CS 229, Summer 2025
Problem Set #3

YOUR NAME HERE (`YOUR SUNET HERE`) COLLABORATORS (`COLLABORATORS`)

Due Friday, August 8 at 11:59 pm on Gradescope.

Notes: (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 on our Ed forum, at `https://edstem.org/us/courses/79825/discussion`.

(3) If you missed the first lecture or are unfamiliar with the collaboration or honor code policy, please read the policy on the course website before starting work.

(4) For the coding problems, you may not use any libraries except those defined in the provided `environment.yml` file. In particular, ML-specific libraries such as scikit-learn are not permitted.

(5) The due date is Friday, August 8 at 11:59 pm. If you submit after Friday, August 8 at 11:59 pm, you will begin consuming your late days. The late day policy can be found in the course website: Course Logistics and FAQ.

(6) Please tag your pages carefully! **We will deduct 0.5 points per question for incorrect page tags**.

All students must submit an electronic PDF version of the written question including plots generated from the codes. We **require you to typeset your solutions via LaTeX**. All students must also submit a zip file of their source code to Gradescope, which should be created using the `make_zip.py` script. You should make sure to (1) restrict yourself to only using libraries included in the `environment.yml` file, and (2) make sure your code runs without errors. Your submission may be evaluated by the auto-grader using a private test set, or used for verifying the outputs reported in the writeup. Please make sure that your PDF file and zip file are submitted to the corresponding Gradescope assignments respectively. We reserve the right to not give any points to the written solutions if the associated code is not submitted.

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 themself. 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.

Regarding Notation: The notation used in this problem set matches the notation used in the lecture notes. Some notable differences from lecture notation:

- The superscript "(i)" represents an index into the training set – for example, $x^{(i)}$ is the i-th feature vector and $y^{(i)}$ is the i-th output variable. In lecture notation, these would instead be expressed as x_i and y_i.
- The subscript j represents an index in a vector – in particular, $x_j^{(i)}$ represents the j-th feature in the feature vector $x^{(i)}$. In lecture notation, $x_j^{(i)}$ would be $h_j(x_i)$ or $x_i[j]$.

- The vector that contains the weights parameterizing a linear regression is expressed by the variable θ, whereas lectures use the variable $\mathbf{w}$. As such, $\theta_0 = w_0$, $\theta_1 = w_1$, and so on.

An overview of this notation is also given at the beginning of the lecture notes.

1. **[20 points] Decision Trees and Gini Loss**

When growing a decision tree, we split the input space in a greedy, top-down, recursive manner. Given a parent region R_p, we can choose a split $s_p(j,t)$ which yields two child regions $R_1 = \{X \mid x_j < t, X \in R_p\}$ and $R_2 = \{X \mid x_j \geq t, X \in R_p\}$. Assuming we have defined a per region loss $L(R)$, at each branch we select the split that minimizes the weighted loss of the children:

$$\min_{j,t} \frac{|R_1|L(R_1) + |R_2|L(R_2)}{|R_1| + |R_2|}$$

When performing classification, a commonly used loss is the Gini loss, defined for the K-class classification problem as:

$$G(R_m) = G(\vec{p}_m) = \sum_{k=1}^{K} p_{mk}(1 - p_{mk})$$

Where $\vec{p}_m = \begin{bmatrix} p_{m1} & p_{m2} & \dots & p_{mK} \end{bmatrix}$ and p_{mk} is the proportion of examples of class k that are are present in region R_m. However, we are oftentimes more interested in optimizing the final misclassification loss:

$$M(R_m) = M(\vec{p}_m) = 1 - \max_k p_{mk} \tag{1}$$

For the problems below, assume we are dealing with binary classification and that there are no degenerate cases where positive and negative datapoints overlap in the feature space.

