Summer 2021 CS4641/CS7641 A Homework 2

Instructor: Dr. Mahdi Roozbahani

Deadline: June 28th, Monday, 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.

Instructions for the assignment

- This assignment consists of both programming and theory questions.
- Q4 is bonus for both undergraduate and graduate students.
- To switch between cell for code and for markdown, see the menu -> Cell -> Cell Type
- You can directly type Latex equations into markdown cells.
- Typing with Latex/markdown is required for all the written questions. Handwritten answers will not be accepted.
- If a question requires a picture, you could use this syntax "$<imgsrc="" style="width:300px;" />$" to include them within your ipython notebook.

Using the autograder

- You will find three assignments (for grads) on Gradescope that correspond to HW2: "HW2 - Programming", "HW2 - Non-programming" and "HW2 Programming - Bonus for all. Undergrads will have an additional assignment called "Assignment 2 Programming - Bonus for Undergrads"
- You will submit your code for the autograder on "HW2 - Programming" in the following format:
  - kmeans.py
  - gmm.py
  - semisupervised.py
- 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 "HW2 - 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. The non-programming part corresponds to Q2, Q3.3 (both your response and the generated images with your implementation) and Q4.2.**
- **When submitting to Gradescope, please make sure to mark the page(s) corresponding to each problem/sub-problem.**

Points Distribution

Q1: KMeans Clustering [55 pts]

- **pairwise_dist** [5 pts] - *programming*
- **KMeans Implementation** [30pts] - *programming*
  - _init_centers [5pts]
  - _update_assignment [10pts]
  - _update_centers [10pts]
  - _get_loss function [5pts]
- **Elbow Method** [10 pts]
  - find_num_optimal_clusters [6pts] - *programming*
  - written questions [4pts] - *non-programming*
- **Silhouette Coefficient Evaluation** [10 pts for CS 7641; 10 pts Bonus for CS 4641] - *programming*
  - intra_cluster_dist [4pts]
  - inter_cluster_dist [4pts]
  - silhouette_coefficient [2pts]

Q2: EM Algorithm [20pts; 5 pts Bonus for All]

- 2.1.1 [10pts] - *non-programming*
- 2.1.2 [10pts] - *non-programming*
- 2.1.3 [5pts Bonus for All] - *non-programming*

Q3: GMM implementation [50pts]

- Helper Functions [15pts] - *programming*
  - softmax [5pts]
  - logsumexp [5pts]
  - normalPDF [5pts] - *for CS4641 students only*
  - multinormalPDF [5pts] - *for CS7641 students only*
- GMM Implementation [25pts] - *programming*
  - _init_components [5pts]
  - _ll_joint [10pts]
  - _E_step [5pts]

    - _M_step [10pts]
- Jackson Pollock and pixel clustering [10pts] - *non-programming*

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

- 4.1: KNN [10pts] - *programming*
- 4.2: Getting acquainted with semi-supervised learning approaches [5pts] - *non-programming*
- 4.3: Implementing the EM algorithm [10pts] - *programming*
- 4.4: *Demonstrating the performance of the algorithm* [5pts] - *programming*

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)

There is also a VS Code and Anaconda Setup Tutorial (https://edstem.org/us/courses/5944/discussion/447961) on Ed under the "Links" category

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

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

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

%matplotlib inline
%load_ext autoreload
%autoreload 2

print("If you do not have plotly installed, please install via 'pip install plotly'")
print()
import sys
import matplotlib
import numpy as np
import plotly.express as px
import matplotlib.pyplot as plt
import csv
from mpl_toolkits.mplot3d import axes3d
from tqdm import tqdm
import pandas as pd

print('Version information')

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


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

# Load image
import imageio
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
If you do not have plotly installed, please install via 'pip install plotly'

Version information
python: 3.8.3 (default, Jul  2 2020, 11:26:31)
[Clang 10.0.0 ]
matplotlib: 3.4.2
numpy: 1.18.5
```

1. KMeans Clustering [5 + 30 + 10 + 10 pts]

Dominic has been contacted by Forbes Magazine to interpret the 2021 Fortune 1000 companies and their data using his new-found knowledge in unsupervised learning techniques from his graduate course CS7641. Eloquent in the finances of top companies, Dominic decides to use KMeans to compare and contrast these companies to write an amazing report for Forbes Magazine. Dominic unzips his bulky backpack and opens his prized possession--his laptop. He rests this prized possession on his desk, lifts the cover to awaken his laptop from dormancy. He plans to examine the bottom 500 companies to see the patterns between revenue and profit to see which company is most likely to jump into the Fortune Top 500.

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 $X \in \mathbb{R}^{NxD}$ and $Y \in \mathbb{R}^{MxD}$, obtain the pairwise distance matrix $dist \in \mathbb{R}^{NxM}$ using the euclidean distance metric, where $dist_{i,j} = ||X_i - Y_j||_2$.

DO NOT USE FOR LOOP in your implementation -- they are slow and will make your code too slow to pass our grader. Use array broadcasting instead.

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

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))
```

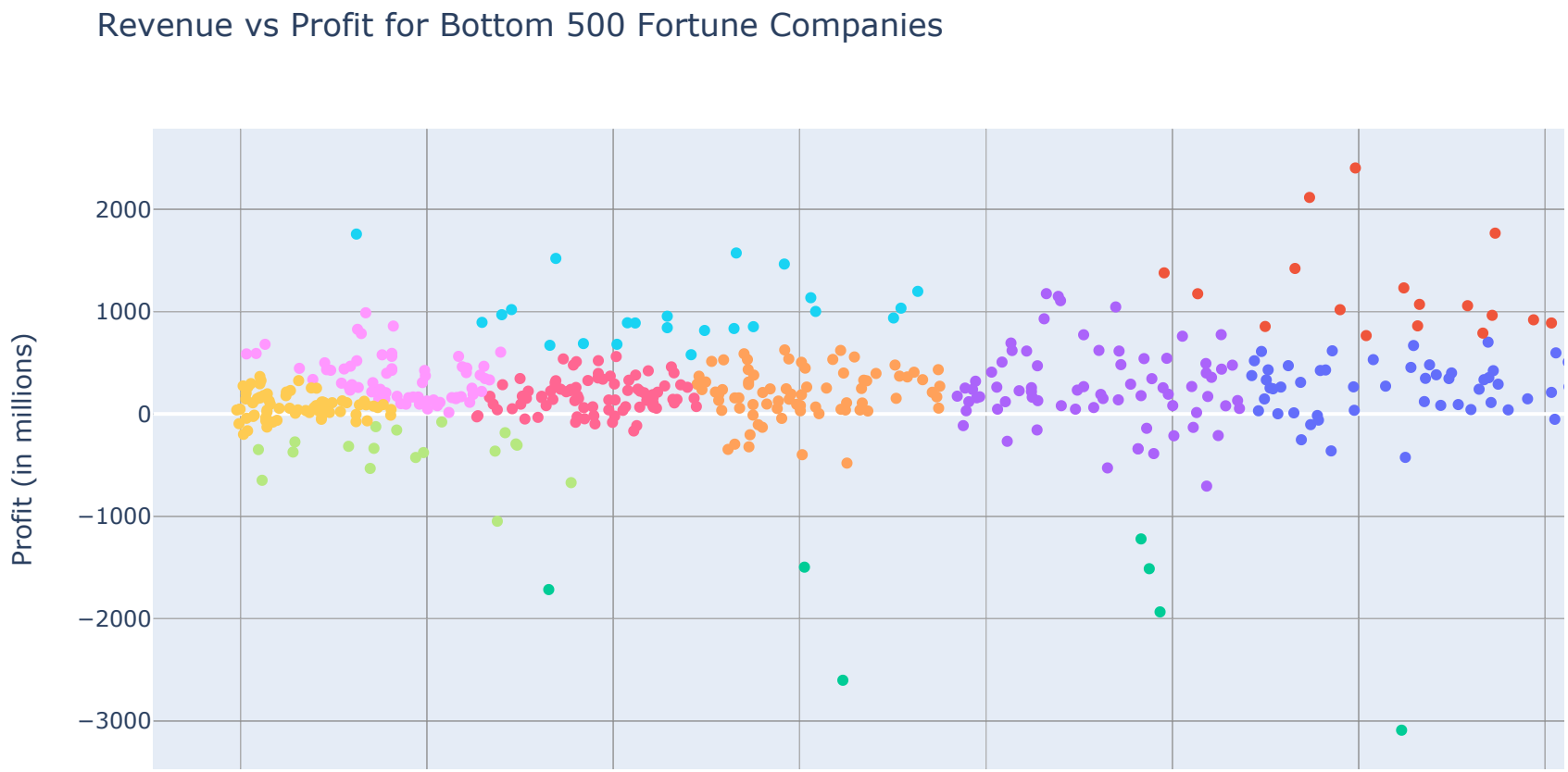
```
*** Expected Answer ***
==x==
[[ 1.62434536 -0.61175641]
 [-0.52817175 -1.07296862]]
==y==
[[ 0.86540763 -2.3015387 ]
 [ 1.74481176 -0.7612069 ]
 [ 0.3190391  -0.24937038]]
==dist==
[[1.85239052 0.19195729 1.35467638]
 [1.85780729 2.29426447 1.18155842]]

*** 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]]
```

