in terms of $\mathcal{N}(a,b)$ where a represents the mean and b represents the variance of a normal distribution, and θ

2.1.3. M-Step: Compute the updated value of σ^2 (You can keep μ fixed for this) [8pts]
-- *Express your answers in terms of* τ, x *(when you expand the* $\mathcal{N}(a, b)$ *terms), and* μ
-- *HINT: Start from this equation and substitute* θ *with* σ^2: $\theta_{new} = argmax_\theta \sum_Z p(Z|X, \theta_{old}) \ln p(X, Z|\theta)$

2.1.4. Explain why $\sum_k = \frac{1}{\sum_{n=1}^N \tau(z_{nk})} \sum_{n=1}^N \tau(z_{nk})(x_n - \mu_k)(x_n - \mu_k)^T$ is not the correct equation to use to solve 2.1.3 [4pts]

Answers:

2.1.1.

2.1.2.

2.1.3.

2.1.4.

3. GMM implementation [43pts Programming + 12pts Written Questions + 5pts bonus for all pts]

Please make sure to read the problem setup in detail. Many questions for this section may have already been answered in the description and hints and docstrings.

A Gaussian Mixture Model(GMM) is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian Distribution. In a nutshell, GMM is a soft clustering algorithm in a sense that each data point is assigned to a cluster with a probability. In order to do that, we need to convert our clustering problem into an inference problem.

Given N samples $X = [x_1, x_2, \dots, x_N]^T$, where $x_i \in \mathbb{R}^D$. Let π be a K-dimensional probability distribution and $(\mu_k; \Sigma_k)$ be the mean and covariance matrix of the k^{th} Gaussian distribution in $\mathbb{R}^d$.

The GMM object implements EM algorithms for fitting the model and MLE for optimizing its parameters. It also has some particular hypothesis on how the data was generated:

- Each data point x_i is assigned to a cluster k with probability of π_k where $\sum_{k=1}^K \pi_k = 1$
- Each data point x_i is generated from Multivariate Normal Distribution $\mathcal{N}(\mu_k, \Sigma_k)$ where $\mu_k \in \mathbb{R}^D$ and $\Sigma_k \in \mathbb{R}^{D \times D}$

Our goal is to find a K-dimension Gaussian distributions to model our data X. This can be done by learning the parameters π, μ and Σ through likelihood function. Detailed derivation can be found in our slide of GMM. The log-likelihood function now becomes:

$$\ln p(x_1, \dots, x_N|\pi, \mu, \Sigma) = \sum_{i=1}^N \ln \Big(\sum_{k=1}^K \pi(k) \mathcal{N}(x_i|\mu_k, \Sigma_k) \Big)$$

From the lecture we know that MLEs for GMM all depend on each other and the responsibility τ. Thus, we need to use an iterative algorithm (the EM algorithm) to find the estimate of parameters that maximize our likelihood function. **All detailed derivations can be found in the lecture slide of GMM.**

- **E-step:** Evaluate the responsibilities

In this step, we need to calculate the responsibility τ, which is the conditional probability that a data point belongs to a specific cluster k if we are given the datapoint, i.e. $P(z_k|x)$. The formula for τ is given below:

$$\tau(z_k) = \frac{\pi_k \mathcal{N}(x|\mu_k, \Sigma_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(x|\mu_j, \Sigma_j)}, \quad \text{for } k = 1, \dots, K$$

Note that each data point should have one probability for each component/cluster. For this homework, you will work with $\tau(z_k)$ which has a size of $N \times K$ and you should have all the responsibility values in one matrix. **We use gamma as τ in this homework**.

- **M-step:** Re-estimate Paramaters

After we obtained the responsibility, we can find the update of parameters, which are given below:

$$\mu_k^{new} = \frac{\sum_{n=1}^N \tau(z_k) x_n}{N_k}$$

$$\Sigma_k^{new} = \frac{1}{N_k} \sum_{n=1}^N \tau(z_k)^T (x_n - \mu_k^{new})^T (x_n - \mu_k^{new})$$

$$\pi_k^{new} = \frac{N_k}{N}$$

where $N_k = \sum_{n=1}^N \tau(z_k)$. Note that the updated value for μ_k is used when updating Σ_k. The multiplication of $\tau(z_k)^T (x_n - \mu_k^{new})^T$ is element-wise so it will preserve the dimensions of $(x_n - \mu_k^{new})^T$.

- We repeat E and M steps until the incremental improvement to the likelihood function is small.

Special Notes

- For undergraduate student: you may assume that the covariance matrix Σ is diagonal matrix, which means the features are independent. (i.e. the red intensity of a pixel is independent from its blue intensity, etc). Make sure you set **Full_matrix = False** before you submit your code to Gradescope.
- For graduate student: please assume full covariance matrix. Make sure you set **Full_matrix = True** before you submit your code to Gradescope
- The class notes assume that your dataset $\boldsymbol{X}$ is $(\boldsymbol{D}, \boldsymbol{N})$ but **the homework dataset is $(\boldsymbol{N}, \boldsymbol{D})$ as mentioned on the instructions, so the formula is a little different from the lecture note in order to obtain the right dimensions of parameters.**

Hints

1. **DO NOT USE FOR LOOPS OVER N. No credit will be given for implementing the function with for or while loops that visit every datapoint.** You can always find a way to avoid looping over the observation datapoints in our homework problem. If you have to loop over D or K, that is fine.
2. You can initiate $\boldsymbol{\pi(k)}$ the same for each $\boldsymbol{k}$, i.e. $\boldsymbol{\pi(k) = \frac{1}{K}, \forall k = 1, 2, \ldots, K}$.
3. In part 3 you are asked to generate the model for pixel clustering of image. We will need to use a multivariate Gaussian because each image will hava $\boldsymbol{N}$ pixels and $\boldsymbol{D = 3}$ features which corresponds to red, green, and blue color intensities. It means that each image is a $\boldsymbol{(N \times 3)}$ dataset matrix. In the following parts, remember $\boldsymbol{D = 3}$ in this problem.
4. To avoid using for loops in your code, we recommend you take a look at the concept Array Broadcasting in Numpy (https://numpy.org/doc/stable/user/theory.broadcasting.html#array-broadcasting-in-numpy). Also, certain calculations that required different shapes of arrays can also be achieved by broadcasting.
5. Be careful of the dimensions of your parameters. Before you test anything on the autograder, please look at the instructions below on the shapes of the variables you need to output and how to format your return statement. Print the shape of an array by print(array.shape) (https://www.w3schools.com/python/numpy/numpy_array_shape.asp) could enhance the functionality of your code and help you debugging. Also notice that **a numpy array in shape $(\boldsymbol{N}, 1)$ is NOT the same as that in shape $(\boldsymbol{N},)$** so be careful and consistent on what you are using. You can see the detailed explanation here. Difference between numpy.array shape (R, 1) and (R,) (https://stackoverflow.com/questions/22053050/difference-between-numpy-array-shape-r-1-and-r)

