Spring 2024 CS4641/CS7641 Homework 2

Instructor: Dr. Mahdi Roozbahani

Deadline: Friday, March 8th, 11:59 pm EST

- 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

- Grads will find three assignments and Undergrads will find four assignments on Gradescope that correspond to HW2: "Assignment 2 Programming", "Assignment 2 - Non-programming", "Assignment 2 Programming - Bonus for all", and "Assignment 2 Programming - Bonus for Undergrad" (Undergrad Only).

- 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 need to submit to Gradescope a PDF copy of your Jupyter Notebook with the cells ran. See this EdStem Post for multiple ways on to convert your .ipynb into a .pdf file.** 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. The pages in the PDF should be of size 8.5" x 11", otherwise there may be a deduction in points for extra long sheets.**

Using the local tests

- For some of the programming questions we have included a local test using a small toy dataset to aid in debugging. The local tests are all stored in localtests.py
- There are no points associated with passing or failing the local tests, you must still pass the autograder to get points.
- **It is possible to fail the local test and pass the autograder** since the autograder has a certain allowed error tolerance while the local test allowed error may be smaller. Likewise, passing the local tests does not guarantee passing the autograder.
- **You do not need to pass both local and autograder tests to get points, passing the Gradescope autograder is sufficient for credit.**
- It might be helpful to comment out the tests for functions that have not been completed yet.
- It is recommended to test the functions as it gets completed instead of completing the whole class and then testing. This may help in isolating errors. Do not solely rely on the local tests, continue to test on the autograder regularly as well.

Deliverables and Points Distribution

Q1: KMeans Clustering & DBScan [50pts total: 37pts + 6% Bonus for Undergrad]

Deliverables: kmeans.py and dbscan.py

- **pairwise_dist** [5 pts] - *programming*
- **KMeans Implementation** [30pts] - *programming*
  - init_centers [2pts]
  - kmpp_init [3pts G / 1.5% UG] **BONUS FOR UNDERGRAD**
  - update_assignment [5pts]
  - update_centers [5pts]
  - get_loss function [5pts]
  - train [10pts]
- **Rand Statistic** [5 pts] - *programming*
- **DBScan** [10 pts G / 4.5% UG] - *programming* **BONUS FOR UNDERGRAD**
  - regionQuery [2pts G / 0.9% UG]
  - expandClusters [4pts G / 1.8% UG]
  - fit [4pts G / 1.8% UG]

Q2: EM Algorithm [15pts total + 1% Bonus for All]

Deliverables: Written Report

- **2.1 Performing EM Update** [15 pts] - *non-programming*
    - 2.1.1 [3pts] - *non-programming*
    - 2.1.2 [3pts] - *non-programming*
    - 2.1.3 [9pts] - *non-programming*
- **2.2 Gradient Descent and EM algorithm** [1%] - *non-programming* **BONUS FOR ALL**

Q3: GMM implementation [60pts total + 1% 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 [1%] *non-programming* **BONUS FOR ALL**
- 3.5 Generate samples from a Gaussian Mixture [5pts] *non-programming*

Q4: Cleaning Super Duper Messy data with semi-supervised learning [8% Bonus for All]

Deliverables: semisupervised.py and Written Report

- 4.1: KNN [2.8%] - *programming*
    - 4.1.a. complete*, incomplete*, unlabeled_ [0.7%]
    - 4.1.b. CleanData __call__ [1.4%]

  - 4.1.c. MeanCleanData [0.7%]
- 4.2: Getting acquainted with semi-supervised learning approaches [3.5%] - *programming & non-programming*
  - 4.2.a. Write highlight summary [1.2%] - *non-programming*
  - 4.2.b. Implement EM algorithm [2.3%] - *programming*
- 4.3: Demonstrating the performance of the algorithm [1.1%] - *programming*
  - accuracy_semi_supervised [0.55%]
  - accuracy_GNB [0.55%]
- 4.4: Interpretation of Results [0.6%] - *non-programming*

Note: It is highly recommended that you do Q4 (if not for the HW then before the project) as it teaches you imperfect data handling and a good understanding of how the models you have learnt can be used together for better results.

Q5: Evaluating Data Representation in K-Means Clustering [4pts]

Deliverables: Written Report

0 Set up

This notebook is tested under python 3.11.**, and the corresponding packages can be downloaded from miniconda. You may also want to get yourself familiar with several packages:

- jupyter notebook
- numpy
- matplotlib

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 [ ]:

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

from __future__ import absolute_import, division, print_function
```

```
%matplotlib inline

import sys

import localtests as localtests
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
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
```

1. Painting on Canvas - KMeans Clustering & DBScan [50pts total: 37pts + 6% Bonus for Undergrad]

Rob Boss, a perpetual Ph.D student at Georgia Tech University, is teaching his dissertation robot, Pablo, how to paint in preparation for the Clough Art Crawl. Unfortunately, Rob is on a shoestring PhD budget and can only afford to buy Pablo a limited variety of colors to practice with. Luckily Rob has taken Machine Learning and remembers that KMeans clustering can be used to compress images down to a few colors. Here you will help Rob paint this happy lovely dog by implementing KMeans.

(Artwork generated by Fotor)

No description has been provided for this image

KMeans is trying to solve the following optimization problem:

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

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

DO NOT USE LOOPS in your implementation, **using for-loops or while-loops will result in 0 credit for this portion.** Use array broadcasting instead.

Hint: Make sure you use the X^2 + Y^2 – 2XY shortcut calculation. This should eliminate the need for creating a third dimension (which would otherwise timeout).

Hint: To calculate X^2 and Y^2 you can refer to the sum of squares function from assignment 1. For detailed explanation of pairwise distance function check this document - https://static.us.edusercontent.com/files/WLtSuk4PzW8e8M6VTsDq0BdJ

We have provided some unit tests in localtests.py for you to check your implementation. See Using the Local Tests for more details.

In [ ]:

```
localtests.KMeansTests().test_pairwise_dist()
# localtests.KMeansTests().test_pairwise_speed()
```

1.2 KMeans Implementation [30pts: 27pts + 1.5% Bonus for Undergrad]

In this section, you are asked to implement several methods in **kmeans.py**

Initialization: [5pts: 2pts + 3pts Bonus for Undergrad]

The Kmeans algorithm is sensitive to how the centers are initialized. The naive approach is to randomly initialize the centers. However, a bad initialization can increase the time required for convergence or may even converge to a non-optimal solution.

- **init_centers** [2pts]: Here you will initialize the centers randomly **(Required for all)**

Hint: Please initialize centers by randomly sampling points from the dataset in case the autograder fails.

- **kmpp_init** [3pts Bonus for Undergrad]: Here you will use the intuition that points further away from each other will probably be better initial centers by implementing a version of KMeans++ **(Bonus for Undergrad, required for Grads)**

Hint: We need to initialize the centers without repetition.

KMeans++

The algorithm for KMPP that you will implement can be described as follows:

1. Sample 1% of the points from the dataset, uniformly at random (UAR) and without replacement. This sample will be the dataset the remainder of the algorithm uses to minimize initialization overhead.

2. From the above sample, select only one random point to be the first cluster center.
3. For each point in the sampled dataset, find the nearest cluster center and record the squared distance to get there.
4. Examine all the squared distances and take the point with the maximum squared distance as a new cluster center. In other words, we will choose the next center based on the maximum of the minimum calculated distance instead of sampling randomly like in step 2. You may break ties arbitrarily.
5. Repeat 3-4 until all k-centers have been assigned. You may use a loop over K to keep track of the data in each cluster.

Updating Cluster Assignments: [5pts]

After you've chosen your centers, you will need to update the membership of each point based on the closest center. You will implement this in **update_assignment**. See docstring for more details.

Updating Centers Assignments: [5pts]

Since cluster memberships may have changed, you will need to update the cluster centers. You will implement this in **update_centers**. See docstring for more details.

Loss & Convergence [5pts]

We will consider KMeans to be converged when the change in loss drops below a threshold value. The loss will be defined as the sum of the squared distances between each point and its respective center.

Hint: You may use a loop over K to keep track of the data in each cluster. Avoid looping over N individual datapoints.

Train the model [10pts]

In the **train** method you will use all of the previously implemented steps to train your KMeans algorithm until convergence. Since the centers have already been initialized in the **init** function the general steps for the `train` method is as follows:

1. Update the cluster assignment for each point
2. Update the cluster centers based on the new assignments from Step 1
3. Check to make sure there is no mean without a cluster, i.e. no cluster center without any points assigned to it.
   - In the event of a cluster with no points assigned, pick a random point in the dataset to be the new center and update your cluster assignment accordingly.
4. Calculate the loss and check if the model has converged to break the loop early.
   - The convergence criteria is measured by whether the percentage difference in loss compared to the previous iteration is less than the given relative tolerance threshold (self.rel_tol).

5. Iterate through steps 1 to 4 max_iters times. **Make sure to avoid infinite looping.**

We have provided the following local tests to help you check your implementation. Provided unit-tests are meant as a guide and are not intended to be comprehensive. See Using the Local Tests for more details.

In [ ]:

```
localtests.KMeansTests().test_init()
localtests.KMeansTests().test_update_centers()
localtests.KMeansTests().test_kmeans_loss()
```

1.3 Visualize KMeans [0pts]

We will now take a look at how image quality is impacted by the number of clusters

In [ ]:

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

# Note that because of a different file structure, students' paths will be c
from utilities import *

image_values = image_to_matrix("./data/images/dog.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, image_values, r, c, ch).reshape(r, c, ch)
image_5 = update_image_values(5, image_values, r, c, ch).reshape(r, c, ch)
image_10 = update_image_values(10, image_values, r, c, ch).reshape(r, c, ch)
image_20 = update_image_values(20, image_values, r, c, ch).reshape(r, c, ch)

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

1.4 Rand Statistic [5pts]

In this section, you will create a function to assess the quality of a clustering algorithm using the Rand Statistic. The Rand Statistic quantifies the goodness or quality of a clustering algorithm when compared to a ground truth.

As discussed in class, the Rand Index is calculated as follows:

$$Rand = \frac{TP + TN}{TP + FP + FN + TN}$$

TP (True Positive) represents the number of pairs of data points that are correctly clustered together in both the algorithm's result and the ground truth.

TN (True Negative) is the count of pairs of data points that are correctly placed in separate clusters in both the algorithm's result and the ground truth.

FP (False Positive) counts the pairs of data points that are incorrectly clustered together in the algorithm's result but correctly separated in the ground truth.

FN (False Negative) is the number of pairs of data points that are incorrectly separated in the algorithm's result but correctly clustered together in the ground truth.

We have provided the following local tests to help you check your implementation. Provided unit-tests are meant as a guide and are not intended to be comprehensive. See Using the Local Tests for more details.

Refer to the class notes for more information on the Rand Statistic

In [ ]:

```
localtests.KMeansTests().test_rand_statistic()
```

1.5 Limitation of K-Means

You've now done the best you can selecting the perfect starting points and the right number of clusters. However, one of the limitations of K-Means Clustering is that it depends 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 that crosses the circles. You can visualize this limitation in the cell below.

In [ ]:

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

# visualize limitation of kmeans
from kmeans import *
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
```

```
kmeans = KMeans(X1, 2)
centers1, cluster_idx1, loss1 = kmeans.train()
visualise(X1, cluster_idx1, 2)
kmeans = KMeans(X2, 2)
centers2, cluster_idx2, loss2 = kmeans.train()
visualise(X2, cluster_idx2, 2)
```

1.6 DBSCAN [10pts Grad / 4.5% 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".

Refer to the class slides for the DBSCAN pseudocode to complete fit(), expandCluster(), and regionQuery() in dbscan.py.

HINTS:

- You might find it easier to implement expandCluster() before attempting to implement fit().
- regionQuery() could be used in you implementation of expandCluster()

The following unittests will help get you started, but is in no way comprehensive. You are encouraged to extend and create your own test cases. See Using the Local Tests for more details.

In [ ]:

```
localtests.DBScanTests().test_region_query()
localtests.DBScanTests().test_expand_cluster()
```

Then, test your fitting 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 [ ]:

```
###############################
### 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 implementa
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 0s ar
visualise(X1, cluster_idx)
```

2. EM algorithm [15pts + 1% Bonus for All]

2.1 Performing EM Update [15 pts]

ANSWERS 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:

$$\mathbf{z} \sim Bernoulli(\alpha) = \begin{cases} \alpha & \text{if } z = 0 \\ 1 - \alpha & \text{if } z = 1 \end{cases}$$

$$\mathbf{x_n} \mid \mathbf{z = 0} \sim \mathcal{N}(3\upsilon, 4\omega^2)$$

$$\mathbf{x_n} \mid \mathbf{z = 1} \sim \mathcal{N}(2\upsilon, 9\omega^2)$$

For a dataset of N datapoints, find the following:

2.1.1. Write the marginal probability of x, i.e. $p(x)$ [3pts]
-- Express your answers in terms of $\mathcal{N}(x|3\upsilon, 4\omega^2)$ and $\mathcal{N}(x|2\upsilon, 9\omega^2)$ may be simpler
-- HINT: For this question suppose we have a Gaussian Distribution $\mathcal{N}(3\upsilon, 7\omega^2)$, it means $\mathcal{N}(\mu = 3\upsilon, \sigma^2 = 7\omega^2)$
-- HINT: Start with the Sum Rule

$$\begin{aligned} p(x) &= \sum_z p(x|z)P(z) \\ &= p(x|z=0)P(z=0) + p(x|z=1)P(z=1) \\ &= \boxed{\alpha \cdot \mathcal{N}(x|3\upsilon, 4\omega^2) + (1-\alpha) \cdot \mathcal{N}(x|2\upsilon, 9\omega^2)} \end{aligned}$$

2.1.2. E-Step: Compute the posterior probabilities, i.e, $p(z_0|x), p(z_1|x)$ [3pts]
-- Express your answers in terms of $\mathcal{N}(3\upsilon, 4\omega^2)$ and $\mathcal{N}(2\upsilon, 9\omega^2)$
-- HINT: Try to apply Beyes Rule

$p(z|x) = \frac{p(x|z)P(z)}{p(x)}$

$$\boxed{p(z=0|x) = \frac{\alpha \cdot \mathcal{N}(x|3\upsilon, 4\omega^2)}{\alpha \cdot \mathcal{N}(x|3\upsilon, 4\omega^2) + (1-\alpha) \cdot \mathcal{N}(x|2\upsilon, 9\omega^2)}}$$

$$\boxed{p(z=1|x) = \frac{(1-\alpha)\cdot\mathcal{N}(x|2\upsilon, 9\omega^2)}{\alpha\cdot\mathcal{N}(x|3\upsilon, 4\omega^2) + (1-\alpha)\cdot\mathcal{N}(x|2\upsilon, 9\omega^2)}}$$

2.1.3. M-Step: Compute the updated value of ω^2. (You can keep μ fixed when you calculate the derivative.) [9pts]

-- Note that ω^2 is a shared variable between the two distributions, your final answer should be one equation including both Gaussian distributions

-- Express your answers in terms of τ, x, and υ (you will need to expand $\mathcal{N}(3\upsilon, 4\omega^2)$ and $\mathcal{N}(2\upsilon, 9\omega^2)$ into its PDF form)

-- HINT: Start from the below equation, note that θ is shorthand for various variables, and take the derivative w.r.t. ω^2

$$\ell(\theta|x) = \sum^{N}\sum^{Z} p(z_k|x_n, \theta_{old}) ln\left[p(x_n, z_k|\theta)\right]$$
$$\ell(\upsilon, \omega^2, \alpha \mid x) = \sum^{N}\sum^{Z} p(z_k \mid x_n, \theta_{old}) \ln\left[p(x_n, z_k \mid \mu_k, \sigma_k^2, \alpha)\right]$$
$$= \sum^{N}\sum^{Z} p(z_k \mid x_n, \theta_{old}) \ln\left[p(z_k \mid \alpha) p(x_n \mid z_k, \upsilon, \omega^2)\right]$$

Recall that $p(x_n \mid z_k, \upsilon, \omega^2) \to \mathcal{N}(x_n \mid \mu_k, \sigma_k)$ has been defined at the beginning of the problem.