In [6]:
```
### 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.
```

```
  File "<ipython-input-6-0145f3a6cebf>", line 3
    In this section, you are asked to implement _init_centers [5pts], _update_assignment [10pts], _u
pdate_centers [10pts] and _get_loss function [5pts].
       ^
SyntaxError: invalid syntax
```

In [7]:
```
###################################
### NO NEED TO CHANGE THIS CELL ###
###################################

from kmeans import KMeans

def read_file():
    companies = []
    with open(r'./data/Fortune_1000.csv') as fin:
        fortune_dataset = csv.reader(fin)
        dataset_in_list = []
        header = True
        for row in fortune_dataset:
            if header:
                header = False
            else:
                if row[4] != '' and row[3] != '-' and row[3] != '' and int(row[1]) > 500:
                    company = str(row[0])
                    profit = int(float(row[4]))
                    revenue = int(float((row[3])))
                    dataset_in_list.append([company, profit, revenue])
    data = np.array(dataset_in_list)
    companies = data[1:, 0]
    data = data[1:, 1:].astype(int)
    return data, companies

def visualize (data, cluster_idx, centers, K, companies):
    df = pd.DataFrame(np.concatenate((companies, data, cluster_idx), axis=1))
    df = df.rename(columns={0: "Company", 1:"Profit (in millions)", 2:"Revenue (in millions)", 3:"Clu
ster"})
    visual = px.scatter(
        df, x='Revenue (in millions)', y='Profit (in millions)',
        color="Cluster",
        hover_name="Company",
        title="Revenue vs Profit for Bottom 500 Fortune Companies",
    )
    visual.update_xaxes(type='linear')
    visual.update_yaxes(type='linear')
    visual.show()
```

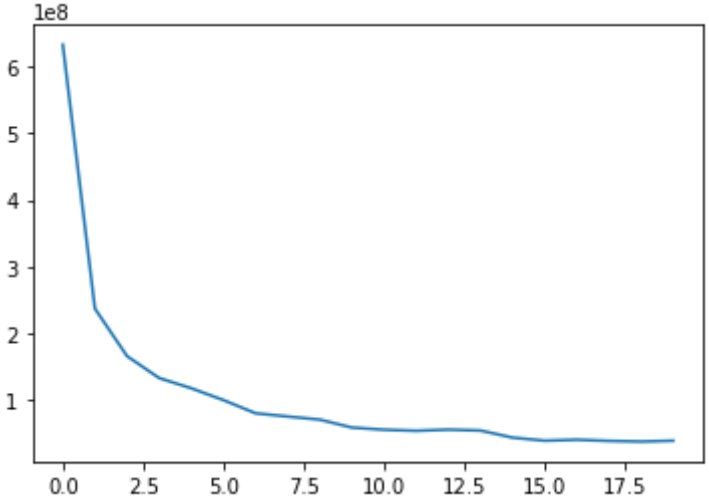
In [8]:
```
###################################
### NO NEED TO CHANGE THIS CELL ###
###################################

data, companies = read_file()
cluster_idx2, centers2, loss2 = KMeans()(data, 10, max_iters=5000)
visualize(data, cluster_idx2.reshape(-1, 1), centers2.reshape(-1, 1), 2, companies.reshape(-1, 1))
```



1.3 Elbow Method [6 pts Programming + 4 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 upgrade Dominic's software by implementing the elbow method to find the optimal number of clusters in the function find_optimal_num_clusters below.

In [9]:
```
###################################
### NO NEED TO CHANGE THIS CELL ###
###################################

from kmeans import find_optimal_num_clusters
find_optimal_num_clusters(data, max_K=20)
```



Out[9]:
```
[632251349.8430583,
 237454685.9637826,
 166420360.38688534,
 133592458.09045123,
 118310969.66735226,
 100619105.4952231,
 80871062.1696319,
 76035155.24185564,
 71379407.33917774,
 59498211.34029791,
 56261286.02112286,
 54664498.709260605,
 56389977.71319344,
 55076559.09917016,
 44461709.177699685,
 39905468.55668835,
 41273606.17911718,
 39565080.794156656,
 38565676.249300234,
 39838708.75660725]
```

Written Questions [4 pts]:

1) Approximately what value does the elbow method give? Was the number of clusters used by Dominic (Dominic used 10 clusters) greater or lower than the elbow method value?

SOLUTION:

The output of my code was almost 10, which is equal to what Dominic used.

2) Dominic comes up with another idea - to optimize the number of clusters, just choose a k that makes the loss close to 0. Would this work? Why or why not?

SOLUTION:

It is not a good idea, since it makes number of clusters equal to number of data points. The loss function is zero, but no meaningful clustering happens.

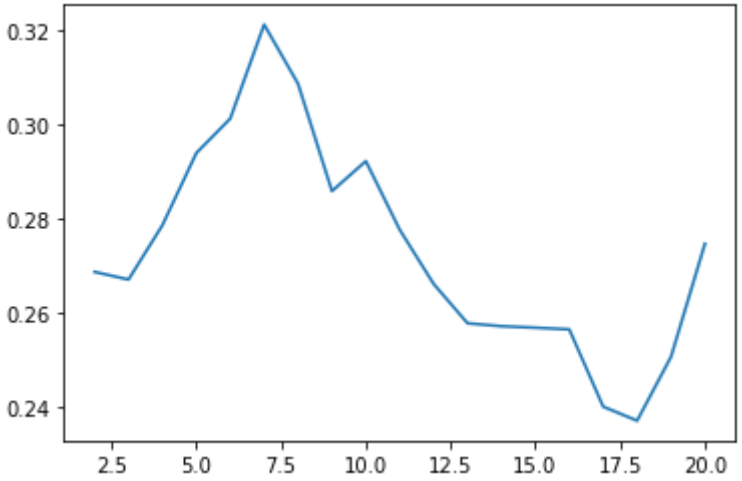
1.4 Silhouette Coefficient Evaluation [10 pts for CS 7641; 10 pts Bonus for CS 4641]

The average silhouette of the data is another useful criterion for assessing the natural number of clusters. The silhouette of a data instance is a measure of how closely it is matched to data within its cluster and how loosely it is matched to data of the neighbouring cluster.

The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters.