    - The dataset $\boldsymbol{X}$: $(\boldsymbol{N}, \boldsymbol{D})$
    - $\boldsymbol{\mu}$: $(\boldsymbol{K}, \boldsymbol{D})$.
    - $\boldsymbol{\Sigma}$: $(\boldsymbol{K}, \boldsymbol{D}, \boldsymbol{D})$
    - $\boldsymbol{\tau}$: $(\boldsymbol{N}, \boldsymbol{K})$
    - $\boldsymbol{\pi}$: array of length $\boldsymbol{K}$
    - ll_joint: $(\boldsymbol{N}, \boldsymbol{K})$

3.1 Helper functions [15 pts]

To facilitate some of the operations in the GMM implementation, we would like you to implement the following three helper functions. In these functions, "logit" refers to an input array of size $(\boldsymbol{N}, \boldsymbol{D})$. Remember the goal of helper functions is to facilitate our calculation so **DO NOT USE FOR LOOP ON N**.

3.1.1. softmax [5 pts]

Given $\boldsymbol{logit} \in \mathbb{R}^{N \times D}$, calculate $\boldsymbol{prob} \in \mathbb{R}^{N \times D}$, where $\boldsymbol{prob_{i,j} = \frac{\exp(logit_{i,j})}{\sum_{d=1}^{D} \exp(logit_{i,d})}}$.

Note: it is possible that $\boldsymbol{logit_{i,j}}$ is very large, making $\exp(\cdot)$ of it to explode. To make sure it is numerically stable, you need to subtract the maximum for each row of $\boldsymbol{logits}$.

Special Notes

- Do not add back the maximum for each row.
- Add **keepdims=True** in your np.sum() function to avoid broadcast error.

3.1.2. logsumexp [3 pts Programming + 2 pts Written Questions]

Given $\boldsymbol{logit} \in \mathbb{R}^{N \times D}$, calculate $\boldsymbol{s} \in \mathbb{R}^{N}$, where $\boldsymbol{s_i = \log\left(\sum_{j=1}^{D} \exp(logit_{i,j})\right)}$. Again, pay attention to the numerical problem. You may face similar condition as in the softmax function. In this case, add the maximum for each row of $\boldsymbol{logit}$ back for your functions

Special Notes

- This function is used in the call() function, which is given, and helps calculate the loss of log-likelihood. You will not have to call it in functions that you are required to implement.

Written Questions [2 pts]:

1) Why should we add the maximum for each row of $\boldsymbol{logit}$ to **logsumexp()** function?

Hint: start with a simple example like $\boldsymbol{logit} \in \mathbb{R}^{1 \times D}$

Answer:

3.1.3. Multivariate Gaussian PDF [5 pts]

You should be able to write your own function based on the following formula, and you are **NOT allowed** to use outside resource packages other than those we provided.

(for undergrads only) normalPDF

Using the covariance matrix as a diagonal matrix with variances of the individual variables appearing on the main diagonal of the matrix and zeros everywhere else means that we assume the features are independent. In this case, the multivariate normal density function simplifies to the expression below:

$$\mathcal{N}(x:\mu,\Sigma)=\prod_{i=1}^{D}\frac{1}{\sqrt{2\pi\sigma_i^2}}\exp\left(-\frac{1}{2\sigma_i^2}(x_i-\mu_i)^2\right)$$

where σ_i^2 is the variance for the i^{th} feature, which is the diagonal element of the covariance matrix.

(for grads only) multinormalPDF

Given the dataset $X\in\mathbb{R}^{N\times D}$, the mean vector $\mu\in\mathbb{R}^{D}$ and covariance matrix $\Sigma\in\mathbb{R}^{D\times D}$ for a multivariate Gaussian distrubution, calculate the probability $p\in\mathbb{R}^{N}$ of each data. The PDF is given by

$$\mathcal{N}(X:\mu,\Sigma)=\frac{1}{(2\pi)^{D/2}}|\Sigma|^{-1/2}\exp\left(-\frac{1}{2}(X-\mu)\Sigma^{-1}(X-\mu)^T\right)$$

where $|\Sigma|$ is the determinant of the covariance matrix.

Hints

- If you encounter "LinAlgError", you can mitigate your number/array by summing a small value before taking the operation, e.g. np.linalg.inv(Σ_k + SIGMA_CONST). You can arrest and handle such error by using Try and Exception Block (https://realpython.com/python-exceptions/#the-try-and-except-block-handling-exceptions) in Python.
- In the above calculation, you must avoid computing a (N,N) matrix. Using the above equation for large N will crash your kernel and/or give you a memory error on Gradescope. Instead, you can do this same operation by calculating $(X-\mu)\Sigma^{-1}$, a (N,D) matrix, transpose it to be a (D,N) matrix and do an element-wise multiplication with $(X-\mu)^T$, which is also a (D,N) matrix. Lastly, you will need to sum over the 0 axis to get a $(1,N)$ matrix before proceeding with the rest of the calculation. This uses the fact that doing an element-wise multiplication and summing over the 0 axis is the same as taking the diagonal of the (N,N) matrix from the matrix multiplication.
- In Numpy implementation for each individual μ, you can either use a 2-D array with dimension $(1,D)$ for each Gaussian Distribution, or a 1-D array with length D. Same to other array parameters. Both ways should be acceptable but pay attention to the shape mismatch problem and be **consistent all the time** when you implement such arrays.