You can refer to this lecture to gain an understanding of the EM Algorithm. For your convenience, I have included the link below:

https://mahdi-roozbahani.github.io/CS46417641-spring2024/course/09-gaussian-mixture.pdf

$$\frac{d\ell}{d\omega^2} = \frac{-36\omega^2(\tau_0+\tau_1)+81\tau_0\upsilon^2-54\tau_0\upsilon x_n+9\tau_0 x_n^2+16\tau_1\upsilon^2-16\tau_1\upsilon x_n+4\tau_1 x_n^2}{72{\omega^2}^2}$$

$$0 = \frac{-36\omega^2(\tau_0+\tau_1)+81\tau_0\upsilon^2-54\tau_0\upsilon x_n+9\tau_0 x_n^2+16\tau_1\upsilon^2-16\tau_1\upsilon x_n+4\tau_1 x_n^2}{72{\omega^2}^2}$$

$$\boxed{\omega^2 = \frac{81\tau_0\upsilon^2 - 54\tau_0\upsilon x_n + 9\tau_0 x_n^2 + 16\tau_1\upsilon^2 - 16\tau_1\upsilon x_n + 4\tau_1 x_n^2}{36(\tau_0+\tau_1)}}$$

2.2 Gradient Ascent and EM algorithm [1% Bonus for All]

2.2. Compared to the Gradient Ascent algorithm, what's the computational advantage of using EM algorithm for the problem presented in 2.1? Please provide your own analysis and it can be qualitatively. [5pts]

-- HINT: Think about the difference in updating parameters during each iteration. i.e. How many parameters need to be updated in gradient descent? What we did in for each iteration in EM algorithm to simplify it?

The EM algorithm, with its clear demarcation between the E-step (expectation) and M-step (maximization), offers a structured approach to parameter updates that can be more computationally efficient than gradient ascent, especially in the context of mixture models. In gradient ascent, every parameter update requires recalculating gradients for the entire model, which can be computationally expensive. EM, on the other hand, simplifies the process by decoupling the estimation of latent variables (E-step) from the optimization of parameters (M-step), allowing for more targeted and often faster convergence.

3. GMM implementation [65pts total: 60pts + 1% Bonus for All]

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 $X = [x_1, x_2, \ldots, x_N]^T$, where $x_i \in \mathbb{R}^D$. Let π be a K-dimensional probability density function 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:

$$\ln p(x_1, \ldots, x_N | \pi, \mu, \Sigma) = \sum_{i=1}^{N} \ln \Big(\sum_{k=1}^{K} \pi(k) \mathcal{N}(x_i | \mu_k, \Sigma_k) \Big) \tag{2}$$

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:

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

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:

$$\mu_k^{new} = \frac{\sum_{n=1}^N \tau(z_k)x_n}{N_k} \tag{3}$$

$$\Sigma_k^{new} = \frac{1}{N_k}\sum_{n=1}^N \tau(z_k)^T(x_n-\mu_k^{new})^T(x_n-\mu_k^{new}) \tag{4}$$

$$\pi_k^{new} = \frac{N_k}{N} \tag{5}$$

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 X is (D, N) but **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 datapoints in our homework problem. If you have to loop over D or K, that is 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 hava N pixels and $D = 3$ features which corresponds 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. 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) could enhance the functionality of your code and help you debugging. 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,)

- 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 [15pts]

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) that reperesents the unnormalized scores, that are passed to the softmax( ) or logsumexp( ) function. Remember the goal of helper functions is to facilitate our calculation so **DO NOT USE FOR LOOP OVER N**.

3.1.1. softmax [5pts]

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

Notes:

- $logit$ here refers to the unnormalized scores that are passed in as a parameter to the softmax function. The softmax operation normalizes these scores, resulting in them having values between 0 and 1. This allows us to interpret the normalized scores as a probability distribution over the classes.
- It is possible that $logit_{i,j}$ is very large, making $\exp(\cdot)$ of it to explode. To make sure it is numerically stable, for each row of $logits$ subtract the maximum of that row.
    - By property of Softmax equation, subtracting a constant value does not change the output. Refer to Mathematical properties
    - For an intuitive understanding on why this helps us, consider plotting e^{-x} and e^{x} on a graphing calculator when $x \geq 0$

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 [3pts Programming + 2pts Written Questions]

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 face similar condition as in the softmax function. In this case, add the maximum for each row of $logit$ 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 [2pts]:

1. Why should we add the maximum for each row of $logit$ to **logsumexp()** function? Show your reason by calculating and observing the relationship between F and s_1.
   -- Use a simple input like $logit \in \mathbb{R}^{1\times 3}$ and work through a mathematical example.
   -- Let N=1, D=3, $logit = \{logit_{11}, logit_{12}, logit_{13}\}$ and $max = logit_{13}$ is the maximum for this row. F is the an array that substract the maximum for each row of $logits$.
   -- Start by subtracting the max of the row from each element in
   $s_1 = \log\left(\sum_{j=1}^{D} \exp(logit_{1,j})\right)$