In [10]:
```
###################################
### NO NEED TO CHANGE THIS CELL ###
###################################

from kmeans import intra_cluster_dist, inter_cluster_dist, silhouette_coefficient

def plot_silhouette_coefficient(data, max_K=15):
    """
    Plot silhouette coefficient for different number of clusters, no need to implement
    """
    clusters = np.arange(2, max_K+1)

    silhouette_coefficients = []
    for k in range(2, max_K+1):
        labels, _, _ = KMeans()(data, k)
        silhouette_coefficients.append(silhouette_coefficient(data, labels))
    plt.plot(clusters, silhouette_coefficients)
    return silhouette_coefficients

np.random.seed(1)
data = np.random.rand(200,3) * 100
plot_silhouette_coefficient(data, max_K=20)
```

Out[10]:
```
[0.2686854256351734,
 0.26705661036946593,
 0.2786296337024161,
 0.293975278845551,
 0.3012965213229007,
 0.3213223961130688,
 0.3087309398520805,
 0.2858764509135404,
 0.29225860724747216,
 0.2776452854209834,
 0.2661204643356518,
 0.2577510967242535,
 0.257132123433868,
 0.25681230260861093,
 0.2564554609259566,
 0.24001008060895793,
 0.237025705178037,
 0.25061777693877985,
 0.27467872305719593]
```

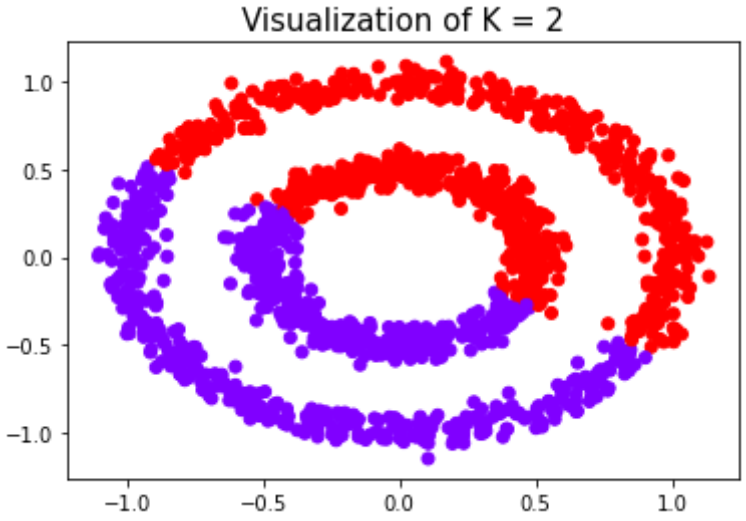


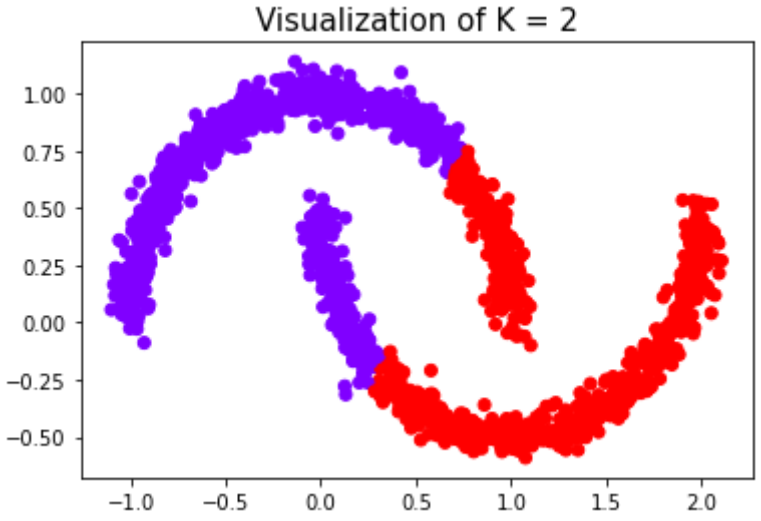
Limitation of K-Means

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 [11]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

# visualize limitation of kmeans, do not have to implement

from sklearn.datasets.samples_generator 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):# 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')
    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)
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/deprecation.py:143: FutureWarning:

The sklearn.datasets.samples_generator module is  deprecated in version 0.22 and will be removed in version 0.24. The corresponding classes / functions should instead be imported from sklearn.datasets. Anything that cannot be imported from sklearn.datasets is now part of the private API.
```





2 EM algorithm [25 pts]

2.1 EM Algorithm in Coin Toss problem [20 pts; 5 Bonus for All]

Suppose we have a bunch of coins C consisting three kinds of coins. Mathematically, it obeys a mixed Bernoulli distribution:

$$X \sim F = \pi_1 F_1 + \pi_2 F_2 + (1 - \pi_1 - \pi_2) F_3$$

where $\pi \in [0, 1]$, and $F_1 = Ber(p_1)$, $F_2 = Ber(p_2)$, $F_3 = Ber(p_3)$. That is to say, each coin belongs to F_1, F_2 or F_3. Here $Ber(p)$ means the coin gives 1 (head) with probability p and gives 0 (tail) with probability $1 - p$. We initialized parameters $p_1 = \frac{1}{2}$, $p_2 = \frac{5}{6}$, $p_3 = \frac{1}{3}$, $\pi_1 = \frac{1}{4}$, $\pi_2 = \frac{1}{2}$. Now, we draw 3 coins X_1, X_2, X_3 independently from C and have 6 independent trials for each of them. The result shows:

$Coins$	X_1	X_2	X_3
$Trial1$	0	1	1
$Trial2$	0	1	1
$Trial3$	1	0	1
$Trial4$	0	1	0
$Trial5$	1	0	1
$Trial6$	1	1	1

2.1.1. Use the Expectation step of EM algorithm for one step and calculate the responsibilities (τ) [10 pts].

SOLUTION:

In general the resposibilty function is the probability of z for “A” data point x, meaning that this is probability that point "n" is generated from a distribuition k normalized by P(X)(the probability of the entire data set occurring). Suppose that z_{ij} shows the esponsibilty of latent variable of coin X_i to belongs to type j.

With that, responsibility of each class is:

$$\tau = p(z|x, \theta) = \frac{p(z, x|\theta)}{\sum_z p(x, z|\theta)}$$

Also we can compute F_i of each category:

$$For i \in 1, 2, 3$$
$$F_i(X_1) = (P_i)^3 \times (1 - P_i)^3$$
$$F_i(X_2) = (P_i)^4 \times (1 - P_i)^2$$
$$F_i(X_3) = (P_i)^5 \times (1 - P_i)^1$$

So in E-STEP we have:

$$\tau(z_{11}) = P(Z = type1|X1) = \frac{\pi_1 \times F_1(X_1)}{\sum_i \pi_i \times F_i(X_1)} = \frac{0.25 \times (0.5)^3 \times (0.5)^3}{0.25 \times (0.5)^3 \times (0.5)^3 + 0.5 \times (5/6)^3 \times (1/6)^3 + 0.25 \times (1/3)^3 \times (2/3)^3} = 0.4889$$

$$\tau(z_{12}) = P(Z = type2|X1) = \frac{\pi_2 \times F_2(X_1)}{\sum_i \pi_i \times F_i(X_1)} = \frac{0.5 \times (5/6)^3 \times (1/6)^3}{0.25 \times (0.5)^3 \times (0.5)^3 + 0.5 \times (5/6)^3 \times (1/6)^3 + 0.25 \times (1/3)^3 \times (2/3)^3} = 0.1676$$

$$\tau(z_{13}) = P(Z = type3|X1) = \frac{\pi_3 \times F_3(X_1)}{\sum_i \pi_i \times F_i(X_1)} = \frac{0.25 \times (1/3)^3 \times (2/3)^3}{0.25 \times (0.5)^3 \times (0.5)^3 + 0.5 \times (5/6)^3 \times (1/6)^3 + 0.25 \times (1/3)^3 \times (2/3)^3} = 0.3433$$

And:

$$\tau(z_{21}) = P(Z = type1|X2) = \frac{\pi_1 \times F_1(X_2)}{\sum_i \pi_i \times F_i(X_2)} = \frac{0.25 \times (0.5)^4 \times (0.5)^2}{0.25 \times (0.5)^4 \times (0.5)^2 + 0.5 \times (5/6)^4 \times (1/6)^2 + 0.25 \times (1/3)^4 \times (2/3)^2} = 0.3261$$

$$\tau(z_{22}) = P(Z = type2|X1) = \frac{\pi_2 \times F_2(X_2)}{\sum_i \pi_i \times F_i(X_2)} = \frac{0.5 \times (5/6)^4 \times (1/6)^2}{0.25 \times (0.5)^4 \times (0.5)^2 + 0.5 \times (5/6)^4 \times (1/6)^2 + 0.25 \times (1/3)^4 \times (2/3)^2} = 0.5592$$

$$\tau(z_{23}) = P(Z = type3|X2) = \frac{\pi_3 \times F_3(X_2)}{\sum_i \pi_i * F_i(X_2)} = \frac{0.25 \times (1/3)^4 \times (2/3)^2}{0.25 \times (0.5)^4 \times (0.5)^2 + 0.5 \times (5/6)^4 \times (1/6)^2 + 0.25 \times (1/3)^4 \times (2/3)^2} = 0.1145$$

And:

$$\tau(z_{31}) = P(Z = type1|X3) = \frac{\pi_1 \times F_1(X_3)}{\sum_i \pi_i \times F_i(X_3)} = \frac{0.25 \times (1/2)^5 \times (1/2)}{0.25 \times (0.5)^5 \times (0.5) + 0.5 \times (5/6)^5 \times (1/6) + 0.25 \times (1/3)^5 \times (2/3)} = 0.1025$$

$$\tau(z_{32}) = P(Z = type2|X1) = \frac{\pi_2 \times F_1(X_3)}{\sum_i \pi_i \times F_i(X_3)} = \frac{0.5 \times (5/6)^5 \times (1/6)}{0.25 \times (0.5)^5 \times (0.5) + 0.5 \times (5/6)^5 \times (1/6) + 0.25 \times (1/3)^5 \times (2/3)} = 0.8794$$

$$\tau(z_{33}) = P(Z = type3|X3) = \frac{\pi_3 \times F_3(X_3)}{\sum_i \pi_i \times F_i(X_3)} = \frac{0.25 \times (1/3)^5 \times (2/3)}{0.25 \times (0.5)^5 \times (0.5) + 0.5 \times (5/6)^5 \times (1/6) + 0.25 \times (1/3)^5 \times (2/3)} = 0.0180$$

$$\tau(z) = \begin{pmatrix} 0.4889 & 0.1676 & 0.3433 \\ 0.3261 & 0.5592 & 0.1145 \\ 0.1025 & 0.8794 & 0.0180 \end{pmatrix}$$

2.1.2. Use the Maximization step of the EM algorithm to update $F = F(p_1 = \frac{1}{2}, p_2 = \frac{5}{6}, p_3 = \frac{1}{3}, \pi_1 = \frac{1}{4}, \pi_2 = \frac{1}{2})$ to $F'(p_1', p_2', p_3', \pi_1', \pi_2')$. What are the values of $p_1', p_2', p_3', \pi_1', \pi_2'$. (Round the answer to three decimal places.) [10 pts].

(**Hint**: $\theta^{new} = argmax_\theta \sum_Z p(Z|X, \theta^{old}) \ln p(X, Z|\theta)$)

SOLUTION:

In the Maximization step we take derivative of the log liklihood w.r.t. θ and equate it to 0. So, we have:

$$\frac{\delta \log[p(x|\theta)]}{\delta\theta} = \frac{\frac{\delta[p(x|\theta)]}{\delta\theta}}{p(x|\theta)} = 0$$

.In addition, based on the Hint: $\theta^{new} = argmax_\theta \sum_Z p(Z|X, \theta^{old}) \ln p(X, Z|\theta)$. So:

$$\sum_Z p(Z|X, \theta) = \sum_{i=1}^{N} \sum_{k}^{K} \tau(z_{kn})$$

Where N is the total number of points (N=6) and K is the number of different types (K=3). Now,

$$\sum_z ln(p(Z|x, \theta)) = \sum_{i=1}^{N} \sum_{k=1}^{K} ln(\pi_k) + ln(P(x|p)) = \sum_{i=1}^{N} \sum_{k=1}^{K} ln(\pi_k) + x_n ln(\pi_k) + (1 - x_n) ln(1 - \pi_k)$$

So we will have:

$$p_k = \frac{1}{\sum_{n=1}^{N} \tau(z_{kn})} \times \sum_{n=1}^{N} \tau(z_{kn}) x_n$$

$$\pi_k = \frac{\sum_{n=1}^{N} \tau(z_{kn})}{N}$$

Here, N=3 since there are 3 coins X_1, X_2 and X_3

$$\pi_1' = \frac{\tau(z_{11}) + \tau(z_{21}) + \tau(z_{31})}{N} = \frac{0.48890 + 0.3261 + 0.1025}{3} = 0.3058$$

$$\pi_2' = \frac{\tau(z_{12}) + \tau(z_{22}) + \tau(z_{32})}{N} = \frac{0.1676 + 0.5592 + 0.8794}{3} = 0.5354$$

$$\pi_3' = \frac{\tau(z_{13}) + \tau(z_{23}) + \tau(z_{33})}{N} = \frac{0.3433 + 0.1145 + 0.0180}{3} = 0.1586$$

Alternatively,

$$\pi_3' = 1 - (\pi_1' + \pi_2') = 1 - (0.360 + 0.405) = 0.235$$

x_1 = Number of heads in coin X_1 / Total number of trials = 3/6.

x_2 = Number of heads in coin X_2 / Total number of trials= 4/6.

x_3 = Number of heads in coin X_3/ Total number of trials = 5/6.

$$p_1' = \frac{\tau(z_{11})x_1 + \tau(z_{21})x_2 + \tau(z_{31})x_3}{\tau(z_{11}) + \tau(z_{12}) + \tau(z_{13})} = \frac{0.4889 \times 3 + 0.3261 \times 4 + 0.1025 \times 5}{(0.4889 + 0.3261 + 0.1025) \times 6} = 0.5964$$

$$p_2' = \frac{\tau(z_{12})x_1 + \tau(z_{22})x_2 + \tau(z_{32})x_3}{\tau(z_{21}) + \tau(z_{22}) + \tau(z_{23})} = \frac{0.1676 \times 3 + 0.5592 \times 4 + 0.8794 \times 5}{(0.1676 + 0.5592 + 0.8794) \times 6} = 0.7405$$

$$p_3' = \frac{\tau(z_{13})x_1 + \tau(z_{23})x_2 + \tau(z_{33})x_3}{\tau(z_{31}) + \tau(z_{32}) + \tau(z_{33})} = \frac{0.3433 \times 3 + 0.1145 \times 4 + 0.0180 \times 5}{(0.3433 + 0.1145 + 0.0180) \times 6} = 0.5527$$

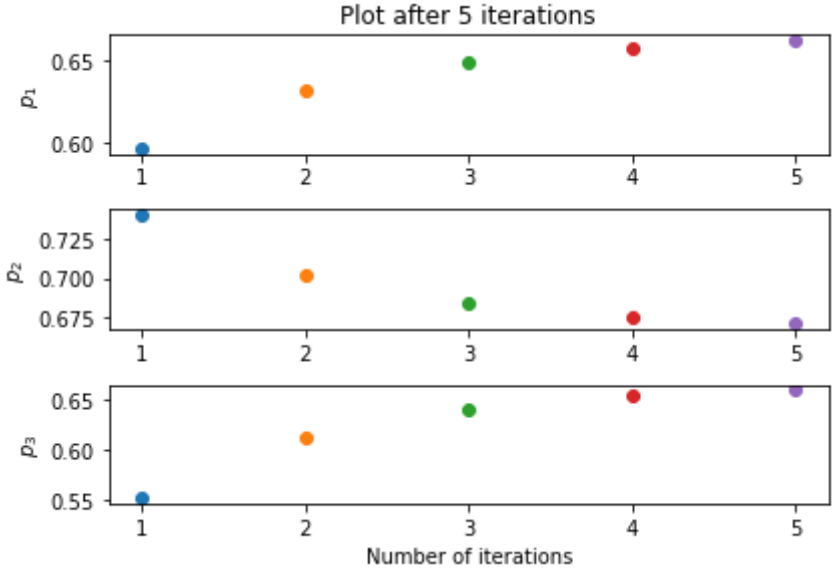
2.1.3. **(Bonus for all)** Use the Jupyter Notebook code cells below to make scatter plots of the values of p_1, p_2, p_3 on different axes for each iteration after 5, 10 and 20 iterations of EM algorithm. Use the first cell to implement the EM Algorithm and the second cell to generate the plot. We will only be looking at the plots you generate to determine points given. **We require you to attach your Jupyter Notebook code cell to the HW2 Non-Programming Gradescope Assignment to receive full credit.** [5 pts]

(**Hint 1**: Why are the values increasing/decreasing? Will the values be stable if we implement more steps? No need to calculate the real number)

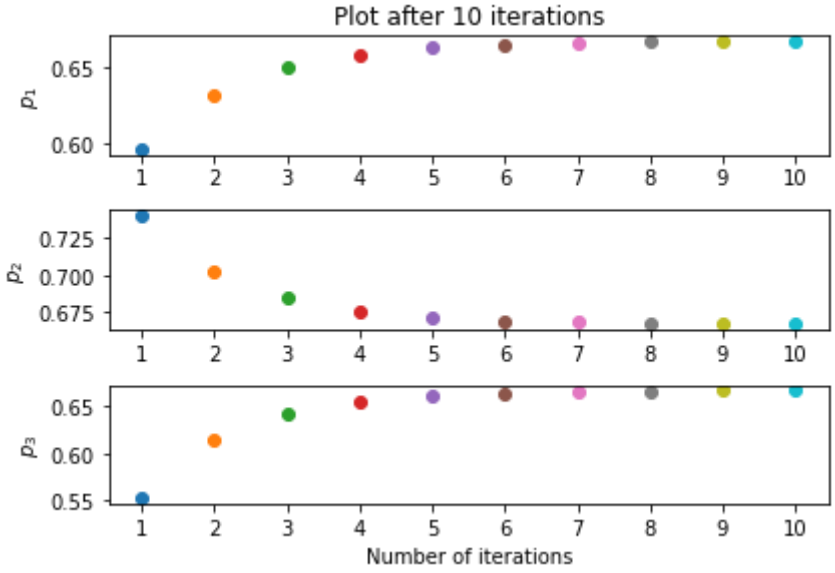
In [43]:
```
### Template for 2.1.3 ###