3.2 GMM Implementation [30 pts]

Things to do in this problem:

3.2.1. Initialize parameters in _init_components() [5 pts]

Examples of how you can initialize the parameters.

1. Set the prior probability π the same for each class.
2. Initialize μ by randomly selecting K numbers of observations as the initial mean vectors. You can use int(np.random.uniform()) (https://numpy.org/doc/stable/reference/random/generated/numpy.random.uniform.html) to get the row index number of the datapoints randomly.
3. Initialize the covariance matrix with np.eye() (https://numpy.org/devdocs/reference/generated/numpy.eye.html) for each k. For grads, you can also initialize the Σ by K diagonal matrices. It will become a full matrix after one iteration, as long as you adopt the correct computation.
4. Other ways of initialization are acceptable and welcome. The autograder will only test the shape of your π,μ, σ. Make sure you pass other evaluations in the autograder.

3.2.2. Formulate the log-likelihood function _ll_joint() [10 pts]

The log-likelihood function is given by:

$$\ell(\theta)=\sum_{i=1}^{N}\ln\left(\sum_{k=1}^{K}\pi(k)\mathcal{N}(x_i|\mu_k,\Sigma_k)\right)$$

In this part, we will generate a (N,K) matrix where each datapoint $x_i,\forall i=1,\ldots,N$ has K log-likelihood numbers. Thus, for each $i=1,\ldots,N$ and $k=1,\ldots,K$,

$$\text{log-likelihood}[i,k]=\log\pi_k+\log\mathcal{N}(x_i|\mu_k,\Sigma_k)$$

Hints:

- If you encounter "ZeroDivisionError" or "RuntimeWarning: divide by zero encountered in log", you can mitigate your number/array by summing a small value before taking the operation, e.g. np.log(π_k + 1e-32).
- You need to use the Multivariate Normal PDF function you created in the last part. Remember the PDF function is for each Gaussian Distribution (i.e. for each k) so you need to use a for loop over K.

3.2.3. Setup Iterative steps for EM Algorithm [5+10 pts]

You can find the detail instruction in the above description box.

Hints:

- For E steps, we already get the log-likelihood at _ll_joint() function. This is not the same as responsibilities (τ), but you should be able to finish this part with just a few lines of code by using _ll_joint() and softmax() defined above.
- For undergrads: Try to simplify your calculation for $\mathbf{\Sigma}$ in M steps as you assumed independent components. Make sure you are only taking the diagonal terms of your calculated covariance matrix.

Function Tests

Use these to test if your implementation of functions in GMM work as expected

In [2]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from gmm import GMM
```

In [3]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

np.random.seed(5)

data = np.random.randn(5, 4)
my_softmax = GMM(data, 3).softmax(data)
expected_softmax = np.array([[0.10782656, 0.04982049, 0.78844897, 0.05390398],
       [0.16080479, 0.70138883, 0.05805256, 0.07975382],
       [0.39637071, 0.23624673, 0.0996817 , 0.26770086],
       [0.21741805, 0.56913777, 0.05890126, 0.15454293],
       [0.27040961, 0.54778227, 0.01886611, 0.16294201]])

print("Your softmax works within the expected range: ", np.allclose(expected_softmax, my_softmax))

# test logsumexp utility
my_logsumexp = GMM(data, 3).logsumexp(data)
expected_logsumexp = np.array([[2.66845878],
       [1.93717399],
       [1.11300859],
       [1.16710435],
       [2.4592084 ]])

print("Your logsumexp works within the expected range: ", np.allclose(expected_logsumexp, my_logsumexp))

points = np.random.randn(15, 3)

mu = np.array([[-0.69166075, -0.39675353, -0.6871727 ],
       [ 0.04221375,  0.58281521, -1.10061918],
       [ 1.62434536, -0.61175641, -0.52817175]])

sigma = np.array([[[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.]],
       [[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.]],
       [[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.]]])

pi = np.ones(3)/3
```

```
Your softmax works within the expected range:  True
Your logsumexp works within the expected range:  True
```

In [4]:

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

# For undergrads

# test normalPDF
my_normalpdf = GMM(points, 3).normalPDF(points, mu[0], sigma[0])
expected_normal_pdf = np.array([0.05385474, 0.00871279, 0.03771877, 0.02631223, 0.039282  ,
       0.00442542, 0.00722764, 0.03184208, 0.00181295, 0.02640811,
       0.00471099, 0.01150689, 0.00063621, 0.01147473, 0.02380852])

print("Your normal pdf works within the expected range: ", np.allclose(expected_normal_pdf, my_normalpdf))


# test ll-joint
my_lljoint = GMM(points, 3)._ll_joint(pi, mu, sigma,False)
expected_lljoint = np.array([[ -4.02007719,  -5.44099343,  -7.3374222 ],
       [ -5.84157502,  -5.69251151,  -8.01291517],
       [ -4.37620973,  -4.91245496,  -5.25244293],
       [ -4.73633379,  -3.99850407,  -6.34444602],
       [ -4.33560118,  -4.63992545,  -5.42838328],
       [ -6.51900277,  -6.91677702,  -6.86391589],
       [ -6.02845446,  -4.65853011,  -7.70587119],
       [ -4.5455788 ,  -5.16685045,  -6.70899907],
       [ -7.4114131 ,  -6.64144377,  -7.97545579],
       [ -4.73269633,  -4.8801069 ,  -4.6231013 ],
       [ -6.45646958,  -4.88294659,  -4.53086302],
       [ -5.56342186,  -7.33396705,  -7.94493792],
       [ -8.45858816,  -6.81718207, -10.18015438],
       [ -5.56622065,  -6.52577609,  -6.12075969],
       [ -4.83632403,  -6.27755579,  -6.21813739]])

print("Your lljoint works within the expected range: ", np.allclose(my_lljoint, expected_lljoint))