(a) [5 points] Show that for any given split, the weighted Gini loss of the children can not exceed that of the parent. (**Hint**: first show that the Gini loss is strictly concave. And then use the fact that G is strictly concave meaning:

$$\forall p_1 \neq p_2, \forall t \in (0,1) : G(tp_1 + (1-t)p_2) > tG(p_1) + (1-t)G(p_2)$$

Answer:

(b) [5 points] List out the cases where Gini loss will stay the same after a split. Show why these do not violate the strong concavity of the Gini loss. Briefly explain why these cases do not prevent a fully grown tree from achieving zero Gini loss. (**Hint**: Recall the definition of strict concavity).

Answer:

(c) [4 points] If instead we use misclassification loss, what additional case causes the loss to stay the same after a split? Show why this is (hint: you may find it useful to define $N_m = |R_m|$ and N_{mk} as the number of examples of class k present in R_m).

Answer:

(d) [4 points]

Bagging, short for "bootstrap aggregating," is a powerful ensemble learning technique that aims to improve the stability and accuracy of machine learning algorithms. It leverages the concept of bootstrapping, which involves simulating the drawing of a new sample from the true underlying distribution of the training set, as the training set is presumed to be a

representative sample of the true distribution. In practice, this is done by generating new datasets through uniform sampling with replacement from the original dataset.

The "aggregating" component of bagging comes into play by repeating this bootstrapping process for each model in the ensemble, allowing each to be trained independently on a unique dataset. When considering decision trees, the method's utility becomes evident as it mitigates overfitting by ensuring that each tree in the ensemble is exposed to different subsets of the training data. This reduces the likelihood that the ensemble will fixate on particular data points, thus lowering overall variance. Statistically, each bootstrapped sample will contain, in expectation, about $1 - \frac{1}{e} \approx 63.2\%$ of unique data points from the original dataset.

However, the effectiveness of bagging depends on the characteristics of the underlying models. For models with low variance (and typically high bias), bagging may produce very similar models, which diminishes its benefits. On the other hand, with high-variance models such as decision trees, bagging capitalizes on the models' instability to promote diversity in the ensemble, thereby enhancing its performance. This results in an ensemble that maintains low bias while reducing variance, leading to a robust aggregate model.

Consider a training set X. In bootstrap sampling, each time we draw a random sample Z of size N from the training data and obtain $Z_1, Z_2, ..., Z_B$ after B times, i.e. we generate B different bootstrapped training data sets. If we apply bagging to regression trees, each time a tree $T_i (i = 1, 2, ..., B)$ is grown based on the bootstrapped data Z_i, and we average all the predictions to get:

$$\hat{T(x)} = \frac{1}{B} \sum_{i=1}^{B} T_i(x)$$

Now, if $T_1, T_2, ..., T_B$ is independent from each other, but each has the same variance σ^2, the variance of the average $\hat{T}$ is σ^2/B. However, in practice, the bagged trees could be similar to each other, resulting in correlated predictions. Assume $T_1, T_2, ..., T_B$ still share the same variance σ^2, but have a positive pair-wise correlation ρ. We define the correlation between two random variables as:

$$Corr(X, Y) = \frac{Cov(X, Y)}{\sqrt{Var(X)}\sqrt{Var(Y)}}$$

Thus, we have $\rho = Corr(T_i(x), T_j(x)), i \neq j$.

Show that in this case, the variance of the average is given by:

$$Var(\frac{1}{B} \sum_{i=1}^{B} T_i(x)) = \rho\sigma^2 + \frac{1 - \rho}{B}\sigma^2$$

Answer:

2. **[10 points] PCA**

In the course note (pg. 167-169), it is shown that PCA finds the "variance maximizing" directions onto which to project the data. In this problem, we find another interpretation of PCA.

Suppose we are given a set of points $\{x^{(1)}, \ldots, x^{(n)}\}$. Let us assume that we have as usual preprocessed the data to have zero-mean and unit variance in each coordinate. For a given unit-length vector u, let $f_u(x)$ be the projection of point x onto the direction given by u. I.e., if $\mathcal{V} = \{\alpha u : \alpha \in \mathbb{R}\}$, then

$$f_u(x) = \arg\min_{v \in \mathcal{V}} ||x - v||^2.$$

Show that the unit-length vector u that minimizes the mean squared error between projected points and original points corresponds to the first principal component for the data. I.e., show that

$$\arg\min_{u:u^Tu=1} \sum_{i=1}^{n} \|x^{(i)} - f_u(x^{(i)})\|_2^2 \ .$$

gives the first principal component.

