CS 229, Fall 2025
Problem Set #3

Victor Sebastian Martinez Perez (`sebasmp`)

Due Wednesday, November 5 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/87361/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 Wednesday, November 5 at 11:59 pm. If you submit after Wednesday, November 5 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 remember to tag your pages.

All students must submit an electronic PDF version of the written question including plots generated from the codes. We highly recommend typesetting 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 solution independently, and without referring to written notes from the joint session. Each student must understand the solution well enough in order to reconstruct it by him/herself. It is an honor code violation to copy, refer to, or look at written or code solutions from a previous year, including but not limited to: official solutions from a previous year, solutions posted online, and solutions you or someone else may have written up in a previous year. Furthermore, it is an honor code violation to post your assignment solutions online, such as on a public git repo. We run plagiarism-detection software on your code against past solutions as well as student submissions from previous years. Please take the time to familiarize yourself with the Stanford Honor Code[1] and the Stanford Honor Code[2] as it pertains to CS courses.

[1] https://communitystandards.stanford.edu/policies-and-guidance/honor-code
[2] https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1164/handouts/honor-code.pdf

1. **[25 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 $\ell_{\text{semi-sup}}(\theta^{(t+1)}) \geq \ell_{\text{semi-sup}}(\theta^{(t)})$. **Hint:** Use Jensen's inequality to show that the E-step constructs a lower bound on the unsupervised log-likelihood $\ell_{\text{unsup}}(\theta)$, and that the M-step maximizes this bound.

Answer:

Lower bound on unlabeled data

For one example i,

$$\log p(x^{(i)};\theta) = \log \sum_{z^{(i)}} p(x^{(i)}, z^{(i)};\theta)$$

Let $Q_i(z^{(i)})$ be any distribution over $z^{(i)}$ such that $\sum_z Q(z) = 1$ and $Q(z) \geq 0$. Then

$$\log p(x^{(i)};\theta) = \log \sum_{z^{(i)}} Q_i(z^{(i)}) \frac{p(x^{(i)}, z^{(i)};\theta)}{Q_i(z^{(i)})}$$

By Jensen's inequality (since $\log$ is concave),

$$\log \sum_{z^{(i)}} Q_i(z^{(i)}) \frac{p(x^{(i)}, z^{(i)};\theta)}{Q_i(z^{(i)})} \geq \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)};\theta)}{Q_i(z^{(i)})} := \text{ELBO}(x^{(i)}; Q_i, \theta)$$

Hence for all Q_i, ELBO gives a lower-bound on $\log p(x;\theta)$:

$$\log p(x^{(i)};\theta) \geq \text{ELBO}(x^{(i)}; Q_i, \theta)$$

$$\ell_{\text{unsup}}(\theta) = \sum_{i=1}^{n} \log p(x^{(i)};\theta) \;\geq\; \sum_{i=1}^{n} \text{ELBO}(x^{(i)}; Q_i, \theta).$$

E-Step (for unlabeled data)

Set Q_i to be the posterior distribution of $z^{(i)}$ given $x^{(i)}$ and current $\theta^{(t)}$:

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

This choice of $Q_i^{(t)}$ gives the tightest lower bound, as attained by equality:

$$\log p(x^{(i)};\theta^{(t)}) = \text{ELBO}(x^{(i)}; Q_i^{(t)}, \theta^{(t)})$$

For labeled examples $(\tilde{x}^{(i)}, \tilde{z}^{(i)})$, the latent variable is known and no Q_i is introduced.

M-Step

Maximize the ELBO w.r.t. θ for fixed $Q_i^{(t)}$:

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

Monotonic convergence

As each iteration tightens and then maximizes the bound,

$$\begin{aligned}\ell_{\text{semi}}(\theta^{(t+1)}) &= \ell_{\text{unsup}}(\theta^{(t+1)}) + \alpha\,\ell_{\text{sup}}(\theta^{(t+1)}) \\ &\geq \sum_{i=1}^{n} \text{ELBO}(x^{(i)}; Q_i^{(t)}, \theta^{(t+1)}) + \alpha\,\ell_{\text{sup}}(\theta^{(t+1)}) \\ &\geq \sum_{i=1}^{n} \text{ELBO}(x^{(i)}; Q_i^{(t)}, \theta^{(t)}) + \alpha\,\ell_{\text{sup}}(\theta^{(t)}) \\ &= \ell_{\text{unsup}}(\theta^{(t)}) + \alpha\ell_{\text{sup}}(\theta^{(t)}) = \ell_{\text{semi}}(\theta^{(t)})\end{aligned}$$

- First inequality: ELBO is a lower bound on each $\log p(x^{(i)}; \theta^{(t+1)})$.
- Second inequality: $\theta^{(t+1)}$ is chosen from the M-step.
- Last equality: tightness at $\theta^{(t)}$.

Thus, the EM algorithm produces a sequence of parameters such that the log-likelihood $\ell_{\text{semi}}(\theta^{(t)})$ is non-decreasing with each iteration.

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 $x^{(i)}|z^{(i)} \sim \mathcal{N}(\mu_{z^{(i)}}, \Sigma_{z^{(i)}})$ i.i.d. So, μ, Σ, and ϕ are the model parameters.

We also have additional $\tilde{n}$ data points $\tilde{x}^{(i)} \in \mathbb{R}^d, i \in \{1, \ldots, \tilde{n}\}$, 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 $\tilde{x}^{(i)}|\tilde{z}^{(i)} \sim \mathcal{N}(\mu_{\tilde{z}^{(i)}}, \Sigma_{\tilde{z}^{(i)}})$ 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 explicit GMM E-step for the semi-supervised setting 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:

Calculate soft guesses for latent variables $z^{(i)}$ just for the unlabeled data points

$$\begin{aligned} w_j^{(i)} &= p(z^{(i)} = j | x^{(i)}; \phi_j, \mu_j, \Sigma_j) \\ &= \frac{p(x^{(i)} | z^{(i)} = j; \phi_j, \mu_j, \Sigma_j)\, p(z^{(i)} = j; \phi_j, \mu_j, \Sigma_j)}{\sum_{l=1}^{k} p(x^{(i)} | z^{(i)} = l; \phi_j, \mu_j, \Sigma_j)\, p(z^{(i)} = l; \phi_j, \mu_j, \Sigma_j)} \\ &= \frac{\phi_j \, \mathcal{N}(x^{(i)} \mid \mu_j, \Sigma_j)}{\sum_{\ell=1}^{k} \phi_\ell \, \mathcal{N}(x^{(i)} \mid \mu_\ell, \Sigma_\ell)} \end{aligned}$$

where $\mathcal{N}(x^{(i)} \mid \mu_j, \Sigma_j)$ is the Gaussian density function for component j evaluated at $x^{(i)}$

$$\mathcal{N}(x^{(i)} \mid \mu_j, \Sigma_j) = \frac{1}{(2\pi)^{d/2} |\Sigma_j|^{1/2}} \exp\left(-\frac{1}{2} (x^{(i)} - \mu_j)^T \Sigma_j^{-1} (x^{(i)} - \mu_j) \right)$$

Semi-supervised M-Step.

We have provided the 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:

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

(c) [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:

(d) [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:

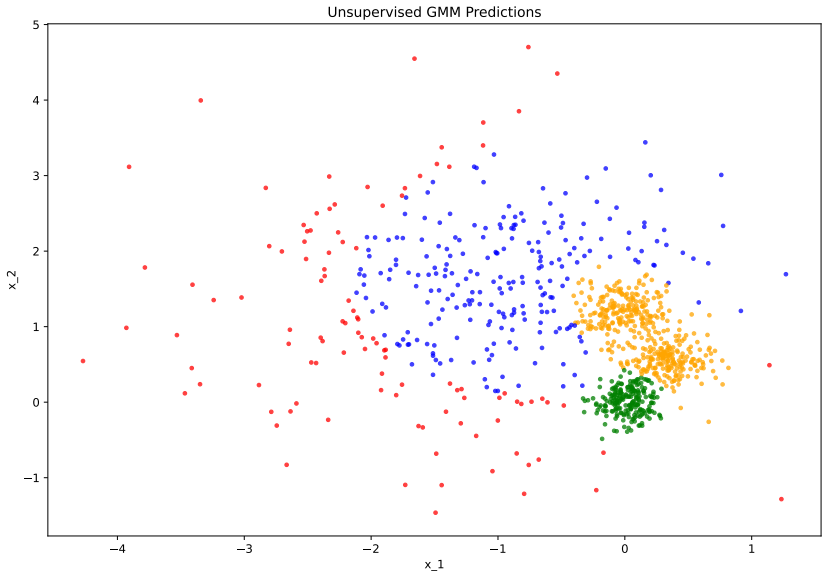

Figure 1: Unsupervised EM Pred 0

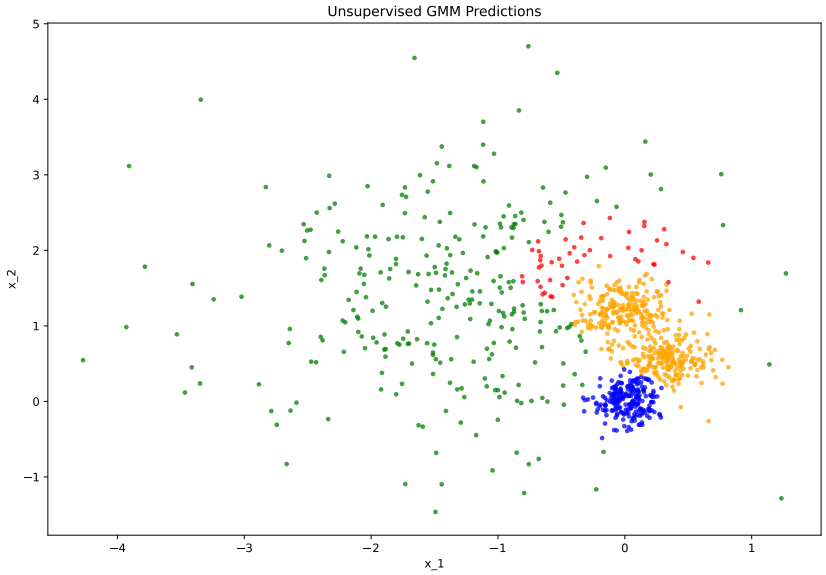

Figure 2: Unsupervised EM Pred 1

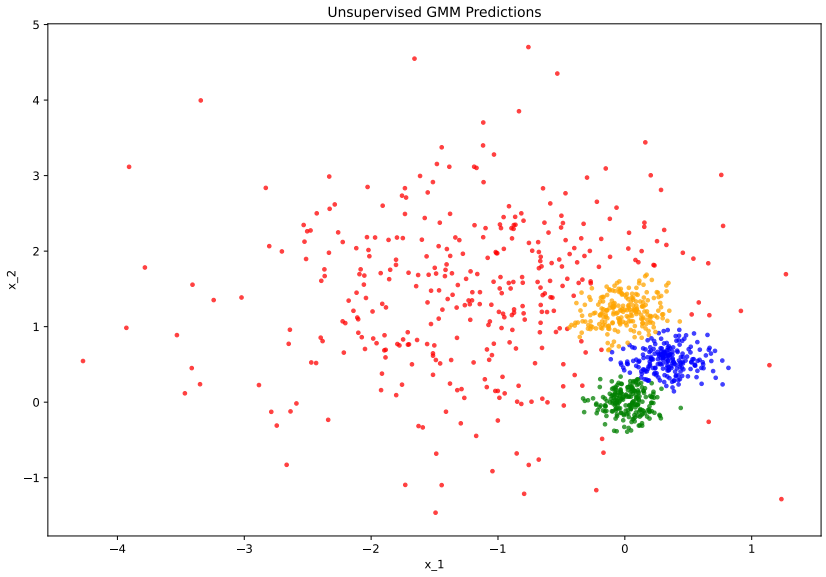

Figure 3: Unsupervised EM Pred 2

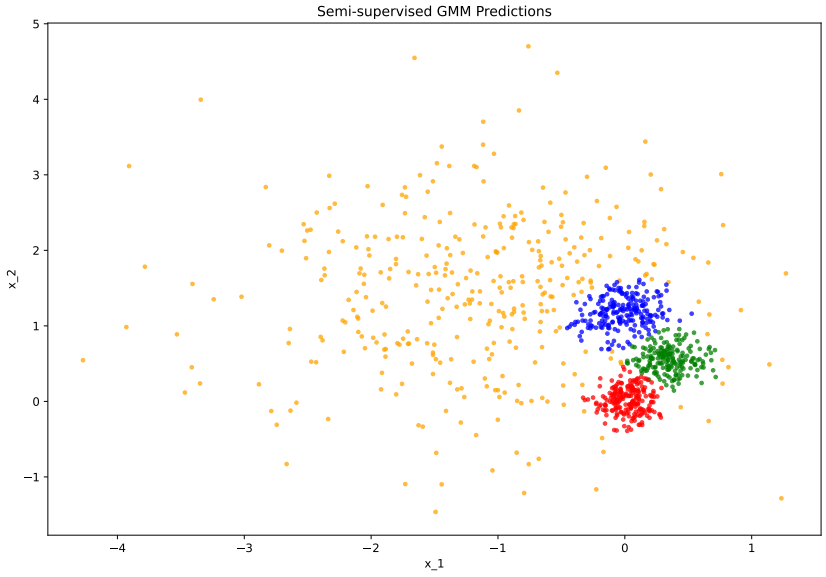

Figure 4: Semi-supervised EM predicion 0

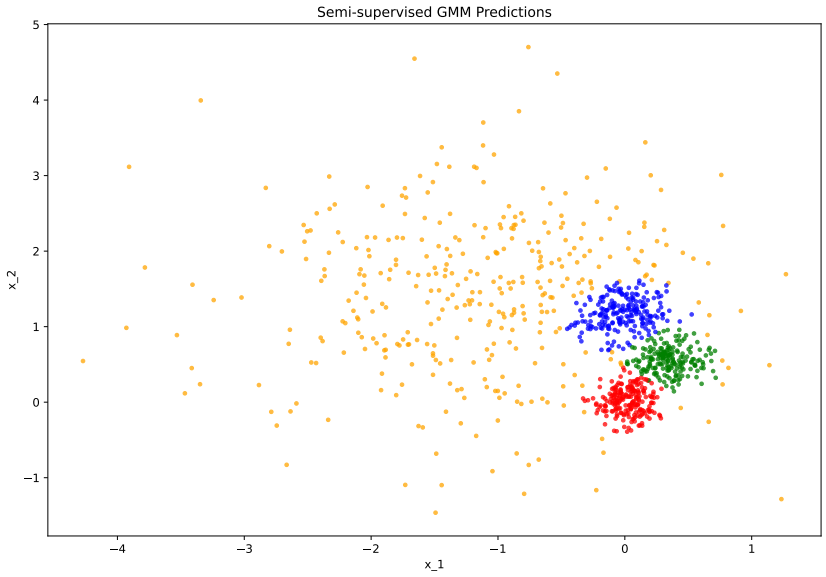

Figure 5: Semi-supervised EM predicion 1

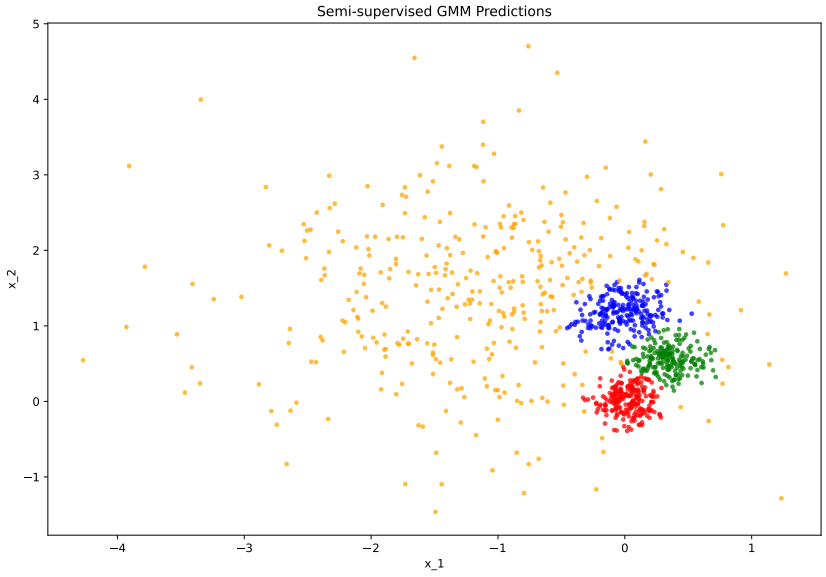

Figure 6: Semi-supervised EM predicion 2

(e) [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:

i. The semi-supervised EM converged in fewer iterations (≈ 23) than unsupervised EM, since the labeled data guide the parameter estimates from the start. The unsupervised EM required more iterations to reach a stable log-likelihood (≈ 100).

ii. The semi-supervised EM was more stable: cluster assignments remained consistent across random initializations (Fig. 4, 5, 6), while the unsupervised EM swapped labels or converged to different local optima (Fig. 1, 2, 3).

iii. The semi-supervised EM achieved better clustering quality. It correctly identified the three compact Gaussian components (red, blue, green) and the single high-variance one (yellow), whereas the unsupervised EM sometimes merged small clusters or split the large one. Overall, labeled data made the assignments more accurate and interpretable.

2. **[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:

For K classes, the class proportions vector in region R_m is:

$$\vec{p}_m = \begin{bmatrix} p_{m1} & p_{m2} & \dots & p_{mK} \end{bmatrix}, \quad \text{where } p_{mk} = \frac{\#\text{ samples of class } k}{|R_m|}, \quad \sum_{k=1}^{K} p_{mk} = 1.$$

The Hessian of $G(\vec{p})$ is:

$$\nabla^2 G(\vec{p}) = -2\mathbb{I}_k$$

Meaning that the Gini loss function is strictly concave in $\vec{p}_m$

Recalling the difinition of the expectation for a discrete random variable:

$$\mathbb{E}[X] = \sum_i P(X = x_i)\, x_i$$

which represents the weighted mean of the possible values x_i.

For the child proportion vectors $\vec{p}_1$ and $\vec{p}_2$, with probabilities (weights) t and $1-t$ respectively (of being selected), we have:

$$\mathbb{E}[X] = t\vec{p}_1 + (1-t)\vec{p}_2.$$

Then applying Jensen's inequality for concave functions, we have:

$$G(\mathbb{E}[X]) \geq \mathbb{E}[G(X)]$$
$$G(t\vec{p}_1 + (1-t)\vec{p}_2) \geq tG(\vec{p}_1) + (1-t)G(\vec{p}_2).$$

Since the Gini Loss is strictly concave, the inequality is strict whenever $\vec{p}_1 \neq \vec{p}_2$:

$$G(t\vec{p}_1 + (1-t)\vec{p}_2) > tG(\vec{p}_1) + (1-t)G(\vec{p}_2).$$

(b) [5 points] List out the conditions on the cardinality and proportion of positive and negative samples on the children 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:

The Gini loss remains unchanged after a split in two cases:

Case 1: One child node is empty ($t = 0$ or $t = 1$). Then:

$$tG(\vec{p}_1) + (1-t)G(\vec{p}_2) = 0 \cdot G(\vec{p}_1) + G(\vec{p}_2) = G(\vec{p}),$$

Case 2: Both child nodes have the same class proportions as the parent node ($\vec{p}_1 = \vec{p}_2 = p$):

$$tG(\vec{p}_1) + (1-t)G(\vec{p}_2) = tG(\vec{p}) + (1-t)G(\vec{p}) = G(\vec{p}),$$

which is equal to the Gini loss before the split in both cases.

These cases do not contradict the strict concavity of the Gini loss, since concavity holds when the two child nodes differ in class proportions, and both cases above are the allowed equality conditions of concavity.

Furthermore, these cases do not prevent a fully grown tree from achivieng zero Gini loss. They may temporarilly stall the reduction at certain splits, but subsequent splits can still lead to pure nodes with zero Gini loss.

(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:

The misclassification loss remains unchanged when the parent's majority class is the same for the child nodes.

The misclassification loss in region R_m is given by:

$$M(R_m) = 1 - \frac{\max_k N_{mk}}{N_m}.$$

After a split into two child nodes R_1 and R_2, the misclassification loss is:

$$M(R_1, R_2) = \frac{N_1}{N_m}\left(1 - \frac{\max_k N_{1k}}{N_1}\right) + \frac{N_2}{N_m}\left(1 - \frac{\max_k N_{2k}}{N_2}\right).$$

If both child nodes have the same majority class as the parent node, then:

$$\max_k N_{1k} + \max_k N_{2k} = \max_k N_{mk}.$$

The misclassification loss for the child nodes is then:

$$\begin{aligned} M(R_1, R_2) &= \frac{N_1}{N_m}\left(1 - \frac{\max_k N_{1k}}{N_1}\right) + \frac{N_2}{N_m}\left(1 - \frac{\max_k N_{2k}}{N_2}\right) \\ &= \frac{N_1 + N_2}{N_m} - \frac{\max_k N_{mk}}{N_m} \\ &= 1 - \frac{\max_k N_{mk}}{N_m} = M(R_m) \end{aligned}$$

Thus, the misclassification loss remains unchanged after the split when both child nodes share the same majority class as the parent node.

(d) [4 points] 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:
The variance of the sum of random variables and the scaling property are:

$$\operatorname{Var}\left(\sum_i X_i\right) = \sum_i \operatorname{Var}(X_i) + 2\sum_{i<j} \operatorname{Cov}(X_i, X_j), \qquad \operatorname{Var}(cX) = c^2 \operatorname{Var}(X).$$

Covariance is defined as:

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

Since all trees $T_1, T_2, \ldots, T_B$ share the same variance σ^2 and pairwise correlation ρ,

$$\mathrm{Cov}(T_i, T_j) = \rho\,\sigma^2 \quad \text{for } i \neq j.$$

Then, the variance of the average prediction is:

$$\begin{aligned}
\mathrm{Var}\left(\frac{1}{B}\sum_{i=1}^{B} T_i\right) &= \frac{1}{B^2}\left(\sum_{i=1}^{B}\mathrm{Var}(T_i) + 2\sum_{i<j}\mathrm{Cov}(T_i, T_j)\right) \\
&= \frac{1}{B^2}\left(B\sigma^2 + 2\cdot\frac{B(B-1)}{2}\rho\sigma^2\right) \\
&= \frac{1}{B^2}\left(B\sigma^2 + B(B-1)\rho\sigma^2\right) \\
&= \frac{\sigma^2}{B}\big(1 + (B-1)\rho\big) \\
&= \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.
\end{aligned}$$

3. **[20 points] AdaBoost Performance**

We learned about boosting in lecture. Statistician Kevin Murphy claims that "It can be shown that, as long as each base learner has an accuracy that is better than chance (even on the weighted dataset), then the final ensemble of classifiers will have higher accuracy than any given component." We will now verify this in the AdaBoost framework.

(a) [3 points] Given a set of n observations (x_i, y_i) where y_i is the label $y_i \in \{-1, 1\}$, let $f_t(x)$ be the weak classifier at step t and let $\hat{w}_t$ be its weight. First we note that the final classifier after T steps is defined as

$$F(x) = \text{sign}\left\{\sum_{t=1}^{T} \hat{w}_t f_t(x)\right\} = \text{sign}\{f(x)\},$$

where

$$f(x) = \sum_{t=1}^{T} \hat{w}_t f_t(x).$$

We can assume that $f(x)$ is never exactly zero.
Show that

$$\varepsilon_{\text{training}} := \frac{1}{n}\sum_{i=1}^{n} 1_{\{F(x_i)\neq y_i\}} \leq \frac{1}{n}\sum_{i=1}^{n} \exp(-f(x_i)y_i),$$

where $1_{\{F(x_i)\neq y_i\}}$ is 1 if $F(x_i) \neq y_i$ and 0 otherwise.
Answer:

0-1 indicator function

For each training example (x_i, y_i), define margin $t_i := y_i f(x_i)$. Then $t_i > 0$ when x_i is correctly classified, and $t_i < 0$ when it is misclassified. Thus the misclassification indicator can be written as

$$\mathbf{1}\{F(x_i) \neq y_i\} = \mathbf{1}\{t_i < 0\}.$$

Upper bound on 0-1 loss

Consider two cases:

- If $t_i < 0$ (misclassification), then $\mathbb{1}[t_i < 0] = 1$ and $\exp(-t_i) > 1$
- If $t_i \geq 0$ (correct), then $\mathbb{1}[t_i < 0] = 0$ and $\exp(-t_i) \geq 0$

Therefore, in both cases, we have:

$$\mathbb{1}[t_i < 0] \leq \exp(-t_i)$$

Overall loss bound

Averaging over the dataset,

$$\frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\{F(x_i) \neq y_i\} \;\leq\; \frac{1}{n}\sum_{i=1}^{n} e^{-\,y_i f(x_i)}.$$

(b) [8 points] The weight for each data point i at step $t+1$ can be defined recursively by

$$\alpha_{i,(t+1)} = \frac{\alpha_{i,t} \exp(-\hat{w}_t f_t(x_i) y_i)}{Z_t},$$

where Z_t is a normalizing constant ensuring the weights sum to 1

$$Z_t = \sum_{i=1}^{n} \alpha_{i,t} \exp(-\hat{w}_t f_t(x_i) y_i).$$

Show that

$$\frac{1}{n} \sum_{i=1}^{n} \exp(-f(x_i) y_i) = \prod_{t=1}^{T} Z_t.$$

Answer:

For a single example x_i, y_i

$$\begin{aligned}
\exp(-f(x_i)y_i) &= \exp(-\sum_{t=1}^{T} \tilde{w}_t f_t(x_i) y_i) \\
&= \exp(-\tilde{w}_1 f_1(x_i) y_i) \cdot \exp(-\tilde{w}_2 f_2(x_i) y_i) \ldots \exp(-\tilde{w}_T f_T(x_i) y_i) \\
&= \prod_{t=1}^{T} \exp(-\tilde{w}_t f_t(x_i) y_i)
\end{aligned}$$

For the whole dataset

$$\frac{1}{n} \sum_{i=1}^{n} \exp(-f(x_i) y_i) = \frac{1}{n} \sum_{i=1}^{n} \prod_{t=1}^{T} \exp(-\tilde{w}_t f_t(x_i) y_i)$$

Since $\alpha_{i,1} = \frac{1}{n}$, then

$$\frac{1}{n} \sum_{i=1}^{n} \exp(-f(x_i) y_i) = \sum_{i=1}^{n} \alpha_{i,1} \prod_{t=1}^{T} \exp(-\tilde{w}_t f_t(x_i) y_i)$$

For $t = 1$:

$$\sum_{i=1}^{n} \alpha_{i,1} \exp(-\tilde{w}_1 f_1(x_i) y_i) \prod_{t=1}^{T} \exp(-\tilde{w}_t f_t(x_i) y_i) = Z_1 \sum_{i=1}^{n} \underbrace{\frac{\alpha_{i,1} \exp(-\tilde{w}_1 f_1(x_i) y_i)}{Z_1}}_{\alpha_{i,2}} \prod_{t=2}^{T} \exp(-\tilde{w}_t f_t(x_i) y_i)$$

For $t = 2$:

$$= Z_1 Z_2 \sum_{i=1}^{n} \underbrace{\frac{\alpha_{i,2} \exp(-\tilde{w}_2 f_2(x_i) y_i)}{Z_2}}_{\alpha_{i},3} \prod_{t=3}^{T} \exp(-\tilde{w}_t f_t(x_i) y_i)$$

For $t = 3, \ldots, T$, and considering $\sum_{i=1}^{n} \alpha_{i,t} = 1$

$$= \prod_{t=1}^{T} Z_t \sum_{i=1}^{n} \alpha_{i,t+1} = \prod_{t=1}^{T} Z_t$$

Finally,

$$\boxed{\frac{1}{n}\sum_{i=1}^{n}\exp(-f(x_i)y_i) = \prod_{t=1}^{T} Z_t}$$

(c) [9 points] We showed above that training error is bounded above by $\prod_{t=1}^{T} Z_t$. At step t the values Z_1, Z_2, ..., Z_{t-1} are already fixed therefore at step t we can choose α_t to minimize Z_t. Let

$$\varepsilon_t = \sum_{i=1}^{n} \alpha_{i,t} 1_{\{f_t(x_i)\neq y_i\}}$$

be the weighted training error for the weak classifier $f_t(x)$. Then we can re-write the formula for Z_t as

$$Z_t = (1-\varepsilon_t)\exp(-\hat{w}_t) + \varepsilon_t \exp(\hat{w}_t).$$

(i) [3 points] First find the value of $\hat{w}_t$ that minimizes Z_t. Then show that the corresponding optimal value is

$$Z_t^{\text{opt}} = 2\sqrt{\varepsilon_t(1-\varepsilon_t)}.$$

(ii) [3 points] Assume we choose Z_t as Z_t^{opt} in part c (i). Then re-write $\varepsilon_t = 1/2 - \gamma_t$, where $\gamma_t > 0$ implies better than random and $\gamma_t < 0$ implies worse than random. Then show that

$$Z_t \leq \exp(-2\gamma_t^2).$$

(You may want to use the fact that $\log(1-x) \leq -x$ for $0 \leq x < 1$.)

(iii) [3 points] Finally, show that if each classifier is better than random, i.e., $\gamma_t > \gamma$ for all t and $\gamma > 0$, then

$$\varepsilon_{\text{training}} \leq \exp(-2T\gamma^2),$$

which shows that the training error can be made arbitrarily small with enough steps.

Answer:

(i) Differentiate and set to zero

$$\frac{d}{d\tilde{w}} Z_t = 0 = -(1-\varepsilon)\exp(-\tilde{w}_t) + \varepsilon_t \exp(\tilde{w}_t)$$

Then,

$$\tilde{w}_t^* = \frac{1}{2}\log(\frac{1-\varepsilon_t}{\varepsilon_t})$$

So,

$$Z_t^{\text{opt}} = (1-\varepsilon)\sqrt{\frac{\varepsilon}{1-\varepsilon}} + \varepsilon_t\sqrt{\frac{1-\varepsilon}{\varepsilon}} = \boxed{2\sqrt{\varepsilon_t(1-\varepsilon_t)}}$$

(ii)

$$Z_t^{\text{opt}} = 2\sqrt{\varepsilon_t(1-\varepsilon_t)} = 2\sqrt{\left(\tfrac{1}{2}-\gamma_t\right)\left(\tfrac{1}{2}+\gamma_t\right)} = 2\sqrt{\tfrac{1}{4}-\gamma_t^2} = \sqrt{1-4\gamma_t^2}.$$

Taking logarithms and using the inequality $\log(1-x) \leq -x$ for $0 \leq x < 1$:

$$\log Z_t^{\text{opt}} = \tfrac{1}{2}\log(1-4\gamma_t^2) \leq \tfrac{1}{2}(-4\gamma_t^2) = -2\gamma_t^2.$$

Exponentiating both sides gives

$$\boxed{Z_t^{\text{opt}} \leq e^{-2\gamma_t^2}.}$$

(iii) If each weak classifier is better than random, i.e. $\gamma_t \geq \gamma > 0$ for all t, then

$$\varepsilon_{\text{training}} \leq \prod_{t=1}^{T} Z_t \leq \prod_{t=1}^{T} e^{-2\gamma_t^2} = e^{-2\sum_{t=1}^{T}\gamma_t^2} \leq e^{-2T\gamma^2}.$$

Thus

$$\boxed{\varepsilon_{\text{training}} \leq e^{-2T\gamma^2},}$$

4. **[25 points] A Simple Neural Network**

Let $X = \{x^{(1)}, \cdots, x^{(n)}\}$ be a dataset of n samples with 2 features, i.e $x^{(i)} \in \mathbb{R}^2$. The samples are classified into 2 categories with labels $y^{(i)} \in \{0, 1\}$. A scatter plot of the dataset is shown in Figure **??**:

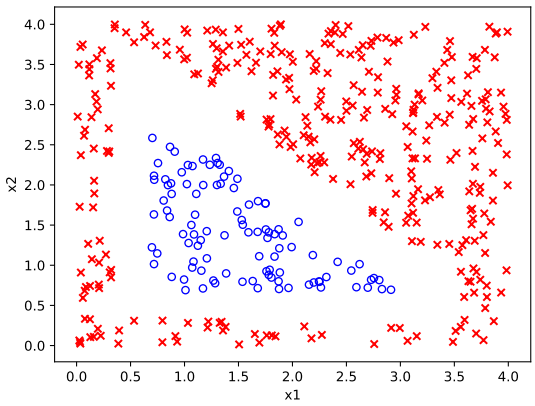

Figure 7: Plot of dataset X.

The examples in class 1 are marked as as "×" and examples in class 0 are marked as "○". We want to perform binary classification using a simple neural network with the architecture shown in Figure **??**:

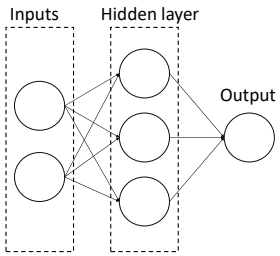

Figure 8: Architecture for our simple neural network.

Denote the two features x_1 and x_2, the three neurons in the hidden layer h_1, h_2, and h_3, and the output neuron as o. Let the weight from x_i to h_j be $w_{i,j}^{[1]}$ for $i \in \{1, 2\}, j \in \{1, 2, 3\}$, and the weight from h_j to o be $w_j^{[2]}$. Finally, denote the intercept weight for h_j as $w_{0,j}^{[1]}$, and the intercept weight for o as $w_0^{[2]}$. For the loss function, we'll use average squared loss instead of the usual negative log-likelihood:

$$l = \frac{1}{n} \sum_{i=1}^{n} \left(o^{(i)} - y^{(i)} \right)^2,$$

where $o^{(i)}$ is the result of the output neuron for example i.

(a) [5 points] Suppose we use the sigmoid function as the activation function for h_1, h_2, h_3 and o. What is the gradient descent update to $w_{1,2}^{[1]}$, assuming we use a learning rate of α? Your answer should be written in terms of $x^{(i)}$, $o^{(i)}$, $y^{(i)}$, and the weights.

Answer:

Let the activations be defined as:

$$a_2^{(i)} = w_{0,2}^{[1]} + w_{1,2}^{[1]} x_1^{(i)} + w_{2,2}^{[1]} x_2^{(i)}, \quad h_2^{(i)} = \sigma(a_2^{(i)}),$$

$$z^{(i)} = w_0^{[2]} + \sum_{j=1}^{3} w_j^{[2]} h_j^{(i)}, \quad o^{(i)} = \sigma(z^{(i)}),$$

where $\sigma(u) = \frac{1}{1+e^{-u}}$.

The loss is

$$\ell = \frac{1}{n} \sum_{i=1}^{n} (o^{(i)} - y^{(i)})^2.$$

Using backpropagation and the fact that $\sigma'(u) = \sigma(u)(1 - \sigma(u))$, we have:

$$\frac{\partial \ell}{\partial w_{1,2}^{[1]}} = \frac{2}{n} \sum_{i=1}^{n} (o^{(i)} - y^{(i)})\, o^{(i)}(1 - o^{(i)})\, w_2^{[2]}\, h_2^{(i)}(1 - h_2^{(i)})\, x_1^{(i)}.$$

The gradient descent update (with learning rate α) is therefore:

$$\boxed{w_{1,2}^{[1]} \leftarrow w_{1,2}^{[1]} - \alpha \cdot \frac{2}{n} \sum_{i=1}^{n} (o^{(i)} - y^{(i)})\, o^{(i)}(1 - o^{(i)})\, w_2^{[2]}\, h_2^{(i)}(1 - h_2^{(i)})\, x_1^{(i)}}.$$

(b) [10 points] Now, suppose instead of using the sigmoid function for the activation function for h_1, h_2, h_3 and o, we instead used the step function $f(x)$, defined as

$$f(x) = \begin{cases} 1, x > 0 \\ 0, x \leq 0 \end{cases}$$

Is it possible to have a set of weights that allow the neural network to classify this dataset with 100% accuracy?

If you believe it's possible, please implement your approach by completing the `optimal_step_weights` method in `src/simple_nn/simple_nn.py` and including the corresponding `step_weights.pdf` plot showing perfect prediction in your writeup.

If it is not possible, please explain your reasoning in the writeup.

Hint 1: There are three sides to a triangle, and there are three neurons in the hidden layer.

Hint 2: A solution can be found where all weight and bias parameters take values only in $\{-1, -0.5, 0, 1, 3, 4\}$. You are free to come up with other solutions as well.

Answer:

If each hidden unit uses a step activation:

$$h_j(x) = \mathbb{1}\{\, w_{1,j}^{[1]} x_1 + w_{2,j}^{[1]} x_2 + w_{0,j}^{[1]} > 0 \,\}, \qquad j = 1, 2, 3,$$

each one represents a linear inequality that defines one side of the triangle.

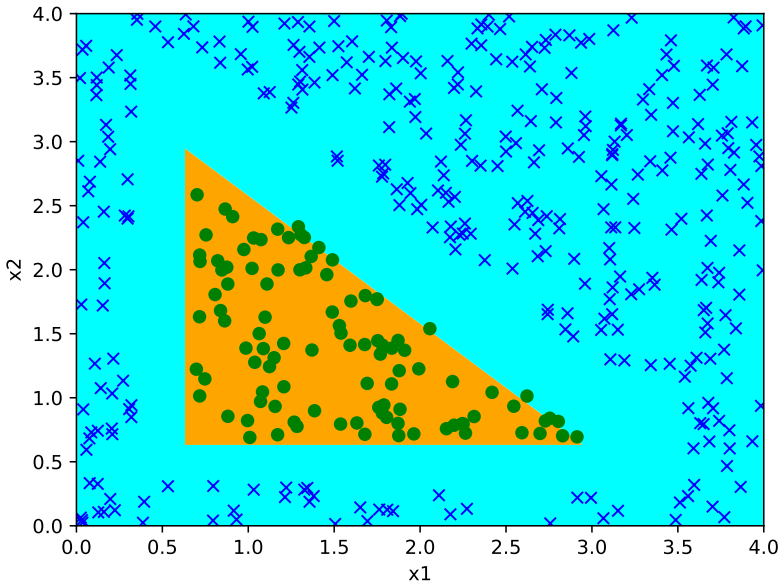

Figure 9: Step Weights

By setting the hidden weights and biases so that these three lines match the triangle's edges, with normal vectors pointing inward, points inside the triangle satisfy all three inequalities, producing $h_1 = h_2 = h_3 = 1$. Points outside the triangle violate at least one of them.
The output neuron can then implement a logical AND:

$$o(x) = \mathbb{1}\{\, w_1^{[2]} h_1 + w_2^{[2]} h_2 + w_3^{[2]} h_3 + w_0^{[2]} > 0 \,\},$$

for example with $w_1^{[2]} = w_2^{[2]} = w_3^{[2]} = 1$, and $w_0^{[2]} = -2.5$.
Then, the network outputs 1 if and only if all three hidden units are active. Hence, the network can achieve 100% training accuracy.
In `src/simple_nn/simple_nn.py` the step activations are:

$$h_1 = \mathbb{1}\{-0.5 + x_1 \geq 0\}, \quad h_2 = \mathbb{1}\{-0.5 + x_2 \geq 0\}, \quad h_3 = \mathbb{1}\{3.75 - x_1 - x_2 \geq 0\}.$$

Since the dataset labels the exterior as 1 (mean$(y) \approx 0.77$), the output implements NOT-AND:

$$o = \mathbb{1}\{\, 2.5 - h_1 - h_2 - h_3 > 0 \,\} = 1 - \mathbb{1}\{h_1 = h_2 = h_3 = 1\},$$

yielding perfect accuracy. Fig. 9 shows the plot with the perfect prediction.

(c) [10 points] Let the activation functions for h_1, h_2, h_3 be the linear function $f(x) = x$ and the activation function for o be the same step function as before.
Is it possible to have a set of weights that allow the neural network to classify this dataset with 100% accuracy?
If you believe it's possible, please implement your approach by completing the `optimal_linear_weights` method in `src/simple_nn/simple_nn.py` and including the corresponding `linear_weights.pdf` plot showing perfect prediction in your writeup.

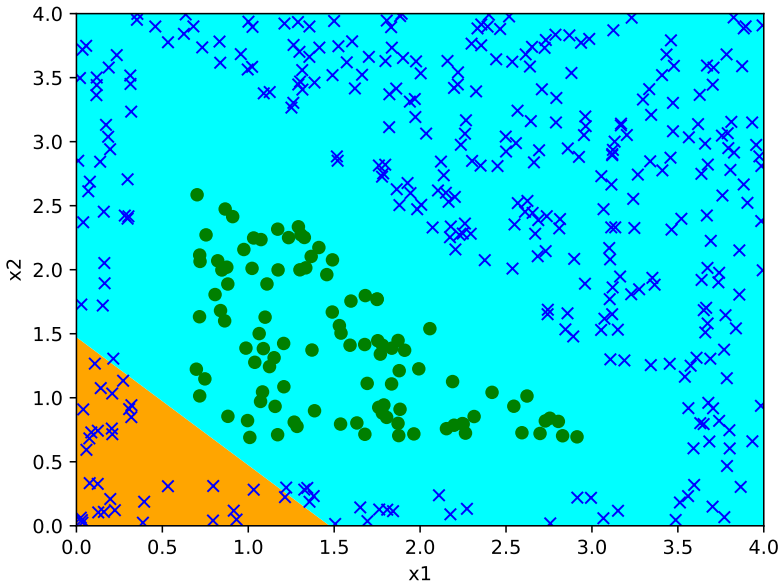

Figure 10: Linear Weights

If it is not possible, please explain your reasoning in the writeup.

Hint: The hints from the previous sub-question might or might not apply.

Answer:

If the hidden layer is linear,

$$h_j(x) = w_{1,j}^{[1]} x_1 + w_{2,j}^{[1]} x_2 + w_{0,j}^{[1]},$$

and the output is a step function:

$$o(x) = \mathbb{1}\{\, w_0^{[2]} + \sum_{j=1}^{3} w_j^{[2]} h_j(x) > 0 \,\},$$

then the overall decision boundary is given by a single linear inequality

$$o(x) = \mathbb{1}\{\, b + v^\top x > 0 \,\}$$

This corresponds to a single straight line in the input space, so the model can only separate the data with one half-plane. Because the dataset is defined by a triangular region, which requires three boundaries, such a network cannot perfectly classify all points. So a linear hidden layer followed by a single step output is insufficient to achieve perfect accuracy.

In `src/simple_nn/simple_nn.py`, with linear hidden units, the network reduces to $o(x) = \mathbb{1}\{b + v^\top x > 0\}$. A single line cannot realize the triangular decision set, so perfect classification is impossible; we obtain ≈ 0.701 in the example. Fig. 10 shows the plot with the prediction.