# test E step
my_estep = GMM(points, 3)._E_step(pi, mu, sigma)
expected_estep = np.array([[0.78263086, 0.1889996 , 0.02836954],
       [0.43960461, 0.5102696 , 0.05012579],
       [0.49967803, 0.29228189, 0.20804008],
       [0.30379836, 0.63536137, 0.06084027],
       [0.482415  , 0.35583974, 0.16174526],
       [0.4201512 , 0.28226332, 0.29758548],
       [0.1952397 , 0.76827858, 0.03648172],
       [0.60525648, 0.32518058, 0.06956295],
       [0.26819685, 0.57922475, 0.1525784 ],
       [0.33570952, 0.28969704, 0.37459344],
       [0.0788462 , 0.38032345, 0.54083035],
       [0.79198478, 0.13482761, 0.07318761],
       [0.15769863, 0.81410708, 0.02819429],
       [0.51088176, 0.19569997, 0.29341827],
       [0.67215193, 0.15905541, 0.16879265]])

print("Your E step works within the expected range: ", np.allclose(my_estep, expected_estep))

# test M step
my_pi, my_mu, my_sigma = GMM(points, 3)._M_step(expected_estep, False)
expected_pi = np.array([0.43628293, 0.394094  , 0.16962307])
expected_mu = np.array([[-0.22485851, -0.06104529,  0.33535978],
        [ 0.00642446,  0.72356306,  0.19284601],
        [ 0.36429099,  0.01967377,  0.11272403]])
expected_sigma = np.array([[[0.20711668, 0.        , 0.        ],
        [0.        , 0.58982465, 0.        ],
        [0.        , 0.        , 0.51253382]],

       [[0.22612501, 0.        , 0.        ],
        [0.        , 0.88759053, 0.        ],
        [0.        , 0.        , 0.54600949]],

       [[0.38012959, 0.        , 0.        ],
        [0.        , 0.35395325, 0.        ],
        [0.        , 0.        , 0.846592  ]]])
print("Your M step works within the expected range: ", np.allclose(my_pi, expected_pi) and np.allclose(my_mu,
```

```
Your normal pdf works within the expected range:  True
Your lljoint works within the expected range:  True
Your E step works within the expected range:  True
Your M step works within the expected range:  True
```

In [5]:

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

# For grads

# # test mutlinormalPDF
sigma_grad = np.array([[[0.30792796,  0.07909229, -0.11016917],
        [0.07909229,  0.86422655,  0.06975468],
        [-0.11016917,  0.06975468,  0.63212106]],

       [[0.30792796,  0.07909229, -0.11016917],
        [0.07909229,  0.86422655,  0.06975468],
        [-0.11016917,  0.06975468,  0.63212106]],

       [[0.30792796,  0.07909229, -0.11016917],
        [0.07909229,  0.86422655,  0.06975468],
        [-0.11016917,  0.06975468,  0.63212106]]])
my_multinormalpdf = GMM(data, 3).multinormalPDF(points, mu[0], sigma_grad[0])
expected_multinormal_pdf = np.array([1.24124610e-01, 1.09187066e-02, 2.83199962e-02, 4.84688341e-02,
       3.96140713e-02, 3.55094160e-04, 1.15479475e-02, 5.06551884e-02,
       3.37348839e-04, 6.34219531e-03, 1.20587696e-04, 8.14168432e-03,
       5.76457373e-04, 1.82882581e-03, 1.49926366e-02])

print("Your multinormal pdf works within the expected range: ", np.allclose(expected_multinormal_pdf, my_mult


# test ll_joint
sigma_now = sigma * 0.5
my_lljoint = GMM(points, 3)._ll_joint(pi, mu, sigma_now, True)
expected_lljoint = np.array([[ -3.14500571,  -5.9868382 ,  -9.77969574],
       [ -6.78800137,  -6.48987436, -11.13068168],
       [ -3.85727081,  -4.92976126,  -5.6097372 ],
       [ -4.57751893,  -3.10185949,  -7.79374338],
       [ -3.7760537 ,  -4.38470224,  -5.9616179 ],
       [ -8.14285688,  -8.93840538,  -8.83268311],
       [ -7.16176025,  -4.42191156, -10.51659372],
       [ -4.19600894,  -5.43855224,  -8.52284947],
       [ -9.92767754,  -8.38773887, -11.05576292],
       [ -4.570244  ,  -4.86506515,  -4.35105393],
       [ -8.01779049,  -4.87074452,  -4.16657737],
       [ -6.23169506,  -9.77278544, -10.99472719],
       [-12.02202766,  -8.73921549, -15.46516011],
       [ -6.23729264,  -8.15640353,  -7.34637072],
       [ -4.77749941,  -7.65996291,  -7.54112612]])

print("Your lljoint works within the expected range: ", np.allclose(my_lljoint, expected_lljoint))


# test E step
my_estep = GMM(points, 3)._E_step(pi, mu, sigma_now)
expected_estep = np.array([[0.94372325, 0.05503671, 0.00124004],
       [0.42366876, 0.57082286, 0.00550839],
       [0.65984771, 0.22577041, 0.11438188],
       [0.18470545, 0.80788672, 0.00740783],
       [0.60368247, 0.32845499, 0.06786254],
       [0.5120336 , 0.23109797, 0.25686843],
       [0.06053431, 0.93735212, 0.00211357],
       [0.76813271, 0.22172086, 0.01014643],
       [0.16700188, 0.77894757, 0.05405055],
       [0.33447807, 0.24907403, 0.4164479 ],
       [0.01402184, 0.3262493 , 0.65972887],
       [0.96383555, 0.0279336 , 0.00823084],
       [0.0361238 , 0.96272152, 0.00115468],
       [0.67723134, 0.09937515, 0.22339351],
       [0.8936077 , 0.05003903, 0.05635326]])


print("Your E step works within the expected range: ", np.allclose(my_estep, expected_estep))

# test M step
my_pi, my_mu, my_sigma = GMM(points, 3)._M_step(expected_estep, True)
expected_pi = np.array([0.4828419 , 0.39149886, 0.12565925])
expected_mu = np.array([[-0.26263543, -0.23026888,  0.37410807],
        [ 0.02946666,  0.96190945,  0.19697914],
        [ 0.64855787, -0.0282273 , -0.12987796]])
expected_sigma = np.array([[[ 0.18480413,  0.08971316,  0.08911991],
         [ 0.08971316,  0.36907686,  0.08744919],
         [ 0.08911991,  0.08744919,  0.48008412]],

        [[ 0.17533767, -0.0757907 , -0.08561511],
```

```
         [-0.0757907 ,  0.73833814,  0.16291358],
         [-0.08561511,  0.16291358,  0.52631415]],

        [[ 0.35145756,  0.10609808, -0.51519293],
         [ 0.10609808,  0.15893304, -0.08535478],
         [-0.51519293, -0.08535478,  0.99893729]]])
print("Your M step works within the expected range: ", np.allclose(my_pi, expected_pi) and np.allclose(my_mu,
```

```
Your multinormal pdf works within the expected range:  True
Your lljoint works within the expected range:  True
Your E step works within the expected range:  True
Your M step works within the expected range:  True
```