To ensure numerical stability in the `logsumexp()` function, we use a technique where we subtract then add back the maximum value of each row of $logit$. This approach prevents overflow or underflow in exponential calculations. Mathematically, for $logit = logit_{11}, logit_{12}, logit_{13}$ with $max = logit_{13}$ being the maximum, we first calculate $F = logit_{1j} - max$ for each j, which ensures the highest value for exponentiation is 0. The calculation of s_1 becomes $s_1 = \log\left(\sum_{j=1}^{D}\exp(F_j)\right) + max$, where $F = logit_{11} - max, logit_{12} - max, 0$. This technique maintains accuracy and computational safety by keeping exponentiated values within a manageable range, ensuring that the subtraction and addition of max does not alter the mathematical outcome but aids in stabilizing the computation.

3.1.3. Multivariate Gaussian PDF [5pts]

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 in Python. Please only add `SIGMA_CONST` to all elements when `sigma_i` is not invertible.

- 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.

- Please **DO NOT** use `self.D` in your implementation of `multinormalPDF()` .

3.2 GMM Implementation [30pts]

Things to do in this problem:

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

Examples of how you can initialize the parameters.

1. `create_pi()` : Set the prior probability π the same for each class.
2. `create_mu()` : Initialize μ by randomly selecting K numbers of observations as the initial mean vectors. You can use int(np.random.uniform()) to get the row index number of the datapoints randomly.
3. `create_sigma()` : 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.
4. You are expected to call these methods in the `_init_components()` method
5. 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() [10pts]

The log-likelihood function is given by:

$$\ell(\theta) = \sum_{i=1}^{N} \ln \Big(\sum_{k=1}^{K} \pi(k) \mathcal{N}(x_i | \mu_k, \Sigma_k) \Big) \tag{6}$$

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

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

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. $\text{log-likelihood}[i, k] = \log(\pi_k + \text{1e-32}) + \log(\mathcal{N}(x_i | \mu_k, \Sigma_k) + 1e\text{-}32)$. If you pass the local test cases but fail the autograder, make sure you sum a small value like the example we given.
- 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 [5pts + 10pts]

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. See Using the Local Tests for more details.

In [ ]:

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

from gmm import GMM

gmm_tester = localtests.GMMTests()
```

In [ ]:

```
gmm_tester.test_helper_functions()
gmm_tester.test_init_components()
```

In [ ]:

```
gmm_tester.test_undergrad()
```

In [ ]:

```
gmm_tester.test_grad()
```

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 we will be looking for the resulting images that the code produces in the report. Replicating similar outputs to the Expected Outputs file would be sufficient for full credit.

In [ ]:

```
###############################
### 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_char

    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 [ ]:

```
%%time
### Above lines added only in solution ###

import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import imageio.v2 as imageio


# the direction of two images. Both of them are from ImageNet
img1_dir = "./data/images/gmm-example6.jpg"
img2_dir = "./data/images/gmm-example8.jpg"


# this is for you to implement
def perform_compression(image, min_clusters=10, max_clusters=20):
    """
    Using the helper function above to find the optimal number of clusters t
    You can simply examine the answer based on your visual inspection (i.e. 

    Args:
        image: input image of shape(H, W, 3)
        min_clusters, max_clusters: the minimum and maximum number of cluste
        (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.

    UNDERGRADS HINT: set full_matrix = False when using the helper functions
    """
    image = np.array(image, dtype=np.float64) / 255

    # Flatten the image
    pixels = image.reshape((-1, 3))

    best_clusters = None
    best_compressed_image = None
    for n_clusters in range(min_clusters, max_clusters + 1):
        kmeans = KMeans(n_clusters=n_clusters, n_init=10, random_state=0).fi
        compressed_image = kmeans.cluster_centers_[kmeans.labels_]
        compressed_image = compressed_image.reshape(image.shape)

        best_clusters = n_clusters
        best_compressed_image = compressed_image

    fig, ax = plt.subplots(1, 2, figsize=(12, 6))
    ax[0].imshow(image)
    ax[0].set_title('Original Image')
    ax[0].axis('off')

    ax[1].imshow(best_compressed_image)
    ax[1].set_title(f'Compressed Image\n{best_clusters} Clusters')
    ax[1].axis('off')

    plt.show()


image1 = imageio.imread(img1_dir)
perform_compression(image1, 10, 20)

image2 = imageio.imread(img2_dir)
perform_compression(image2, 10, 15)
```

3.4 Compare full covariance matrix with diagonal covariance matrix [1% Bonus for All]

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(), _M_step(), _E_step() function. **You will earn full credit only if you implement all functions AND provide an explanation.**

In [ ]:

```
################################
### 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 covaria
    """
    # 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 [ ]:

```
compare_matrix(image2, 20)
```

3.5 Generate samples from a Gaussian Mixture [5pts]

In this question, you will be fitting your GMM implementation on a 2D Gaussian Mixture to estimate the parameters of the distributions that make up the mixture, and then using these estimated parameters to generate samples.

In [ ]:

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

data = np.load("./data/gaussian_clusters.npy")
print(data.shape)

plt.plot(data[:, 0], data[:, 1], "x")
plt.axis("equal")
plt.title("2-D Gaussian Mixture")
plt.show()
```

