XCS229i Problem Set 5

Due Friday, 20 September 2020.

Guidelines

1. These questions require thought, but do not require long answers. Please be as concise as possible.
2. If you have a question about this homework, we encourage you to post your question in our Slack workspace, at `http://xcs229i-scpd.slack.com/`
3. Familiarize yourself with the collaboration and honor code policy before starting work.
4. For the coding problems, you may not use any libraries except those defined in the provided starter code. In particular, ML-specific libraries such as `scikit-learn` are not permitted.

Submission Instructions

Written Submission: All students must submit an electronic PDF version of the written questions. We highly recommend typesetting your solutions via LaTeX, though it is not required. If you choose to hand write your responses, please make sure they are well organized and legible when scanned. The source LaTeXfor all problem sets is available on GitHub.

Coding Submission: All students must also submit a zip file of their source code. Create a submission using the following bash command:

```
zip -j ps5_submission.zip src/k_means/k_means.py src/semi_supervised/gmm.py
```

If you are **NOT** able to successfully zip your code using the following bash command or do **NOT** have the zip command line tool on your machine, please run the following python script to zip your code as an alternative:

```
python zip_submission.py
```

You should make sure to (1) restrict yourself to only using libraries included in the starter code, and (2) make sure your code runs without errors. Your submission will be evaluated by the auto-grader using a private test set and will be used for verifying the outputs reported in the writeup.

Before beginning the programming assignments in this course, we highly recommend you walk through our Anaconda tutorial to familiarize yourself with our coding environment.

Honor code: We strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down the solutions independently, and without referring to written notes from the joint session. In other words, each student must understand the solution well enough in order to reconstruct it by him/herself. In addition, each student should write on the problem set the set of people with whom s/he collaborated. Further, because we occasionally reuse problem set questions from previous years, we expect students not to copy, refer to, or look at the solutions in preparing their answers. It is an honor code violation to intentionally refer to a previous year's solutions.

1. **[7 points] K-means for compression**

In this problem, we will apply the K-means algorithm to lossy image compression by reducing the number of colors used in an image.

We will be using the files `src/k_means/peppers-small.tiff` and `src/k_means/peppers-large.tiff`.

The `peppers-large.tiff` file contains a 512x512 image of peppers represented in 24-bit color. This means that, for each of the $512 \times 512 = 262,144$ pixels in the image, there are three 8-bit numbers (each ranging from 0 to 255) that represent the red, green, and blue intensity values for that pixel. The straightforward representation of this image therefore takes about $262,144 \times 3 = 786,432$ bytes (a byte being 8 bits). To compress the image, we will use K-means to reduce the image to $k = 16$ colors. More specifically, each pixel in the image is considered a point in the three-dimensional (r, g, b)-space. To compress the image, we will cluster these points in color-space into 16 clusters, and replace each pixel with the closest cluster centroid.

Follow the instructions below. Be warned that some of these operations can take a while (several minutes even on a fast computer)!

(a) **[5 point(s) Coding] K-Means Compression Implementation.** First let us *look* at our data. From the `src/k_means/` directory, open an interactive Python prompt, and type

```
from matplotlib.image import imread; import matplotlib.pyplot as plt;
```

and run `A = imread('peppers-large.tiff')`. Now, `A` is a "three dimensional matrix," and `A[:,:,0]`, `A[:,:,1]` and `A[:,:,2]` are 512x512 arrays that respectively contain the red, green, and blue values for each pixel. Enter `plt.imshow(A); plt.show()` to display the image.

Since the large image has 262,144 pixels and would take a while to cluster, we will instead run vector quantization on a smaller image. Repeat (a) with `peppers-small.tiff`.

Next we will implement image compression in the file `src/k_means/k_means.py` which has some starter code. Treating each pixel's (r, g, b) values as an element of $\mathbb{R}^3$, implement K-means with 16 clusters on the pixel data from this smaller image, iterating (preferably) to convergence, but no more than 30 iterations (i.e. the variable `max_iter` in `k_means.py`). For initialization, set each cluster centroid to the (r, g, b)-values of a randomly chosen pixel in the image.

Take the image of `peppers-large.tiff`, and replace each pixel's (r, g, b) values with the value of the closest cluster centroid from the set of centroids computed with `peppers-small.tiff`. Consider visually comparing it to the original image to verify that your implementation is reasonable.

(b) **[2 point(s) Written] Compression Factor.**

If we represent the image with these reduced (16) colors, by (approximately) what factor have we compressed the image?

The originial picture was of the format (512, 512, 24) with the 24 counted as *bits* or (512, 512, 3) with the 3 counted as *bytes*. If we compress the picture to contain only 16 colors, we only need 4 bits to store these 16 different values. Therefore, the picture can be stored as a (512, 512, 4) bit picture. Hereby, it is compressed by a factor of $24/4 = 6$.

On the other hand, before there was a possible range of $16,777,216 = 256^3$ different colors. Now we have just 16 different. This gives a ratio of $\frac{256^3}{16} = 1,048,576$ times fewer possible colors in the picture.

2. **[33 points] Semi-supervised EM**

Expectation Maximization (EM) is a classical algorithm for unsupervised learning (*i.e.,* learning with hidden or latent variables). In this problem we will explore one of the ways in which the EM algorithm can be adapted to the semi-supervised setting, where we have some labelled examples along with unlabelled examples.

In the standard unsupervised setting, we have $n \in \mathbb{N}$ unlabelled examples $\{x^{(1)}, \dots, x^{(n)}\}$. We wish to learn the parameters of $p(x, z; \theta)$ from the data, but $z^{(i)}$'s are not observed. The classical EM algorithm is designed for this very purpose, where we maximize the intractable $p(x; \theta)$ indirectly by iteratively performing the E-step and M-step, each time maximizing a tractable lower bound of $p(x; \theta)$. Our objective can be concretely written as:

$$\begin{aligned}\ell_{\text{unsup}}(\theta) &= \sum_{i=1}^{n} \log p(x^{(i)}; \theta) \\ &= \sum_{i=1}^{n} \log \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}; \theta)\end{aligned}$$

Now, we will attempt to construct an extension of EM to the semi-supervised setting. Let us suppose we have an *additional* $\tilde{n} \in \mathbb{N}$ labelled examples $\{(\tilde{x}^{(1)}, \tilde{z}^{(1)}), \dots, (\tilde{x}^{(\tilde{n})}, \tilde{z}^{(\tilde{n})})\}$ where both x and z are observed. We want to simultaneously maximize the marginal likelihood of the parameters using the unlabelled examples, and full likelihood of the parameters using the labelled examples, by optimizing their weighted sum (with some hyperparameter α). More concretely, our semi-supervised objective $\ell_{\text{semi-sup}}(\theta)$ can be written as:

$$\begin{aligned}\ell_{\text{sup}}(\theta) &= \sum_{i=1}^{\tilde{n}} \log p(\tilde{x}^{(i)}, \tilde{z}^{(i)}; \theta) \\ \ell_{\text{semi-sup}}(\theta) &= \ell_{\text{unsup}}(\theta) + \alpha \ell_{\text{sup}}(\theta)\end{aligned}$$

We can derive the EM steps for the semi-supervised setting using the same approach and steps as before. You are *strongly encouraged* to show to yourself (no need to include in the write-up) that we end up with:

E-step (semi-supervised)

For each $i \in \{1, \dots, n\}$, set

$$Q_i^{(t)}(z^{(i)}) := p(z^{(i)} | x^{(i)}; \theta^{(t)})$$

M-step (semi-supervised)

$$\theta^{(t+1)} := \arg\max_{\theta} \left[\sum_{i=1}^{n} \left(\sum_{z^{(i)}} Q_i^{(t)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i^{(t)}(z^{(i)})} \right) + \alpha \left(\sum_{i=1}^{\tilde{n}} \log p(\tilde{x}^{(i)}, \tilde{z}^{(i)}; \theta) \right) \right]$$

(a) **[5 point(s) Written] Convergence.** First we will show that this algorithm eventually converges. In order to prove this, it is sufficient to show that our semi-supervised objective $\ell_{\text{semi-sup}}(\theta)$ monotonically increases with each iteration of E and M step. Specifically, let $\theta^{(t)}$ be the parameters obtained at the end of t EM-steps. Show that $\ell_{\text{semi-sup}}(\theta^{(t+1)}) \geq \ell_{\text{semi-sup}}(\theta^{(t)})$.

BEGIN PROOF HERE

$$\begin{aligned}
\ell(\theta^{(t+1)}) &= \alpha\ell_{\text{sup}}(\theta^{(t+1)}) + \ell_{\text{unsup}}(\theta^{(t+1)}) && \text{Definition} \\
&\geq \alpha\ell_{\text{sup}}(\theta^{(t+1)}) + \sum_{i=1}^{n}\sum_{z^{(i)}} Q_i^{(t)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta^{(t+1)})}{Q_i^{(t)}(z^{(i)})} && \text{Jensen's inequality} \\
&\geq
\end{aligned}$$

END PROOF

ANSWER
BEGIN PROOF HERE

$$\begin{aligned}
\ell(\theta^{(t+1)}) &= \alpha\ell_{\text{sup}}(\theta^{(t+1)}) + \ell_{\text{unsup}}(\theta^{(t+1)}) && \text{Definition} \\
&\geq \alpha\ell_{\text{sup}}(\theta^{(t+1)}) + \sum_{i=1}^{n}\sum_{z^{(i)}} Q_i^{(t)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta^{(t+1)})}{Q_i^{(t)}(z^{(i)})} && \text{Jensen's inequality} \\
&\geq \alpha\ell_{\text{sup}}(\theta^{(t)}) + \sum_{i=1}^{n}\sum_{z^{(i)}} Q_i^{(t)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta^{(t)})}{Q_i^{(t)}(z^{(i)})} \\
&= \ell(\theta^{(t)})
\end{aligned}$$

END PROOF

This simple proof is true due to the definition of the E- and M-Step. Here we let θ denote the model parameters ϕ, μ, Σ.

First, in the E-step you choose such a Q_i that the Lower Bound equals the Upper Bound - that you get a tight Lower Bound. This means that the Log-Likelihood for t and $t+1$ *equals* each other at this given $\theta^{(t)}$.

Second, in the M-step we maximize this Lower Bound. So, by maximizing this expression (`argmax`), the $(t+1)$ parameters has to be greater than or equal to (due to maximization and that the Lower Bound = Upper Bound) at the point.

Semi-supervised GMM

Now we will revisit the Gaussian Mixture Model (GMM), to apply our semi-supervised EM algorithm. Let us consider a scenario where data is generated from $k \in \mathbb{N}$ Gaussian distributions, with unknown means $\mu_j \in \mathbb{R}^d$ and covariances $\Sigma_j \in \mathbb{S}_+^d$ where $j \in \{1, \dots, k\}$. We have n data points $x^{(i)} \in \mathbb{R}^d, i \in \{1, \dots, n\}$, and each data point has a corresponding latent (hidden/unknown) variable $z^{(i)} \in \{1, \dots, k\}$ indicating which distribution $x^{(i)}$ belongs to. Specifically, $z^{(i)} \sim \text{Multinomial}(\phi)$, such that $\sum_{j=1}^{k} \phi_j = 1$ and $\phi_j \geq 0$ for all j, and $x^{(i)}|z^{(i)} \sim \mathcal{N}(\mu_{z^{(i)}}, \Sigma_{z^{(i)}})$ i.i.d. So, μ, Σ, and ϕ are the model parameters.

We also have additional $\tilde{n}$ data points $\tilde{x}^{(i)} \in \mathbb{R}^d, i \in \{1, \dots, \tilde{n}\}$, and an associated *observed* variable $\tilde{z}^{(i)} \in \{1, \dots, k\}$ indicating the distribution $\tilde{x}^{(i)}$ belongs to. Note that $\tilde{z}^{(i)}$ are known constants (in contrast to $z^{(i)}$ which are unknown *random* variables). As before, we assume $\tilde{x}^{(i)}|\tilde{z}^{(i)} \sim \mathcal{N}(\mu_{\tilde{z}^{(i)}}, \Sigma_{\tilde{z}^{(i)}})$ are i.i.d.

In summary we have $n + \tilde{n}$ examples, of which n are unlabelled data points x's with unobserved z's, and $\tilde{n}$ are labelled data points $\tilde{x}^{(i)}$ with corresponding observed labels $\tilde{z}^{(i)}$. The traditional EM algorithm is designed to take only the n unlabelled examples as input, and learn the model parameters μ, Σ, and ϕ.

Our task now will be to apply the semi-supervised EM algorithm to GMMs in order to also leverage the additional $\tilde{n}$ labelled examples, and come up with semi-supervised E-step and M-step update rules specific to GMMs. Whenever required, you can cite the lecture notes for derivations and steps.

(b) **[5 point(s) Written] Semi-supervised E-Step.** Clearly state the latent variables that need to be re-estimated in the E-step. Derive the E-step to re-estimate all the stated latent variables. Your final E-step expression must only involve x, z, μ, Σ, ϕ and universal constants.

BEGIN PROOF HERE

First, we see that this model has a supervised and unsupervised part. Since only the unsupervised part has some latent variables that need to be re-estimated (for supervised, the weights w are constants and not random), the focus will be set on this part of the model in this question - in other words, the E-Step (Expectation Step) is only relevant for the unsupervised Data points.
In this E-Step, we need to find the "weights", w.

$$w_j^{(i)} := p(z^{(i)} = j|x^{(i)}; \phi, \mu, \Sigma)$$

Where applying Bayes' Rule we get

$$p(z^{(i)} = j|x^{(i)}; \phi, \mu, \Sigma) = \frac{p(x^{(i)}|z^{(i)} = j; \mu_j, \Sigma_j) \cdot \phi_j}{\sum_{l=1}^{k} p(x^{(i)}|z^{(i)} = l; \mu_l, \Sigma_l) \cdot \phi_l}$$

We know that $p(x|z = j)$ is Gaussian distributed with the j'th gaussian using the j-parameters. Using the PDF for Multivariate Gaussian we end up with

$$p(z^{(i)} = j|x^{(i)}; \phi, \mu, \Sigma) = \frac{\frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma_j|^{\frac{1}{2}}} exp\left(-\frac{1}{2}(x-\mu_j)^T\Sigma^{-1}(x-\mu_j)\right) \cdot \phi_j}{\sum_{l=1}^{k} \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma_l|^{\frac{1}{2}}} exp\left(-\frac{1}{2}(x-\mu_l)^T\Sigma^{-1}(x-\mu_l)\right) \cdot \phi_l}$$

$$p(z^{(i)} = j|x^{(i)}; \phi, \mu, \Sigma) = \frac{\frac{1}{|\Sigma_j|^{\frac{1}{2}}} exp\left(-\frac{1}{2}(x-\mu_j)^T\Sigma^{-1}(x-\mu_j)\right) \cdot \phi_j}{\sum_{l=1}^{k} \frac{1}{|\Sigma_l|^{\frac{1}{2}}} exp\left(-\frac{1}{2}(x-\mu_l)^T\Sigma^{-1}(x-\mu_l)\right) \cdot \phi_l}$$

Now w is found. It denotes how likely each datapoint is to be distributed for a specific Gaussian distribution. And it clear to see from the Bayes' Rule that all w_j sum to 1 (that $x^{(i)}$ *is* distributed by one of these K Gaussians)
END PROOF

(c) **[10 point(s) Written] Semi-supervised M-Step.** Clearly state the parameters that need to be re-estimated in the M-step. Derive the M-step to re-estimate all the stated parameters. Specifically, derive closed form expressions for the parameter update rules for $\mu^{(t+1)}$, $\Sigma^{(t+1)}$ and $\phi^{(t+1)}$ based on the semi-supervised objective.

BEGIN PROOF HERE
List the parameters which need to be re-estimated in the M-step:

ANSWER

In this M-step we need to re-estimate the following parameters: μ, ϕ, Σ. The weights that are being used in the closed-form update rules are found in E-Step.

In order to simplify derivation, it is useful to denote

$$w_j^{(i)} = Q_i^{(t)}(z^{(i)} = j),$$

and

$$\tilde{w}_j^{(i)} = \begin{cases} \alpha & \tilde{z}^{(i)} = j \\ 0 & \text{otherwise.} \end{cases}$$

We further denote $S = \Sigma^{-1}$, and note that because of the chain rule, $\nabla_S \ell = 0 \Rightarrow \nabla_\Sigma \ell = 0$. So we choose to rewrite the M-step in terms of S and maximize it w.r.t S, and re-express the resulting solution back in terms of Σ.
Based on this, the M-step becomes (The following simplification is not required for full credit, but will help later derivations and may be used by the teaching staff as justification for awarding partial credit):

$$\phi^{(t+1)}, \mu^{(t+1)}, S^{(t+1)} = \arg\max_{\phi,\mu,S} \left(\sum_{i=1}^{n} \sum_{j=1}^{k} Q_i^{(t)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \phi, \mu, S)}{Q_i^{(t)}(z^{(i)})} + \alpha \sum_{i=1}^{\tilde{n}} \log p(\tilde{x}^{(i)}, \tilde{z}^{(i)}; \phi, \mu, S) \right)$$

$$=$$

Now, calculate the update steps by maximizing the expression within the argmax for each parameter (We will do the first for you).

ϕ_j: We construct the Lagrangian including the constraint that $\sum_{j=1}^{k} \phi_j = 1$, and absorbing all irrelevant terms into constant C:

$$\begin{aligned}
\mathcal{L}(\phi, \beta) &= C + \sum_{i=1}^{n}\sum_{j=1}^{k} w_j^{(i)} \log \phi_j + \sum_{i=1}^{\tilde{n}}\sum_{j=1}^{k} \tilde{w}_j^{(i)} \log \phi_j + \beta \left(\sum_{j=1}^{k} \phi_j - 1 \right) \\
\nabla_{\phi_j} \mathcal{L}(\phi, \beta) &= \sum_{i=1}^{n} w_j^{(i)} \frac{1}{\phi_j} + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)} \frac{1}{\phi_j} + \beta = 0 \\
\Rightarrow \phi_j &= \frac{\sum_{i=1}^{n} w_j^{(i)} + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)}}{-\beta} \\
\nabla_{\beta} \mathcal{L}(\phi, \beta) &= \sum_{j=1}^{k} \phi_j - 1 = 0 \\
&\Rightarrow \sum_{j=1}^{k} \frac{\sum_{i=1}^{n} w_j^{(i)} + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)}}{-\beta} = 1 \\
&\Rightarrow -\beta = \sum_{j=1}^{k} \left(\sum_{i=1}^{n} w_j^{(i)} + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)} \right) \\
\Rightarrow \phi_j^{(t+1)} &= \frac{\sum_{i=1}^{n} w_j^{(i)} + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)}}{\sum_{j=1}^{k} \left(\sum_{i=1}^{n} w_j^{(i)} + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)} \right)} \\
&= \frac{\sum_{i=1}^{n} w_j^{(i)} + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)}}{n + \alpha \tilde{n}}
\end{aligned}$$

μ_j: Next, derive the update for μ_j. Do this by maximizing the expression with the argmax above with respect to μ_j.

First, calculate the gradient with respect to μ_j:

$\nabla_{\mu_j} =$

ANSWER

Citing the lecture notes, the gradient with respect to μ_j in the *unsupervised* you get

$$\nabla_{\mu_j} = \sum_{i=1}^{n} w_j^{(i)} (\Sigma_j^{-1} x^{(i)} - \Sigma_j^{-1} \mu_j)$$

Since using $\tilde{w}_j$ instead in the supervised term (where the weight is a constant rather then random), we get the almost same expression when taking the gradient with respect to μ_j in the *supervised*:

$$\nabla_{\mu_j} = \sum_{i=1}^{n} \tilde{w}_j^{(i)} (\Sigma_j^{-1} \tilde{x}^{(i)} - \Sigma_j^{-1} \mu_j)$$

So the total gradient of the *semi-supervised* expression above turns to be

$$\nabla_{\mu_j} = \sum_{i=1}^{n} w_j^{(i)} (\Sigma_j^{-1} x^{(i)} - \Sigma_j^{-1} \mu_j) + \sum_{i=1}^{n} \tilde{w}_j^{(i)} (\Sigma_j^{-1} x^{(i)} - \Sigma_j^{-1} \mu_j)$$

Setting this to 0 and solving for μ_j yields the result

$$\mu_j^{(t+1)} = \frac{\sum_{i=1}^{n} w_j^{(i)} x^{(i)} + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)} \tilde{x}^{(i)}}{\sum_{i=1}^{n} w_j^{(i)} + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)}}$$

$\mathbf{\Sigma}_j$: Finally, derive the update for Σ_j via S_j. Again, do this by maximizing the expression with the argmax above with respect to S_j.

.

First, calculate the gradient with respect to S_j:

$\nabla_{S_j} =$

ANSWER

Here, I use the same principle as above. Citing the lecture notes for gradient for the *unsupervised* case and swap w with $\tilde{w}$.

The above is *unsupervised* and bottom gradient is *supervised*:

$$\nabla_{S_j} = w_j^{(i)} S - \sum_{i=1}^{n} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T$$
$$\nabla_{S_j} = \tilde{w}_j^{(i)} S - \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T$$

So the total gradient is

$$\nabla_{S_j} = w_j^{(i)} S - \sum_{i=1}^{n} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T + \tilde{w}_j^{(i)} S - \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T$$

Setting equals 0 and solving for S we get:

$$S_j^{(t+1)} = \frac{\sum_{i=1}^{n} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{n} w_j^{(i)} + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)}}$$

This results in the final set of update expressions:

$$\phi_j := \frac{\sum_{i=1}^{n} w_j^{(i)} + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)}}{n + \alpha \tilde{n}}$$

$$\mu_j := \frac{\sum_{i=1}^{n} w_j^{(i)} x^{(i)} + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)} \tilde{x}^{(i)}}{\sum_{i=1}^{n} w_j^{(i)} + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)}}$$

$$\Sigma_j := \frac{\sum_{i=1}^{n} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{n} w_j^{(i)} + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)}}$$

END PROOF

-

(d) **[5 point(s) Coding] Classical (Unsupervised) EM Implementation.** For this sub-question, we are only going to consider the n unlabelled examples. Follow the instructions in `src/semi_supervised_em/gmm.py` to implement the traditional EM algorithm, and run it on the unlabelled data-set until convergence.

To verify a correct implementation, consider running three trials and using the provided plotting function to construct a scatter plot of the resulting assignments to clusters (one plot for each trial). Your plot will indicate cluster assignments by assigning unique colors for each cluster (*i.e.*, the cluster which had the highest probability in the final E-step). Your plots are not graded.

Your plots should look similar to the following:

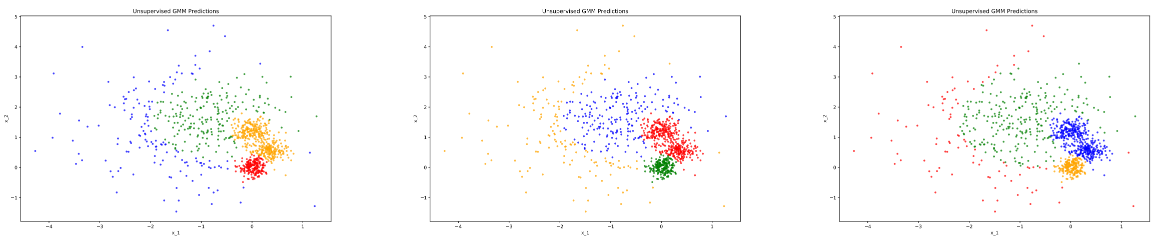

Figure 1: Predictions made by GMM model with unsupervised EM.

(e) **[5 point(s) Coding] Semi-supervised EM Implementation.** Now we will consider both the labelled and unlabelled examples (a total of $n + \tilde{n}$), with 5 labelled examples per cluster. We have provided starter code for splitting the dataset into matrices `x` and `x_tilde` of unlabelled and labelled examples respectively. Add to your code in `src/semi_supervised_em/gmm.py` to implement the modified EM algorithm, and run it on the dataset until convergence.

To verify a correct implementation, consider creating a plot for each trial, as done in the previous sub-question.

Your plots should look similar to the following (your plots are not graded):

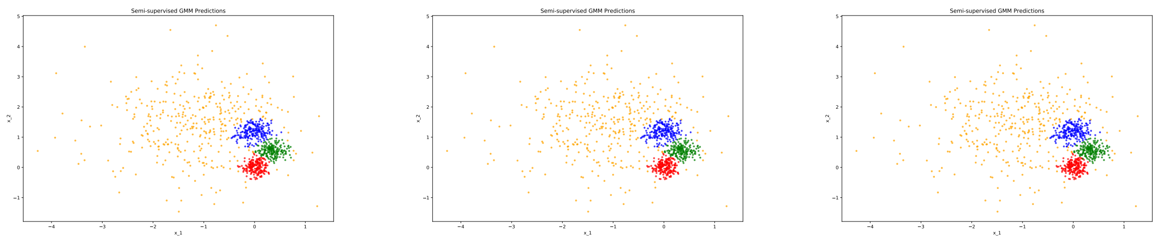

Figure 2: Predictions made by GMM model with semi-supervised EM.

(f) **[3 point(s) Written] Comparison of Unsupervised and Semi-supervised EM.** Briefly describe the differences you saw in unsupervised *vs.* semi-supervised EM for each of the following:

i. Number of iterations taken to converge.

ii. Stability (*i.e.,* how much did assignments change with different random initializations?)

iii. Overall quality of assignments.

Note: The dataset was sampled from a mixture of three low-variance Gaussian distributions, and a fourth, high-variance Gaussian distribution. This should be useful in determining the overall quality of the assignments that were found by the two algorithms.

Even though only 2 % of the data is labelled, it is clear that this affects the result and performance of the algorithms:

It is shown that the number of iterations before convergence for the Semi-supervised is lower than purely unsupervised. More than twice as fast. It is able to detech the Gaussians faster due to the labelled data points.

Furthermore, it is much more stable regarding assigning data to a Gaussian index. As seen on the pictures, it assigns the data to the same Gaussian at every trial due to the fact that some data points already are labelled to a certain Gaussian. Oppositely, the Unsupervised switches the Gaussian indeces (seen by the colors) because it for this algorithm doesnt make any changes in the result.