3.3 Image Compression and pixel clustering [10pts]

Images typically need a lot of bandwidth to be transmitted over the network. In order to optimize this process, most image processors perform lossy compression of images (lossy implies some information is lost in the process of compression).

In this section, you will use your GMM algorithm implementation to do pixel clustering and compress the images. That is to say, you would develop a lossy image compression algorithm. (Hint: you can adjust the number of clusters formed and justify your answer based on visual inspection of the resulting images or on a different metric of your choosing)

Special Notes

- Try to add a small value(e.g. SIGMA_CONST and LOG_CONST) before taking the operation if the output image is solid black.
- The output images may be slightly different due to different initialization methods in GMM() function.

You do NOT need to submit your code for this question to the autograder. Instead you should include whatever images/information you find relevant in the report.

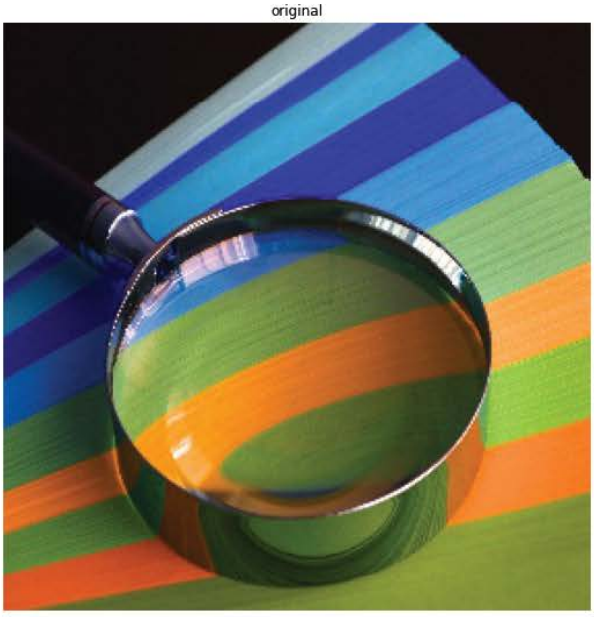
In [6]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

# helper function for performing pixel clustering.
def cluster_pixels_gmm(image, K, full_matrix = True):
    """Clusters pixels in the input image

    Args:
        image: input image of shape(H, W, 3)
        K: number of components
    Return:
        clustered_img: image of shape(H, W, 3) after pixel clustering
    """
    im_height, im_width, im_channel = image.shape
    flat_img = np.reshape(image, [-1, im_channel]).astype(np.float32)
    gamma, (pi, mu, sigma) = GMM(flat_img, K = K, max_iters = 10)(full_matrix)
    cluster_ids = np.argmax(gamma, axis=1)
    centers = mu

    gmm_img = np.reshape(centers[cluster_ids], (im_height, im_width, im_channel))

    return gmm_img

# helper function for plotting images. You don't have to modify it
def plot_images(img_list, title_list, figsize=(20, 10)):
    assert len(img_list) == len(title_list)
    fig, axes = plt.subplots(1, len(title_list), figsize=figsize)
    for i, ax in enumerate(axes):
        ax.imshow(img_list[i] / 255.0)
        ax.set_title(title_list[i])
        ax.axis('off')
```

In [8]:

```python
# the direction of two images. Both of them are from ImageNet
img1_dir ='./data/images/image1.jpg'
img2_dir ='./data/images/image2.jpg'

# example of loading image
image1 = imageio.imread('./data/images/image1.jpg')

# this is for you to implement
def perform_compression(image, min_clusters=5, max_clusters=15):
    """
    Using the helper function above to find the optimal number of clusters that can appropriately produce a s
    You can simply examinate the answer based on your visual inspection (i.e. looking at the resulting images

    Args:
        image: input image of shape(H, W, 3)
        min_clusters, max_clusters: the minimum and maximum number of clusters you should test with. Default a
        (Usually the maximum number of clusters would not exeed 15)

    Return:
        plot: comparison between original image and image pixel clustering.
        optional: any other information/metric/plot you think is necessary.
    """
    raise NotImplementedError

image1 = imageio.imread(img1_dir)
perform_compression(image1, 5, 10)