Now, you need to estimate the parameters of the Gaussian Mixture, and then use these estimated parameters to generate 1000 samples from the Gaussian Mixture. Plot the sampled datapoints. **You should notice that it resembles the original Gaussian Mixture.**

Steps

- First, to estimate the parameters of the Gaussian Mixture, you'll need to fit your GMM implementation to the dataset. You need to specify K=3 to represent 3 gaussians in our model, and run the EM algorithm. You'll have to choose the value for max_iters. If at the end of this section, your plot of the sampled datapoints doesn't look like the original distribution, you may need to increase max_iters to fit the GMM model better, and obtain better estimates of the parameters.
- Once you heve the estimated parameters, we'll need to sample 1000 datapoints from the Gaussian Mixture. You will be using a technique called Rejection Sampling discussed below. Here are some external sources that may help: https://cosmiccoding.com.au/tutorials/rejection_sampling, https://towardsdatascience.com/rejection-sampling-with-python-d7a30cfc327b

    - We will be taking the approach from the first link, but extending it into the 2D space.
- The formula for the density function is $f(x_i) = \sum_{k=1}^{K} \pi(k)\mathcal{N}(x_i|\mu_k, \Sigma_k)$

Generating vs Sampling To generate points directly from a given distribution is done via Inverse transform sampling. In inverse transform sampling, we require taking the inverse of cummulative distribution function for our gaussian mixture model. This operation in general can be expensive unless there is some known formula for inverting the CDF. It is also not always possible to take the inverse of the CDF of a gaussian mixture model. For these reasons we will implement a sampling method instead. This sampling method will give us points matching the gmm without the computation and mathematical concerns of generation.

Rejection Sampling

Conventionally we think of Gaussian Mixture Models as a form of soft clustering, but you can also think of them as an algorithm for estimating density of data points with gaussians. Thus we can take an arbitary data point and using the gaussian mixture model as an estimation for the density at a given location. From here we want the points that we sample to be proportional to the density at a given location.

We go about this by, choosing an arbitary point (x,y). Then we use the density formula function $f(x_i) = \sum_{k=1}^{K} \pi(k)\mathcal{N}(x_i|\mu_k, \Sigma_k)$ to find out what the density of points is at (x,y). Now that we have the density, we can draw a random number between 0 and the maximum density to determine if we will keep or discard (x,y). If the random number drawn is less than the density, then (x,y) is our sample, otherwise we discard (x,y) and repeat. This method ensure that the samples we generate are proportional to the density predicted by our GMM at any given area.

In [ ]:

```
from scipy.stats import multivariate_normal


def density_function(x, pi, mu, sigma):
    """
    Calculate the density of a given point x using the GMM parameters.

    Args:
    - x: A 2D point as a NumPy array.
    - pi: Mixture weights for each Gaussian component.
    - mu: Means of the Gaussian components.
    - sigma: Covariance matrices of the Gaussian components.

    Returns:
    - The density at point x.
    """
    K = len(pi)
    density = 0
    for k in range(K):
        component_density = pi[k] * multivariate_normal.pdf(x, mean=mu[k], c
        density += component_density
    return density


samples = []
max_density = 10

while len(samples) < 1000:
    x, y = np.random.uniform(low=-10, high=10, size=2)
    point = np.array([x, y])
    density = density_function(point, gmm.weights_, gmm.means_, gmm.covariar
    if np.random.uniform(0, max_density) < density:
        samples.append(point)

samples = np.array(samples)


plt.figure()
plt.scatter(samples[:, 0], samples[:, 1], alpha=0.6)
plt.title("Sampled Datapoints from GMM")
plt.show()
```

In [ ]:

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

# Extract x and y
x = data[:, 0]
y = data[:, 1]

# Define the borders of the grid
deltaX = (max(x) - min(x)) / 10
deltaY = (max(y) - min(y)) / 10
xmin = min(x) - deltaX
xmax = max(x) + deltaX
ymin = min(y) - deltaY
ymax = max(y) + deltaY


# Create meshgrid
xx, yy = np.mgrid[xmin:xmax:100j, ymin:ymax:100j]
# coordinates of the points that make the grid
positions = np.vstack([xx.ravel(), yy.ravel()]).T
```

In [ ]:

```
def density(points, pi, mu, sigma, gmm):
    """Evaluate the density at each point on the grid.
    Args:
        points: (N, 2) numpy array containing the coordinates of the points
        pi: (K,) numpy array containing the mixture coefficients for each cl
        mu: (K, D) numpy array containing the means of each cluster
        sigma: (K, D, D) numpy array containing the covariance matrixes of e
        gmm: an instance of the GMM model

    Return:
        densities: (N, ) numpy array containing densities at each point on t

    HINT: You should be using the formula given in the hints.
    """
    # TODO: Implement this function


# get the density at each coordinate on the grid
densities = np.reshape(density(positions, pi, mu, sigma, gmm), xx.shape)
```

Undergrads (if you have used normalPDF in the density function)

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

fig = plt.figure(figsize=(13, 7), dpi=100)
ax = plt.axes(projection="3d")
surf = ax.plot_surface(
    xx, yy, densities, rstride=1, cstride=1, cmap="coolwarm", edgecolor="nor
)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("PDF")
ax.set_title("Surface plot of 2D Gaussian Mixture Densities")
fig.colorbar(surf, shrink=0.5, aspect=5)  # add color bar indicating the PDF
ax.view_init(60, 35)
```