import numpy as np
import matplotlib.pyplot as plt


def bernoulli(p, points, num_trials):
    '''
        p = probability of getting heads in a coin (1x1)
        points = The data in the table (3x6)
        num_trials = 6

        Return: The value of the Bernoulli Distribution according to the formula.
    '''
    p1=points.shape[1]
    pr=p.reshape(-1,1)
    probability = 1
    for index in range(p1):
        probability = probability * (((1-pr)*(1-points[:,index]))+(pr*points[:,index]))
    return probability



def E_step(points, Pi, P):
    '''
        points: (3x6)
        Pi: (3,)
        P: (3,)

        Return: tau (3x3), using the formula used in part a
    '''
    Pi0 = Pi.shape[0]
    Pir = Pi.reshape(-1,1)
    p0 = points.shape[0]
    trials = points.shape[1]
    probability = (Pir) * bernoulli(P, points, trials)
    tau_values = np.zeros([Pi0,Pi0])
    for index in range (Pi0):
        tau_values[index,:] = (probability[index,:].reshape(1,-1))/(probability.sum(axis=0, keepdims
= True)).reshape(-1)
    return  tau_values
```

```
def M_step(points, tau):
    '''
        points: (3x6)
        tau: (3x3), obtained from the E_step

        Return: P (3,), Pi (3,)

    '''
    p0 = points.shape[0]
    pi = np.zeros(p0)
    mu = np.zeros(p0)
    x = points.mean(axis =1, keepdims = True)

    for index in range (p0):
        pi[index]= tau[index,:].sum(axis=0)/p0
        mu[index]= (tau[index,:].dot(x))/(tau[index,:].sum(axis=0, keepdims = True))
    return pi, mu
```

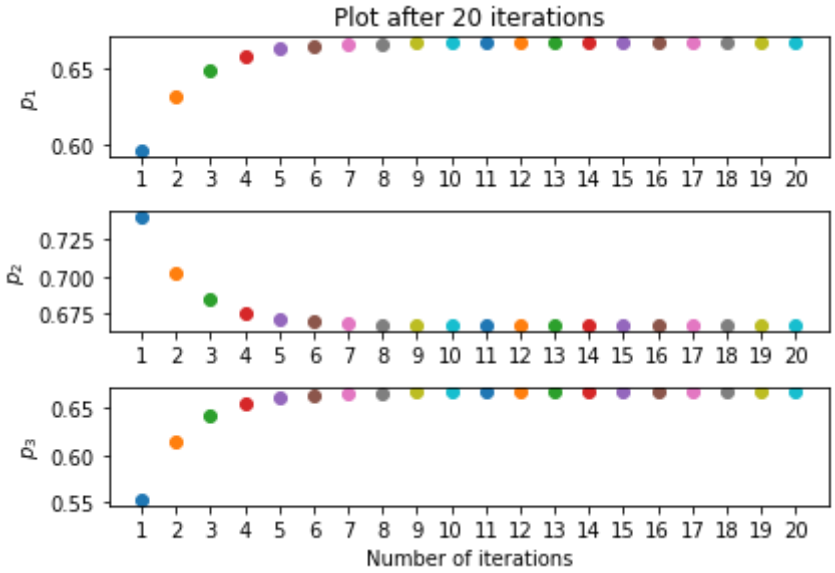
In [44]:
```
import warnings
warnings.filterwarnings('ignore')


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

# This cell will generate the plot required in 2.1.3 #

x = np.array([[0, 0, 1, 0, 1, 1], [1, 1, 0, 1, 0, 1], [1, 1, 1, 0, 1, 1]])
num_trials = x.shape[1]

for iter in [5, 10, 20]:
    x_label = np.arange(1, iter+1)
    p = np.array([1/2, 5/6, 1/3])
    pi = np.array([1/4, 1/2, 1/4])
    for i in range(1, iter+1):
        tau = E_step(x, pi, p)
        pi, p = M_step(x, tau)
        plt.subplot(311)
        plt.scatter(i, p[0])
        plt.xticks(x_label, x_label)
        plt.ylabel('$p_1$')
        plt.title(f'Plot after {iter} iterations')
        plt.subplot(312)
        plt.scatter(i, p[1])
        plt.xticks(x_label, x_label)
        plt.ylabel('$p_2$')
        plt.subplot(313)
        plt.scatter(i, p[2])
        plt.xticks(x_label, x_label)
        plt.ylabel('$p_3$')
    plt.tight_layout()
    plt.xlabel('Number of iterations')
    plt.show()
    print(f'New pi after {i} iterations:\n', pi)
    print(f'New p after {i} iterations:\n', p)
```




```
New pi after 5 iterations:
 [0.31209264 0.53618648 0.15172088]
New p after 5 iterations:
 [0.66228983 0.67100871 0.66032497]
```



```
New pi after 10 iterations:
 [0.31210154 0.53620173 0.15169672]
New p after 10 iterations:
 [0.66652977 0.66680218 0.66646933]
```




```
New pi after 20 iterations:
 [0.31210155 0.53620175 0.1516967 ]
New p after 20 iterations:
 [0.66666653 0.6666668  0.66666647]
```

3. GMM implementation [15 + 25 + 10 pts]

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:

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

Note that each data point should have one probability for each component/cluster. For this homework, you will work with $\tau\left(z_k\right)$ 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 students: you may assume that the covariance matrix Σ is a diagonal matrix, which means the features are independent. (i.e. the red intensity of a pixel is independent of its blue intensity, etc).
- For graduate students: please assume a full covariance matrix.
- The class notes assume that your dataset X is (D, N). However, the homework dataset is (N, 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.** You can always find a way to avoid looping over the observation data points in our homework problem. If you have to loop over D or K, that would be fine.
2. You can initiate $\pi(k)$ the same for each k, i.e. $\pi(k) = \frac{1}{K}, \forall k = 1, 2, \ldots, K$.
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 have N pixels and $D = 3$ features, which correspond to red, green, and blue color intensities. It means that each image is a $(N \times 3)$ dataset matrix. In the following parts, remember $D = 3$ 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, some calculations that required different shapes of arrays can 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. This could enhance the functionality of your code and help you debug. Also notice that **a numpy array in shape $(N, 1)$ is NOT the same as that in shape $(N,)$** 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 X: (N, D)
    - μ: (K, D).
    - Σ: (K, D, D)
    - τ: (N, K)
    - π: array of length K
    - ll_joint: (N, 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 (N, 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 $logit \in \mathbb{R}^{N \times D}$, calculate $prob \in \mathbb{R}^{N \times D}$, where $prob_{i,j} = \frac{\exp(logit_{i,j})}{\sum_{d=1}^{D} exp(logit_{i,d})}$.

Note: it is possible that $logit_{i,j}$ 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 $logits$, and then add it back in your result.

3.1.2. logsumexp [5 pts]

Given $logit \in \mathbb{R}^{N \times D}$, calculate $s \in \mathbb{R}^{N}$, where $s_i = \log\left(\sum_{j=1}^{D} \exp(logit_{i,j})\right)$. Again, pay attention to the numerical problem. You may want to use similar trick as in the softmax function. Note: This function is used in the call() function which is given, so you will not need it in your own implementation. It helps calculate the loss of log-likehood.

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 + 1e-32). 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 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 [25 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, and initialize the covariance matrix with np.eye() 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.
3. Other ways of initialization are acceptable and welcome.

3.2.2. Formulate the log-likelihood function _ll_joint() [5 pts]

The log-likelihood function is given by:

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

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 Σ 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 [35]:
```
##################################
### DO NOT CHANGE THIS CELL ###
##################################

from gmm import GMM
```

In [29]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

np.random.seed(1)

data = np.random.randn(4, 3)

# test softmax utility
my_softmax = GMM(data, 3).softmax(data)
expected_softmax = np.array([[0.81761761, 0.08738232, 0.09500007],
       [0.12135669, 0.84312089, 0.03552242],
       [0.75647821, 0.0617229 , 0.18179889],
       [0.14923883, 0.82635643, 0.02440474]])

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([[1.82570589],
 [1.03605256],
 [2.02389331],
 [1.65283702]])

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


# init random
points = np.random.randn(12, 3)

mu = np.array([[-0.74715829,  1.6924546,   0.05080775],
 [-1.09989127, -0.17242821, -0.87785842],
 [-0.3224172,  -0.38405435,  1.13376944]])

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 [30]:
```
##############################
### 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([0.0037374 , 0.00681159, 0.01294674, 0.00700474, 0.00095577,
       0.00813925, 0.00544499, 0.00385966, 0.00561288, 0.00228524,
       0.06349364, 0.00250289])

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


# test ll-joint
my_lljoint = GMM(points, 3)._ll_joint(pi, mu, sigma,False)
expected_lljoint = np.array([[-6.68797812, -6.20337699, -3.85542789],
       [-6.08774253, -3.85542789, -6.20337698],
       [-5.44552376, -4.81763731, -6.88557037],
       [-6.05977986, -7.90402129, -5.95737486],
       [-8.05161039, -6.27262476, -5.43812535],
       [-5.90966969, -4.86498535, -4.23232988],
       [-6.31167075, -3.98209541, -5.58159406],
       [-6.65578947, -4.38655011, -4.69047683],
       [-6.28130441, -7.15820124, -4.50096327],
       [-7.17989628, -5.55202701, -6.48346667],
       [-3.85542789, -6.08774255, -6.6879781 ],
       [-7.08892294, -8.46315357, -4.53725014]])
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([[0.05098852, 0.08278125, 0.86623023],
       [0.08918842, 0.83136246, 0.07944912],
       [0.3214852 , 0.60234958, 0.07616522],
       [0.44131069, 0.06979118, 0.48889813],
       [0.04861361, 0.28797946, 0.66340693],
       [0.10876892, 0.30917578, 0.5820553 ],
       [0.074913  , 0.76962456, 0.15546244],
       [0.0561508 , 0.54309286, 0.40075634],
       [0.13609235, 0.05662422, 0.80728343],
```

```
       [0.12346309, 0.62879889, 0.24773802],
       [0.85752822, 0.09199548, 0.0504763 ],
       [0.07101476, 0.01796916, 0.91101608]])


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([0.19829313, 0.35762874, 0.44407813])
expected_mu = np.array([[-0.20989007,  0.79579186,  0.06554929],
       [-0.35741548, -0.1535599 , -0.4876455 ],
       [-0.28772515, -0.07512445,  0.79292111]])
expected_sigma = np.array([[[0.64857055, 0.        , 0.        ],
        [0.        , 0.63446774, 0.        ],
        [0.        , 0.        , 0.62167826]],

       [[0.53473119, 0.        , 0.        ],
        [0.        , 0.23538075, 0.        ],
        [0.        , 0.        , 0.38671205]],

       [[0.62612107, 0.        , 0.        ],
        [0.        , 0.24611766, 0.        ],
        [0.        , 0.        , 0.88668642]]])
print("Your M step works within the expected range: ", np.allclose(my_pi, expected_pi) and np.allclos
e(my_mu, expected_mu) and np.allclose(my_sigma, expected_sigma))
print(f"Pi is correct: {np.allclose(my_pi, expected_pi)}")
print(f"Mu is correct: {np.allclose(my_mu, expected_mu)}")
print(f"Sigma is correct: {np.allclose(my_sigma, expected_sigma)}")
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-30-a5ac721767d8> in <module>
      5 # For undergrads
      6 # test normalPDF
----> 7 my_normalpdf = GMM(points, 3).normalPDF(points, mu[0], sigma[0])
      8 expected_normal_pdf = np.array([0.0037374 , 0.00681159, 0.01294674, 0.00700474, 0.00095577,
      9        0.00813925, 0.00544499, 0.00385966, 0.00561288, 0.00228524,

AttributeError: 'GMM' object has no attribute 'normalPDF'
```

In [31]:
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

# For grads
# # test mutlinormalPDF
sigma_grad = np.array([[[ 0.12015895,  0.61720311,  0.30017032],
        [-0.35224985, -1.1425182 , -0.34934272],
        [-0.20889423,  0.58662319,  0.83898341]],

       [[ 0.93110208,  0.28558733,  0.88514116],
        [-0.75439794,  1.25286816,  0.51292982],
        [-0.29809284,  0.48851815, -0.07557171]],

       [[ 1.13162939,  1.51981682,  2.18557541],
        [-1.39649634, -1.44411381, -0.50446586],
        [ 0.16003707,  0.87616892,  0.31563495]]])
my_multinormalpdf = GMM(data, 3).multinormalPDF(points, mu[0], sigma_grad[0])
expected_multinormal_pdf = np.array([8.70516304e-074, 8.62201632e-001, 5.36048920e+015, 2.99498046e+1
88,
       6.91708798e+083, 9.96882978e-062, 7.03348279e-025, 2.16083146e-059,
       1.87537738e-086, 1.84295981e+075, 1.11845126e+000, 5.17746613e-097])
print("Your multinormal pdf works within the expected range: ", np.allclose(expected_multinormal_pdf,
my_multinormalpdf))


# test ll-joint
sigma_now = sigma * 0.5
my_lljoint = GMM(points, 3)._ll_joint(pi, mu, sigma_now, True)
expected_lljoint = np.array([[ -8.48080757,  -7.51160532,  -2.81570712],
       [ -7.28033641,  -2.81570712,  -7.51160531],
       [ -5.99589887,  -4.74012597,  -8.87599209],
       [ -7.22441107, -10.91289393,  -7.01960107],
       [-11.20807212,  -7.65010086,  -5.98110204],
       [ -6.92419072,  -4.83482203,  -3.56951111],
       [ -7.72819284,  -3.06904217,  -6.26803946],
       [ -8.41643028,  -3.87795155,  -4.485805  ],
       [ -7.66746017,  -9.42125381,  -4.10677788],
       [ -9.4646439 ,  -6.20890536,  -8.07178468],
       [ -2.81570712,  -7.28033643,  -8.48080755],
       [ -9.28269723, -12.03115847,  -4.17935163]])
```

```
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([[3.42169503e-03, 9.01904364e-03, 9.87559261e-01],
       [1.12762023e-02, 9.79775837e-01, 8.94796041e-03],
       [2.18977456e-01, 7.68731442e-01, 1.22911017e-02],
       [4.43990237e-01, 1.11041589e-02, 5.44905604e-01],
       [4.49802848e-03, 1.57844511e-01, 8.37657460e-01],
       [2.65137773e-02, 2.14226353e-01, 7.59259870e-01],
       [9.02095401e-03, 9.52129225e-01, 3.88498209e-02],
       [6.87345697e-03, 6.43000762e-01, 3.50125781e-01],
       [2.75025136e-02, 4.76112393e-03, 9.67736363e-01],
       [3.22944111e-02, 8.37677122e-01, 1.30028467e-01],
       [9.85247145e-01, 1.13391711e-02, 3.41368409e-03],
       [6.03734929e-03, 3.86549176e-04, 9.93576102e-01]])


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([0.1479711 , 0.38249961, 0.46952929])
expected_mu = np.array([[-0.15519344,  1.22500376,  0.03548931],
       [-0.36778399, -0.18068954, -0.65203503],
       [-0.28448252, -0.09079301,  0.92618845]])
expected_sigma = np.array([[[ 0.67247982, -0.25027742,  0.0774841 ],
        [-0.25027742,  0.34077941,  0.04853111],
        [ 0.0774841 ,  0.04853111,  0.2987035 ]],

       [[ 0.49792869,  0.07842407, -0.09002534],
        [ 0.07842407,  0.1618932 , -0.10696588],
        [-0.09002534, -0.10696588,  0.20401203]],

       [[ 0.65130447,  0.0049166 , -0.39258756],
        [ 0.0049166 ,  0.22371688,  0.18769942],
        [-0.39258756,  0.18769942,  0.70840301]]])
print("Your M step works within the expected range: ", np.allclose(my_pi, expected_pi) and np.allclos
e(my_mu, expected_mu) and np.allclose(my_sigma, expected_sigma))
print(f"Pi is correct: {np.allclose(my_pi, expected_pi)}")
print(f"Mu is correct: {np.allclose(my_mu, expected_mu)}")
print(f"Sigma is correct: {np.allclose(my_sigma, expected_sigma)}")
```

```
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
Pi is correct: True
Mu is correct: True
Sigma is correct: True
```

3.3 Jackson Pollock and pixel clustering [10pts]

Jackson Pollock was an American artist during the 1930s and 40s. He was famous for his unique method of painting (as you will see) where he used to splash paint with a brush onto the canvas. This is now referred to as "drip painting" which has divided many art critics.

In this section, you will use your GMM algorithm implementation to do pixel clustering and compress the paintings. 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)

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 [36]:
```
################################
### DO NOT CHANGE THIS CELL ###
################################



# helper function for performing pixel clustering. You don't have to modify it
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 [37]:
```
# helper function for performing pixel clustering. You don't have to modify it
# pick 2 of the images in this list:
url1 = 'https://storage.googleapis.com/thehundreds/media/2017/06/convergence-1.jpg'
url2 = 'https://storage.googleapis.com/thehundreds/media/2017/06/lavender-mist.jpg'
url3 = 'https://storage.googleapis.com/thehundreds/media/2017/06/mural-1.jpg'

# example of loading image from url1
image = imageio.imread(imageio.core.urlopen(url1).read())

# 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 pro
duce a single image.
    You can simply examinate the answer based on your visual inspection (i.e. looking at the resultin
g images) or provide any metrics you prefer.

    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 are 5a dn 15.
        (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.
    """
    # cluster_idx => 256x256x3;
    # mean_values_of_the_clusters => n_clusters x 1
    # for i in range(256):
    # for j in range(256):
    # for k in range 3:
    # gmm_image_k[i,j,k] = cluster_idx[i,j,k]
    comparison = np.zeros([max_clusters - min_clusters, 1])
    for K in range(min_clusters, max_clusters+1, 5):
        gmm_image_k = cluster_pixels_gmm(image, K, full_matrix = True)
        try:
            np.isnan(np.min(gmm_image_k))
            plot_images([image, gmm_image_k], ['original', 'gmm='+str(K)])
            image_np = np.array (image)
            comparison[K] = np.linalg.norm(gmm_image_k - image_np)
        except:
```

```
            print("An exception occurred")
    return comparison
    #return gmm_image_k
```

In [38]:
```
# this is for you to implement
image1 = imageio.imread(imageio.core.urlopen(url1).read())
perform_compression(image1, 5, 25)
```

```
iter 9, loss: nan: 100%|██████████| 10/10 [00:06<00:00,  1.53it/s]
iter 9, loss: nan: 100%|██████████| 10/10 [00:15<00:00,  1.52s/it]
iter 9, loss: nan: 100%|██████████| 10/10 [00:22<00:00,  2.25s/it]
iter 9, loss: nan: 100%|██████████| 10/10 [00:29<00:00,  2.98s/it]
  0%|          | 0/10 [00:00<?, ?it/s]

An exception occurred

iter 9, loss: nan: 100%|██████████| 10/10 [00:40<00:00,  4.01s/it]

An exception occurred
```

Out[38]:

```
array([[ 0.],
       [ 0.],
       [ 0.],
       [ 0.],
       [ 0.],
       [nan],
       [ 0.],
       [ 0.],
       [ 0.],
       [ 0.],
       [nan],
       [ 0.],
       [ 0.],
       [ 0.],
       [ 0.],
       [nan],
       [ 0.],
       [ 0.],
       [ 0.],
       [ 0.]])
```

original

gmm=5

original

gmm=10

original

gmm=15

original

gmm=20

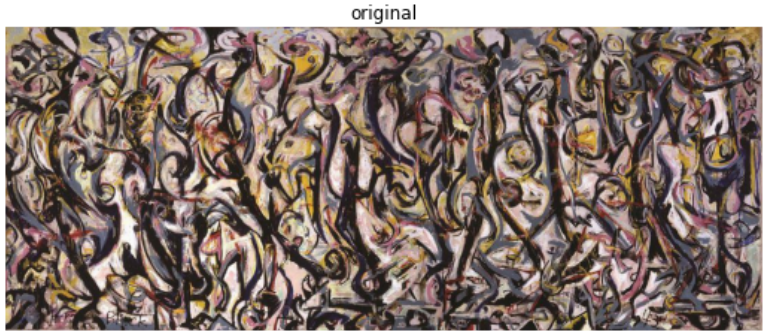
original

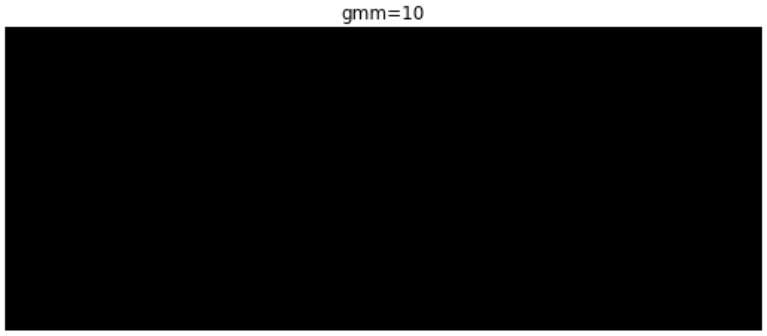
gmm=25

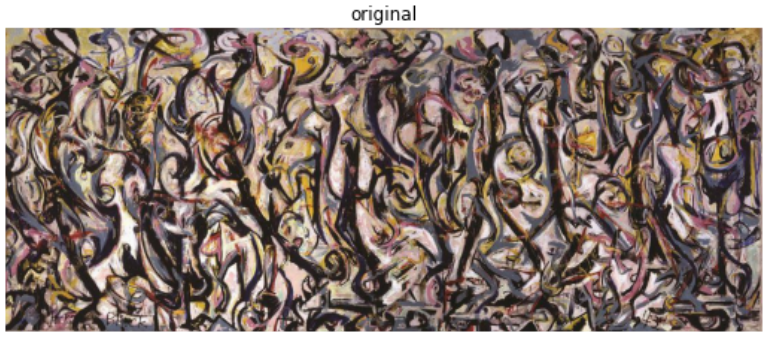
In [42]:
```
# this is for you to implement
image3 = imageio.imread(imageio.core.urlopen(url3).read())
perform_compression(image3, 10, 15)
```

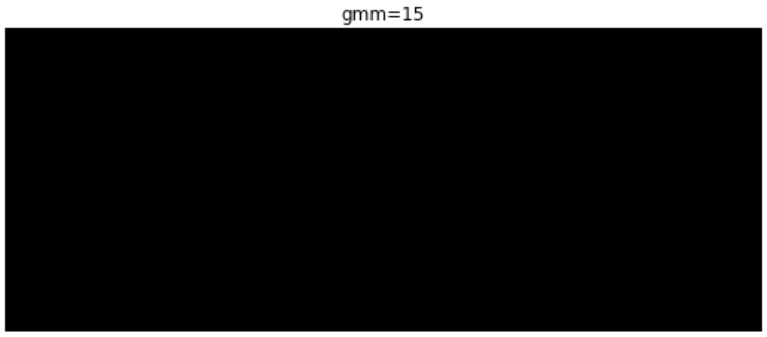
```
iter 9, loss: nan: 100%|██████████| 10/10 [00:09<00:00,  1.08it/s]
  0%|          | 0/10 [00:00<?, ?it/s]

An exception occurred

iter 9, loss: nan: 100%|██████████| 10/10 [00:14<00:00,  1.43s/it]

An exception occurred
```

Out[42]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```









SOLUTION:

In this question, I was getting black images. I tried all possible solutions posted in Edstem, (numebers 283 and 312) regading eliminating Nans, 0s, etc. However, none of them work for me. In fact, it depends how you impelmeted those function and how to error is aggregated. Then, depending upon the way we implement we may got different solution. I tried 2-3 ways of implemetation and then rectification. However, finally none of them work for me. Here, I wrote what would be the expected outocme. Compression is a technique that cause we reduce the size of data such that we need less storage and less movement of data. It is also reduces compuation time, power and energy, and required bandwidth. It has two types generally: lossless and lossy. In lossles, we dont reduce the quality of the image. we utilize the intrinsic nature of the data such as leading zeros, ones, data similarity to compress the image. The most famous algorithms are Run Length encoding Huffman Coding and Arithmetic Coding. In the lossy, the data size is reduced in order to compress the image even more. The most famous approached are Discrete Cosine Transform, Fractal compression, Transform Coding. In this section, to have a lossy compression, we can replace each pixel with its centroid points (cluster_id) comining from gmm clusterig (gmm_image_k). For choosing best number of cluster, different method such as Silhouette score, elbow methods can be used. In the case of perfroming lossy compression, by increasing number of iterations, the images would be less blury. Chosing the right number of cluster is highly dependent to the images that we have. we should explore design space and find the differences as I did in my code to see where it ges its minimum. On the other hand we can change number of min clusters, max clusters, and look at the images and observe which one gives best results (i.e., which one is more close to original image quality as we are doing lossy compression)

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

(Preamble: This part of the assignment was designed to expose you to interesting topics we did not cover in class, while allowing you to do minimal work by reusing most of your previous implementations with some modifications.)

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

You are a consultant for Nike, a company known for its many athlete endorsements and shoes. The project you are working on is a new shoe designed to help basketball players increase their vertical jump when dunking while providing increased shock absorption using a combination of innovative rubbers and other such materials. To experiment to see the effectiveness of the shoe, Nike brings in D1 athletes and run various drills that place an important on vertical jump. Athletes wear different varients of the shoe. Then, these results are recorded on a Windows 8 Computer. The data includes 3 features corresponding to the makeup of the shoe and 1 column for the label. 1 means the shoe increased the perfomance of the athlete and 0 means the shoe had no impact on the performance.

However, like a classic Windows 8 Computer, you randomly see the blue screen of death (https://en.wikipedia.org/wiki/Blue_screen_of_death). After you restart the computer, 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 the Nike in cleaning their data and implementing a semi-supervised learning framework to help them create a general classifier.

You are given four files for this task:

- Labeled_athlete_complete.txt: containing the complete material characterization data and corresponding labels (increased vertical = 1 and no change = 0);
- Labeled_athlete_incomplete.txt: containing partial material characterization data and corresponding labels (increased vertical = 1 and no change = 0);
- Unlabeled_athlete.txt: containing only complete material characterization results;
- Independent_athlete.txt: a labeled dataset the students obtained from a previous experiment, which you can use to test your model after training.

4.1 KNN [10pts]

The first 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 [ ]:
```
################################
### 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.

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

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]])

clean_data = CleanData()(incomplete_data, complete_data, 2)
print("*** Expected Answer - k = 2 ***")
print("""==complete data==
[[ 1.  2.  3.  1.]
 [ 7.  8.  9.  0.]
 [16. 17. 18.  1.]
 [22. 23. 24.  0.]]
==incomplete data==
[[ 1. nan  3.  1.]
 [ 7. nan  9.  0.]
 [nan 17. 18.  1.]
 [nan 23. 24.  0.]]
==clean_data==
[[ 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. ]]""")

print("\n*** My Answer - k = 2***")
print(clean_data)
```

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 1—5.2 and 6.1. Write a brief summary of three interesting highlights of the paper (50-word maximum).

SOLUTION:

Generally having labeled data is expensive; however, unlabeled data is almost free. On the other hand, the accuracy using unlabeled data usally is not high. To overcome this problem, for the text classification, this paper adds a few labeled training documents with a pool of unlabeled documents and modifies EM algorithm by adding a naive Bayes classifier. At first, the algorithm trains a classifier using labeled data and then probabilistically labels the unlabeled data. To enhance the accuracy of EM, the authour perform two things: 1- employing several mixture components per class 2- adding weightening factor to adjust the contribution of unlabeled data

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 method.)

In [ ]:
```
################################
### 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 achieve the full 5 points you must get these scores:

- semi_supervised_score > .81
- GNB_onlycomplete_score > .70
- GNB_cleandata_score > .72

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

from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Load and clean data for the next section
labeled_complete = np.loadtxt('./data/datasets/labeled_athlete_complete.txt', delimiter=',')
labeled_incomplete = np.loadtxt('./data/datasets/labeled_athlete_incomplete.txt', delimiter=',')
clean_data = CleanData()(labeled_incomplete, labeled_complete, 2)
# load unlabeled set
unlabeled = np.loadtxt('./data/datasets/unlabeled_athlete.txt', delimiter=',')
# append unlabeled flag
unlabeled_flag = -1*np.ones((unlabeled.shape[0],1))
unlabeled = np.concatenate((unlabeled, unlabeled_flag), 1)

# ================================================================================
# SEMI SUPERVISED

# format training data
points = np.concatenate((clean_data, unlabeled),0)

# train model
(pi, mu, sigma) = SemiSupervised()(points, 2)


# ================================================================================
# SUPERVISED WITH CLEAN DATA (SKLEARN)

clean_clf = GaussianNB()
clean_clf.fit(clean_data[:,:3], clean_data[:,3])

# ================================================================================
# SUPERVISED WITH ONLY THE COMPLETE DATA (SKLEARN)

complete_clf = GaussianNB()
complete_clf.fit(labeled_complete[:,:3], labeled_complete[:,3])

# ================================================================================
# COMPARISON
```

```
# load test data
independent = np.loadtxt('./data/datasets/independent_athlete.txt', delimiter=',')

# classify test data
classification = SemiSupervised()._E_step(independent[:,:3], pi, mu, sigma)
classification = np.argmax(classification,axis=1)

semi_supervised_score = accuracy_score(classification, independent[:,3])
clean_supervised_score = clean_clf.score(independent[:,:3],independent[:,3])
complete_supervised_score = complete_clf.score(independent[:,:3],independent[:,3])

# ================================================================================

print("""===COMPARISON===""")
print("""SemiSupervised Accuracy:""", semi_supervised_score)
print("""Supervised with clean data: GNB Accuracy:""", clean_supervised_score)
print("""Supervised with only complete data: GNB Accuracy:""", complete_supervised_score)
```