image2 = imageio.imread(img2_dir)
perform_compression(image2, 5, 10)
```

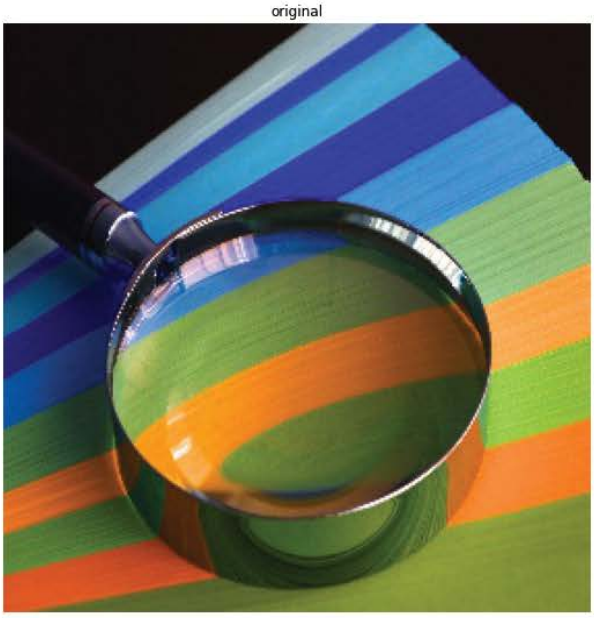
```
iter 9, loss: 1211027.8165: 100%|██████████| 10/10 [00:02<00:00,  3.38it/s]
iter 9, loss: 1146771.5648: 100%|██████████| 10/10 [00:06<00:00,  1.64it/s]
iter 9, loss: 1141961.3035: 100%|██████████| 10/10 [00:03<00:00,  3.20it/s]
iter 9, loss: 1117336.4302: 100%|██████████| 10/10 [00:06<00:00,  1.66it/s]
```

original

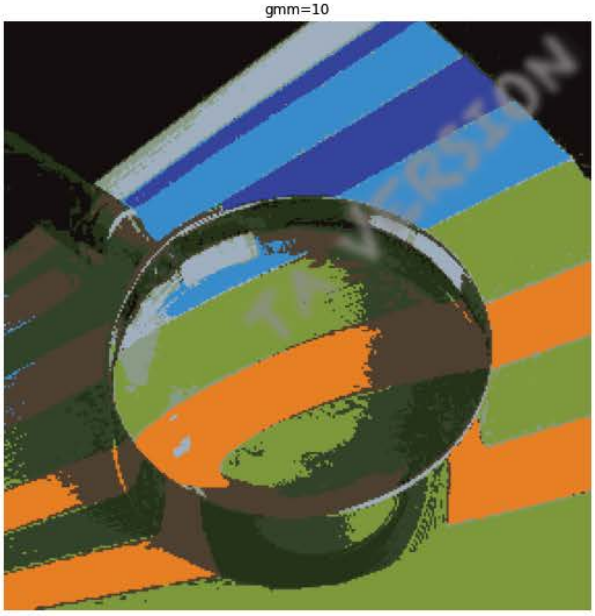


gmm=5

original

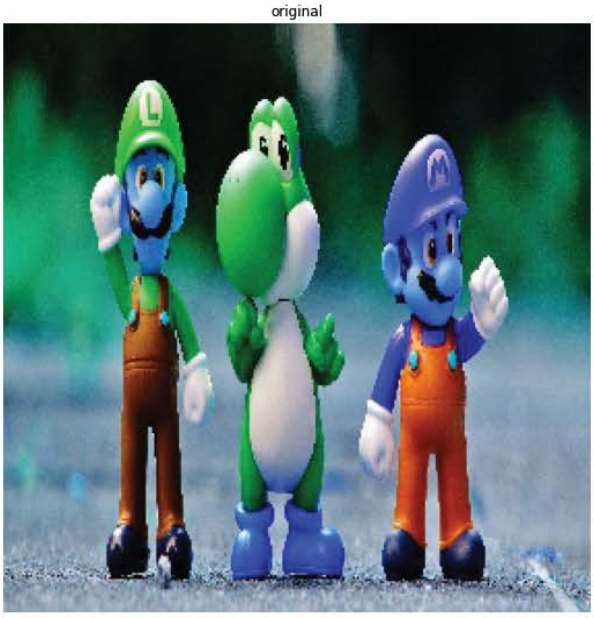


gmm=10



original

gmm=5



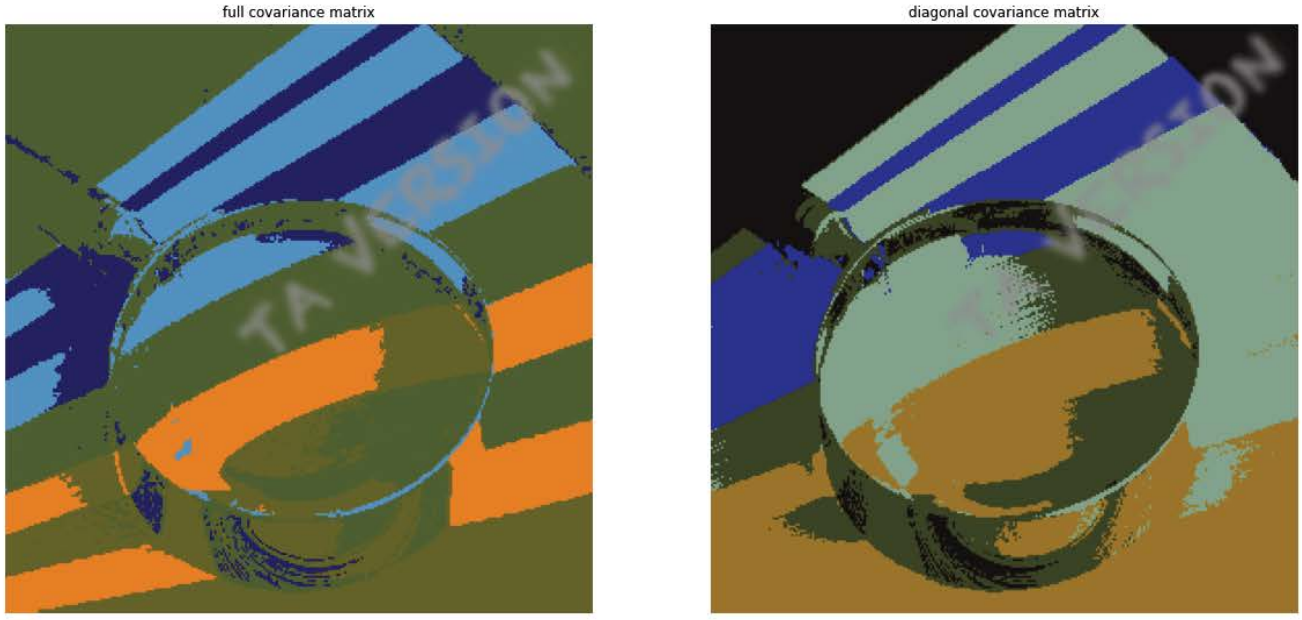
3.4 Compare full covariance matrix with diagonal covariance matrix(Bonus for all) [5 pts]

Compare full covariance matrix with diagonal covariance matrix. Can you explain why the images are different with same clusters? Note: You will have to implement both multinormalPDF and normalPDF, and add a few arguments in the original _ll_joint() and _Mstep() function. You will earn full credit only if you implement both functions AND explain the reason.

In [9]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################

def compare_matrix(image, K):
    """
    Args:
        image: input image of shape(H, W, 3)
        K: number of components

    Return:
        plot: comparison between full covariance matrix and diagonal covariance matrix.
    """
    #full covariance matrix
    gmm_image_full = cluster_pixels_gmm(image, K, full_matrix = True)
    #diagonal covariance matrix
    gmm_image_diag = cluster_pixels_gmm(image, K, full_matrix = False)

    plot_images([gmm_image_full, gmm_image_diag], ['full covariance matrix', 'diagonal covariance matrix'])
```

```
In [10]: 1 compare_matrix(image1, 5)
```

```
iter 9, loss: 1221974.4378: 100%|██████████| 10/10 [00:03<00:00,  3.16it/s]
iter 9, loss: 1237136.1483: 100%|██████████| 10/10 [00:02<00:00,  3.40it/s]
```



4. (Bonus for All) Cleaning Messy data with semi-supervised learning [30pts]

Learning to work with messy data is a hallmark of a well-rounded data scientist. In most real-world settings the data given will usually have some issue, so it is important to learn skills to work around such impasses. This part of the assignment looks to expose you to clever ways to fix data using concepts that you have already learned in the prior questions.

Question

After graduating from Georgia Tech with your shiny new degree, you are recruited to help with safety testing for the Mars rocket at NASA. Of course NASA won't be sending rocket after rocket to stress test your fellow employees' engineering (they also graduated from Tech, so you have full confidence in them), so instead, NASA has decided to run numerous simulations on the current engineering design of the Mars rocket. The simulation collects shuttle data from its sensors, resulting in 8 features which include bypass, rad flow, etc. These features are contained within the first through eighth columns. The ninth column shows the label with 1 being a successful simulation and 0 being an unsuccessful simulation.

However, due to an intern accidentally deleting random data points, 20% of the entries are missing labels and 30% are missing characterization data. Since simply removing the corrupted entries would not reflect the true variance of the data, your job is to implement a solution to clean the data so it can be properly classified.

Your job is to assist NASA in cleaning the data and implementing a semi-supervised learning framework to help them create a general classifier for future simulations.

You are given two files for this task:

- data.csv: the entire dataset with complete and incomplete data
- validation.csv: a smaller, fully complete dataset made after after the intern deleted the datapoints

4.1.a Data Separating [3pts]

The first step is to break up the whole dataset into clear parts. All the data is randomly shuffled in one csv file. In order to move forward, the data needs to be split into three separate arrays:

- labeled_complete: containing the complete characterization data and corresponding labels
- labeled_incomplete: containing partial characterization data (i.e., one of the features is NaN) and corresponding labels
- unlabeled_complete: containing complete characterization data but no corresponding labels (i.e., the label is NaN)

In [2]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from semisupervised import complete_
from semisupervised import incomplete_
from semisupervised import unlabeled_
```

In [3]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

def test_data_separating_methods(complete_, incomplete_, unlabeled_):
    """
    Test data separating methods
    Args:
        complete_, incomplete_, unlabeled_ methods
    """
    data = np.array([
        [1.,    2.,     3.,     1],
        [1.,    np.nan, 3.,     1],
        [7.,    np.nan, 9.,     0],
        [7.,    8.,     9.,     0],
        [26.,   27.,    28.,    np.nan],
        [2.,    3.,     4.,     np.nan],
        [16.,   17.,    18.,    1],
        [np.nan,17.,    18.,    1],
        [11.,   12.,    13.,    np.nan],
        [12.,   np.nan, 14.,    np.nan],
        [22.,   23.,    24.,    0],
        [np.nan,23.,    24.,    0],
        [19.,   20.,    21.,    np.nan]
    ])
    complete_answer = np.array([[ 1.,  2.,  3.,  1.],[ 7.,  8.,  9.,  0.],[16., 17., 18.,  1.],[22., 23., 24.
    incomplete_answer = np.array([[ 1., np.nan,  3.,  1.],[ 7., np.nan,  9.,  0.],[np.nan, 17., 18.,  1.],[np
    unlabeled_answer = np.array([[26., 27., 28., np.nan],[ 2.,  3.,  4., np.nan],[11., 12., 13., np.nan],[19.

    my_complete = complete_(data)
    my_incomplete = incomplete_(data)
    my_unlabeled = unlabeled_(data)

    assert my_complete.shape == complete_answer.shape, f"My complete data shape: {my_complete.shape}; Expected
    assert np.all(np.isclose(my_complete, complete_answer, equal_nan=True)), f"My complete data:\n{my_complete
    print("complete_ method:   Working :)")
    assert my_incomplete.shape == incomplete_answer.shape, f"My incomplete data shape: {my_incomplete.shape};
    assert np.all(np.isclose(my_incomplete, incomplete_answer, equal_nan=True)), f"My incomplete data:\n{my_in
    print("incomplete_ method: Working :)")
    assert my_unlabeled.shape == unlabeled_answer.shape, f"My unlabeled data shape: {my_unlabeled.shape}; Expe
    assert np.all(np.isclose(my_unlabeled, unlabeled_answer, equal_nan=True)), f"My unlabeled data:\n{my_unlab
    print("unlabeled_ method:  Working :)")


test_data_separating_methods(complete_, incomplete_, unlabeled_)
```

```
complete_ method:   Working :)
incomplete_ method: Working :)
unlabeled_ method:  Working :)
```

4.1.b KNN [7pts]

The second step in this task is to clean the Labeled_incomplete dataset by filling in the missing values with probable ones derived from complete data. A useful approach to this type of problem is using a k-nearest neighbors (k-NN) algorithm. For this application, the method consists of replacing the missing value of a given point with the mean of the closest k-neighbors to that point.

In [4]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from semisupervised import CleanData
```

Below is a good expectation of what the process should look like on a toy dataset. If your output matches the answer below, you are on the right track.

NOTE: Your rows of data should match with the expected output, although the order of the rows does not necessarily matter.

In [5]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

def test_cleandata(knn_cleaner):
    """
    Test CleanData implementation
    Args:
        knn_cleaner: CleanData object
    """

    complete_data = np.array([[1.,2.,3.,1],[7.,8.,9.,0],[16.,17.,18.,1],[22.,23.,24.,0]])
    incomplete_data = np.array([[1.,np.nan,3.,1],[7.,np.nan,9.,0],[np.nan,17.,18.,1],[np.nan,23.,24.,0]])
    correct_clean_data = np.array([
        [ 1.   ,2.   ,3.   ,1. ],
        [ 7.   ,8.   ,9.   ,0. ],
        [16.  ,17.  ,18.   ,1. ],
        [22.  ,23.  ,24.   ,0. ],
        [14.5 ,23.  ,24.   ,0. ],
        [ 7.  ,15.5  ,9.   ,0. ],
        [ 8.5 ,17.  ,18.   ,1. ],
        [ 1.   ,9.5  ,3.   ,1. ]
    ])
    clean_data = knn_cleaner(incomplete_data, complete_data, 2)
    assert clean_data.shape == correct_clean_data.shape, f"My cleaned data shape: {clean_data.shape}; Expecte
    if np.all(np.isclose(clean_data, correct_clean_data)):
        print("CleanData: working :)"); return
    clean_data_sorted = np.array(sorted([tuple(row) for row in clean_data]))
    correct_clean_data_sorted = np.array(sorted([tuple(row) for row in correct_clean_data]))
    assert np.all(np.allclose(clean_data_sorted, correct_clean_data_sorted)), f"My cleaned data (sorted):\n{c
    print("CleanData: working :)")

test_cleandata(CleanData())
```

```
CleanData: working :)
```

4.2 Getting acquainted with semi-supervised learning approaches. [5pts]

You will implement a version of the algorithm presented in Table 1 of the paper "Text Classification from Labeled and Unlabeled Documents using EM" (http://www.kamalnigam.com/papers/emcat-mlj99.pdf) by Nigam et al. (2000). While you are recommended to read the whole paper this assignment focuses on items 5.2 and 6.1. Write a brief summary of three interesting highlights of the paper (50-word maximum).

Answer:

4.3 Implementing the EM algorithm. [10 pts]

In your implementation of the EM algorithm proposed by Nigam et al. (2000) on Table 1, you will use a Gaussian Naive Bayes (GNB) classifier as opposed to a naive Bayes (NB) classifier. (Hint: Using a GNB in place of an NB will enable you to reuse most of the implementation you developed for GMM in this assignment. In fact, you can successfully solve the problem by simply modifying the call and _init_components methods.)

In [6]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from semisupervised import SemiSupervised
```

4.4 Demonstrating the performance of the algorithm. [5pts]

Compare the classification error based on the Gaussian Naive Bayes (GNB) classifier you implemented following the Nigam et al. (2000) approach to the performance of a GNB classifier trained using only labeled data. Since you have not covered supervised learning in class, you are allowed to use the scikit learn library for training the GNB classifier based only on labeled data: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html).

To acheive the full 5 points you must implement the `ComparePerformance.accuracy_semi_supervised` and `ComparePerformance.accuracy_GNB` methods and get these scores:

- accuracy_complete_data_only > .87
- accuracy_cleaned_data > .87
- accuracy_semi_supervised > 87%

In [7]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################

from semisupervised import ComparePerformance
```

In [8]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################

# Load training data
all_data = np.loadtxt('data/data.csv', delimiter=',')

# Separate training data into categories: labeled complete, labeled incomplete, and unlabeled points
labeled_complete = complete_(all_data)
labeled_incomplete = incomplete_(all_data)
unlabeled = unlabeled_(all_data)

# Perform data cleaning on labeled incomplete data
cleaned_data = CleanData()(labeled_incomplete, labeled_complete, 10)

# Combine cleaned data with unlabeled data
cleaned_and_unlabeled = np.concatenate((cleaned_data, unlabeled), 0)

# Print data shapes
print(f"All Data shape:                {all_data.shape}")
print(f"Labeled Complete shape:        {labeled_complete.shape}")
print(f"Labeled Incomplete shape:      {labeled_incomplete.shape}")
print(f"Unlabeled shape:               {unlabeled.shape}")
print(f"Cleaned data shape:            {cleaned_data.shape}")
print(f"Cleaned + Unlabeled data shape: {cleaned_and_unlabeled.shape}")

# load validation data
validation = np.loadtxt('data/validation.csv', delimiter=',')

# ========================================================================
# SUPERVISED GNB WITH ONLY THE COMPLETE DATA (SKLEARN)
accuracy_complete_data_only = ComparePerformance.accuracy_GNB(labeled_complete, validation)
# ========================================================================
# SUPERVISED GNB WITH CLEAN DATA (SKLEARN)
accuracy_cleaned_data = ComparePerformance.accuracy_GNB(cleaned_data, validation)
# ========================================================================
# SEMI SUPERVISED GNB WITH ALL DATA (your implementation)
accuracy_semi_supervised = ComparePerformance.accuracy_semi_supervised(cleaned_and_unlabeled, validation, 2)
# ========================================================================
# COMPARISON
print("""===COMPARISON===""")
print(f"Supervised with only complete data, GNB Accuracy: {np.round(100.0 * accuracy_complete_data_only, 3)}%
print(f"Supervised with clean data, GNB Accuracy:         {np.round(100.0 * accuracy_cleaned_data, 3)}%")
print(f"SemiSupervised Accuracy:                          {np.round(100.0 * accuracy_semi_supervised, 3)}%")
```

```
All Data shape:                (13050, 9)
Labeled Complete shape:        (6525, 9)
Labeled Incomplete shape:      (3915, 9)
Unlabeled shape:               (2610, 9)
Cleaned data shape:            (10440, 9)
Cleaned + Unlabeled data shape: (13050, 9)
```

```
iter 20, loss: 453759.2017:  21%|██        | 21/100 [00:00<00:01, 59.32it/s]
```

```
===COMPARISON===
Supervised with only complete data, GNB Accuracy: 88.0%
Supervised with clean data, GNB Accuracy:         88.414%
SemiSupervised Accuracy:                          87.724%
```

In [ ]:
```
1
```