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

fig = plt.figure(figsize=(13, 7), dpi=100)
ax = plt.axes(projection="3d")
w = ax.plot_wireframe(xx, yy, densities)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("PDF")
ax.set_title("Wireframe plot of 2D Gaussian Mixture")
```

Grads and Undergrads (if you have used multinormalPDF in the density function)

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

fig = plt.figure(figsize=(13, 7), dpi=100)
ax = plt.axes(projection="3d")
surf = ax.plot_surface(
    xx, yy, densities, rstride=1, cstride=1, cmap="coolwarm", edgecolor="nor
)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("PDF")
ax.set_title("Surface plot of 2D Gaussian Mixture Densities")
fig.colorbar(surf, shrink=0.5, aspect=5)  # add color bar indicating the PDF
ax.view_init(60, 35)
```

In [ ]:

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

fig = plt.figure(figsize=(13, 7), dpi=100)
ax = plt.axes(projection="3d")
w = ax.plot_wireframe(xx, yy, densities)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("PDF")
ax.set_title("Wireframe plot of 2D Gaussian Mixture")
```

In [ ]:

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



def rejection_sample(xmin, xmax, ymin, ymax, gmm, dmax=1, M=0.1):
    """Performs rejection sampling. Keep sampling datapoints until d <= f(x,
    Args:
        xmin: lower bound on x values
        xmax: upper bound on x values
        ymin: lower bound on y values
        ymax: upper bound on y values
        gmm: an instance of the GMM model
        dmax: the upper bound on d
        M: scale_factor. can be used to control the fraction of samples that

    Return:
        x, y: the coordinates of the sampled datapoint

    HINT: Refer to the links in the hints
    """
    while True:
        x = np.random.uniform(low=xmin, high=xmax)
        y = np.random.uniform(low=ymin, high=ymax)
        d = np.random.uniform(low=0, high=dmax)
        if d < density(np.array([x, y]).reshape(1, 2), pi, mu, sigma, gmm) /
            return x, y
```

In [ ]:

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



# Sample datapoints using Rejection Sampling
generated_datapoints = np.zeros((1000, 2))
i = 0
while i < 1000:
    generated_datapoints[i, 0], generated_datapoints[i, 1] = rejection_sampl
```

```
        xmin, xmax, ymin, ymax, gmm, dmax=1
    )
    if i % 100 == 0:
        print(i)
    i += 1
```

Grads and Undergrads (if you have used multinormalPDF in the density function)

In [ ]:

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

plt.scatter(generated_datapoints[:, 0], generated_datapoints[:, 1])
plt.axis("equal")
plt.title("Sampled Datapoints")
plt.show()
```

4. (Bonus for All) Cleaning Messy data and semi-supervised learning [8% Bonus for All]

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

Congratulations! you recently graduated with your shiny GT degree. You've decided to pursue your childhood dream to study astrophysics. Stationed at a cutting edge Gamma Ray Telescope facility as a Data Scientist, delving into the high-energy physics of the universe, your mission is to probe the enigmatic Sagittarius A*, the supermassive black hole in the center of our galaxy, using a telescope that views the cosmos through gamma rays emitted by the most cataclysmic events in space like neutron star mergers, pulsars, and the voracious accretion disks of black holes. SUPER EXCITING!

The cutting edge telescope you are working with detects these really high energy gamma ray emissions from really small windows in the sky. You find that the telescope can be configured with a few parameters to detect these particles. These parameters, 10 in number, range from the telescope's orientation and timing precision to the sensitivity settings that find the faintest gamma signals against the cosmic background.

However, your cosmic quest faces an unexpected challenge. A data corruption incident has left a 15% void across your dataset, affecting both the intricate telescope parameters and the critical gamma ray detection records. This isn't just a minor hiccup; it's a significant obstacle in your path to unraveling the mysteries of our galaxy's heart.

But there's a silver lining. You remember that the machine learning techniques learnt in CS4641/7641 are the key to navigating this data loss issue. This challenge transforms into an opportunity to showcase the resilience and ingenuity of data science in the face of adversity. How will you leverage your skills to reconstruct the missing pieces and ensure that your exploration of Sagittarius A* yields groundbreaking insights into the universe's most profound secrets?

Your task now is to clean the data and implementing a semi-supervised learning framework to classify the detection of gamma rays for your experiments. The data has 10 feature columns containing the telescope's parameters and one column containing a binary label containing either 0 or 1 representing the absence or a presence of a signal.

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 the intern deleted the datapoints

4.1 Data Cleaning [2.8%]

4.1.a Data Separating [0.7%]

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 **semisupervised.py**, implement the following methods:

- complete_
- incomplete_
- unlabeled_

No description has been provided for this image

In [ ]:

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

localtests.SemisupervisedTests().test_data_separating_methods()
```

4.1.b KNN [1.4%]

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. Given that you are focusing on neighbouring points, the margin of error from actual missing values should be limited.

In the **CleanData** class in **semisupervised.py**, implement the following methods:

- pairwise_dist
- __call__

The unit test is a good expectation of what the process should look like on a toy dataset. If your output matches the answer, you are on the right track. Run the following cell to check.

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

In [ ]:

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

localtests.SemisupervisedTests().test_cleandata()
```

4.1.c Mean of Features [0.7%]

Another method of filling the missing values is by using the mean of individual features. The mean of all non-NaN values of a feature could be used to replace any NaN value belonging to the feature. Implement the mean_clean_data method in accordance with this rule. NOTE: There should be no NaN values in the n*d array that you return from mean_clean_data.

In **semisupervised.py**, implement the following method:

- mean_clean_data

In [ ]:

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

```
###############################

localtests.SemisupervisedTests().test_mean_clean_data()
```

4.2 Semi-supervised Learning [3.5%]

4.2.a Getting acquainted with semi-supervised learning approaches. [1.2%]

Take a look at the algorithm presented in Table 1 of the paper "Text Classification from Labeled and Unlabeled Documents using EM" by Nigam et al. (2000). While you are recommended to read the whole paper this assignment focuses on items 5.1, 5.2, and 6.1. Write a brief summary of three interesting highlights of the paper (50-words maximum).

The study demonstrates how integrating unlabeled data with a small labeled dataset can significantly improve classification accuracy, showcasing the value of semi-supervised learning in scenarios with limited labeled data. It also details the adaptation of the EM algorithm for text classification, where the E-step estimates the distribution of class labels for unlabeled documents, and the M-step updates the model parameters based on these estimations, iteratively refining the classification model. The paper also provides empirical evidence that the semi-supervised learning model outperforms traditional supervised learning methods when labeled data is scarce, highlighting the potential of semi-supervised approaches in enhancing learning efficiency and accuracy with additional unlabeled data.

4.2.b Implementing the EM algorithm. [2.3%]

Implement the EM algorithm proposed by Nigam et al. (2000) on Table 1, using a Gaussian Naive Bayes (GNB) classifier instead of 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 the **SemiSupervised** class in **semisupervised.py**, implement the following methods:

- _init_components
- __call__

4.3 Demonstrating the performance of the algorithm. [1.1%]

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.

In the **ComparePerformance** class in **semisupervised.py**, implement the following method:

- accuracy_semi_supervised
- accuracy_GNB

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 > 71%
- accuracy_cleaned_data > 71%
- accuracy_semi_supervised > 69%

In [ ]:

```
###############################
### DO NOT CHANGE THIS CELL ###
###############################
from semisupervised import (
    CleanData,
    ComparePerformance,
    complete_,
    incomplete_,
    mean_clean_data,
    unlabeled_,
)
```

In [ ]:

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

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

# Separate training data into categories: labeled complete, labeled incomple
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)

# Category for data that is guaranteed to have label values
labeled_data = np.concatenate((labeled_complete, labeled_incomplete), 0)

# Perform mean data cleaning on all labeled data
mean_cleaned_data = mean_clean_data(labeled_data)
```

```
# 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"Labeled shape:                  {labeled_data.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, valida
# =========================================================================
# SUPERVISED GNB WITH MEAN CLEAN DATA (SKLEARN)
accuracy_mean_cleaned_data = ComparePerformance.accuracy_GNB(
    mean_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 * ac
)
print(
    f"Supervised with KNN clean data, GNB Accuracy:     {np.round(100.0 * ac
)
print(
    f"Supervised with Mean clean data, GNB Accuracy:    {np.round(100.0 * ac
)
print(
    f"SemiSupervised Accuracy:                          {np.round(100.0 * ac
)
```

4.4 Interpretation of Results. [0.6%]

What are the differences in using the kNN method and the mean method to fill NaN values? Explain in terms of the results you get from each.

The kNN and mean methods for imputing missing values offer distinct advantages and drawbacks, fundamentally influenced by the dataset's characteristics and the analytical goals. kNN imputation, leveraging the k most similar data points to estimate missing values, excels in preserving local data patterns and relationships between features, making it particularly suited for complex datasets where such relationships are crucial. However, it demands higher computational resources and is sensitive to the choice of k and the presence of outliers, which can skew imputations. In contrast, mean imputation replaces missing values with the feature's average across available data points, offering simplicity and computational efficiency. Despite its ease of implementation, it overlooks inter-feature relationships and is vulnerable to outliers, potentially leading to less accurate or misleading imputations in skewed distributions. Therefore, the selection between kNN and mean imputation should be guided by a careful consideration of the dataset’s structure, computational constraints, and the importance of capturing intricate data relationships for the analysis or predictive modeling at hand.

5. Evaluating Data Representation in K-Means Clustering [4pts]

A national healthcare system employs a K-means clustering algorithm to optimize healthcare resource distribution. Two datasets are used: one from Harborview, an underdeveloped city, and another from Greenfield, a developed city. The algorithm is applied to both datasets to identify healthcare resource allocation needs.

Datasets from vastly different settings might differ in incidence of chronic conditions, or preventive care and better healthcare access. Compare the heatmaps below:

```
In [ ]: %run Vis_DoNotChange.py
```

Question:
Which of the following statements are correct? (Select all that apply)

- A. Sensitivity to scale and scope of the data might result in urgency of certain health conditions getting overlooked.
- B. Uniform resource distribution between cities is the best ethical approach.
- C. Concentrating on general data trends, the algorithm may overlook the specific healthcare needs of smaller, underrepresented groups.
- D. If data contains underrepresentation of certain demographic groups, existing disparities (as shown by the plot) might be magnified.
- E. The outputs of unsupervised algorithms like K-means clustering are inherently unbiased, as they do not rely on pre-labeled data.

Answer = A, C, D