Remark. If we are asked to find a k-dimensional subspace onto which to project the data so as to minimize the sum of squares distance between the original data and their projections, then we should choose the k-dimensional subspace spanned by the first k principal components of the data. This problem shows that this result holds for the case of $k = 1$.

Answer:

3. **[20 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 512×512 image of peppers represented in 24-bit color. This means that, for each of the 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 $262144 \times 3 = 786432$ 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) [15 points] **[Coding Problem] 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 512×512 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 in no case for less than 30 iterations. 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`. Visually compare it to the original image to verify that your implementation is reasonable. **Include in your write-up a copy of this compressed image alongside the original image. Answer:**

(b) [5 points] **Compression Factor.**

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

Answer:

4. **[35 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 EM algorithm can be adapted to the semi-supervised setting, where we have some labeled examples along with unlabeled examples.

In the standard unsupervised setting, we have $n \in \mathbb{N}$ unlabeled examples $\{x^{(1)}, \ldots, x^{(n)}\}$. 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* $\tilde{n} \in \mathbb{N}$ labeled examples $\{(\tilde{x}^{(1)}, \tilde{z}^{(1)}), \ldots, (\tilde{x}^{(\tilde{n})}, \tilde{z}^{(\tilde{n})})\}$ where both x and z are observed. We want to simultaneously maximize the marginal likelihood of the parameters using the unlabeled examples, and full likelihood of the parameters using the labeled 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 $i \in \{1, \ldots, n\}$, 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 points] **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 $\ell_{\text{semi-sup}}(\theta^{(t+1)}) \geq \ell_{\text{semi-sup}}(\theta^{(t)})$.
Answer:

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 $j \in \{1, \ldots, k\}$. We have n data points $x^{(i)} \in \mathbb{R}^d, i \in \{1, \ldots, n\}$, and each data point has a corresponding latent (hidden/unknown) variable $z^{(i)} \in \{1, \ldots, k\}$ 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 $x^{(i)}|z^{(i)} \sim \mathcal{N}\left(\mu_{z^{(i)}}, \Sigma_{z^{(i)}}\right)$ i.i.d. So, μ, Σ, and ϕ are the model parameters.

We also have additional $\tilde{n}$ data points $\tilde{x}^{(i)} \in \mathbb{R}^d, i \in \{1, \ldots, \tilde{n}\}$, and an associated *observed* variable $\tilde{z}^{(i)} \in \{1, \ldots, k\}$ 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 $\tilde{x}^{(i)}|\tilde{z}^{(i)} \sim \mathcal{N}\left(\mu_{\tilde{z}^{(i)}}, \Sigma_{\tilde{z}^{(i)}}\right)$ i.i.d.

In summary we have $n + \tilde{n}$ examples, of which n are unlabeled data points x's with unobserved z's, and $\tilde{n}$ are labeled data points $\tilde{x}^{(i)}$ with corresponding observed labels $\tilde{z}^{(i)}$. The traditional EM algorithm is designed to take only the n unlabeled 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 $\tilde{n}$ labeled 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 points] **Semi-supervised E-Step.** Clearly state which are all 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.

Answer:

(c) [10 points] **Semi-supervised M-Step.** Clearly state which are all 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 $\mu^{(t+1)}$ and $\phi^{(t+1)}$ based on the semi-supervised objective. As an example, we will derive the update rule for $\Sigma^{(t+1)}$ below.

In order to simplify derivation, we 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 chain rule of calculus, $\nabla_S \ell = 0 \Rightarrow \nabla_\Sigma \ell = 0$. 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:

$$\begin{aligned}\phi^{(t+1)}, \mu^{(t+1)}, S^{(t+1)} &= \arg\max_{\phi,\mu,S} \sum_{i=1}^{n}\sum_{j=1}^{k} Q_i^{(t)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \phi, \mu, S)}{Q_i^{(t)}(z^{(i)})} + \sum_{i=1}^{\tilde{n}} \log p(\tilde{x^{(i)}}, \tilde{z^{(i)}}; \phi, \mu, S) \\ &= \arg\max_{\phi,\mu,S} \sum_{i=1}^{n}\sum_{j=1}^{k} w_j^{(i)} \log \frac{\frac{|S_j|^{1/2}}{(2\pi)^{d/2}} \exp\left(-\frac{1}{2}(x^{(i)} - \mu_j)^T S_j (x^{(i)} - \mu_j)\right) \phi_j}{w_j^{(i)}} \\ &\quad + \sum_{i=1}^{\tilde{n}}\sum_{j=1}^{k} \tilde{w}_j^{(i)} \log \frac{\frac{|S_j|^{1/2}}{(2\pi)^{d/2}} \exp\left(-\frac{1}{2}(\tilde{x}^{(i)} - \mu_j)^T S_j (\tilde{x}^{(i)} - \mu_j)\right) \phi_j}{\tilde{w}_j^{(i)}}\end{aligned}$$

Then we derive the update for Σ_j via S_j (absorbing irrelevant constants into C):

$$\begin{aligned}0 &= \nabla_{S_j}(\ldots) \\ &= \nabla_{S_j}\left(C + \sum_{i=1}^{n} w_j^{(i)}\left(\log|S_j| - (x^{(i)} - \mu_j)^T S_j (x^{(i)} - \mu_j)\right) + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)}\left(\log|S_j| - (\tilde{x}^{(i)} - \mu_j)^T S_j (\tilde{x}^{(i)} - \mu_j)\right)\right) \\ &= \sum_{i=1}^{n} w_j^{(i)}\left(S_j^{-1} - (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T\right) + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)}\left(S_j^{-1} - (\tilde{x}^{(i)} - \mu_j)(\tilde{x}^{(i)} - \mu_j)^T\right) \\ &= \left(\sum_{i=1}^{n} w_j^{(i)} + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)}\right) S_j^{-1} - \left(\sum_{i=1}^{n} w_j^{(i)}(x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)}(\tilde{x}^{(i)} - \mu_j)(\tilde{x}^{(i)} - \mu_j)^T\right) \\ \Rightarrow \Sigma_j^{(t+1)} &= \frac{\sum_{i=1}^{n} w_j^{(i)}(x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)}(\tilde{x}^{(i)} - \mu_j)(\tilde{x}^{(i)} - \mu_j)^T}{\sum_{i=1}^{n} w_j^{(i)} + \sum_{i=1}^{\tilde{n}} \tilde{w}_j^{(i)}}\end{aligned}$$

This results in the update expression:

$$\Sigma_j := \frac{\sum_{i=1}^{n} w_j^{(i)}(x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T + \alpha \sum_{i=1}^{\tilde{n}} \mathbf{1}\{\tilde{z}^{(i)} = j\}(\tilde{x}^{(i)} - \mu_j)(\tilde{x}^{(i)} - \mu_j)^T}{\sum_{i=1}^{n} w_j^{(i)} + \alpha \sum_{i=1}^{\tilde{n}} \mathbf{1}\{\tilde{z}^{(i)} = j\}}$$

Now derive the update rules for $\mu^{(t+1)}$ and $\phi^{(t+1)}$. (**Hint**: section 11.4 of the notes would be relevant to this problem.)

Answer:

(d) [5 points] **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.

Run three trials and use the provided plotting function to construct a scatter plot of the resulting assignments to clusters (one plot for each trial). Your plot should indicate cluster assignments with colors they got assigned to (*i.e.,* the cluster which had the highest probability in the final E-step).

Submit the three plots obtained above in your write-up.

Answer:

(e) [7 points] **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.

Create a plot for each trial, as done in the previous sub-question.

Submit the three plots obtained above in your write-up.

Answer:

(f) [3 points] **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.

Answer: