CS 446 / ECE 449 — Homework 5

your NetID here

Instructions.

- Homework is due **Friday, November 14**, at 11:59 **AM** CST; you have **3** late days in total for **all Homeworks**.
- The template for coding problems are available at this link.
- Everyone must submit individually at gradescope under `HW5` and `HW5 - Programming Assignment`.
- The "written" submission at `HW5` **must be typed**, and submitted in any format gradescope accepts (to be safe, submit a PDF). You may use LaTeX, markdown, google docs, MS word, whatever you like; but it must be typed!
- When submitting at `HW5`, Gradescope will ask you to **mark out boxes around each of your answers**; please do this precisely!
- Please make sure your NetID is clear and large on the first page of the homework.
- Your solution **must** be written in your own words. Please see the course webpage for full **academic integrity** information. You should cite any external reference you use. The use of LLM is **not** allowed.
- We reserve the right to reduce the auto-graded score for `HW5 - Programming Assignment` if we detect funny business (e.g., your solution lacks any algorithm and hard-codes answers you obtained from someone else, or simply via trial-and-error with the autograder).
- When submitting to `HW5 - Programming Assignment`, only upload `hw5_q1.py`, `hw5_q3.py`. Additional files will be ignored.

1. Decision Tree (40 pt)

Consider $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=0}^{N-1}$ where $N \in \mathbb{N}$ and, for $i \in \{0, \ldots, N-1\}$, $\boldsymbol{x}^{(i)} \in \mathbb{R}^D$ and $y^{(i)} \in \{0, \ldots, C-1\}$ for $D, C \in \mathbb{N}$ (note: here, we adopt the convention that $0 \notin \mathbb{N}$). In words, we have a training dataset of N data points, each with D *continuous* features and belonging to exactly one of C classes. We wish to train a decision tree classifier using this dataset.

We briefly discussed the decision trees for general continuous spaces. A continuous-valued decision is represented using the total order $\leq$ on $\mathbb{R}$. Specifically, at each node, we will specify a feature j ($j \in \{0, \ldots, D-1\}$) and also a *threshold* T, and then create *two descendant nodes* at the current node. For all data points in the current node, those with x_j at most the threshold will be put into the *left* child node, labeled as $[x_j \leq T]$, and those with x_j strictly greater than the threshold will be put into the *right* child node, labeled as $[x_j > T]$.

[Programming] (16 pt)

In this part of the question, you will implement the training and inference of a decision tree classifier on continuous features. We have provided a template of the code in `hw5_q1.py`. Your task is to fill out all the `TODO`s in the methods of the `DecisionTreeClassifier` class.

We provide below a description of methods related to the training of the decision tree classifier.

- `fit` (no `TODO`): Given a training dataset, build the decision tree classifier. This is done by calling the `_grow_tree` method on the entire training dataset.
- `_grow_tree`: Recursively grow the decision tree at a given node (holding a *subset* of the training dataset). The depth of the node is being tracked to avoid growing too large a tree.
- `_find_best_split`: Given a subset of the training dataset, find the split (feature and threshold) that results in the highest information gain. Here, the candidate thresholds for a feature are all possible values of that feature in the subset.
- `_information_gain`: Compute the information gain of a split (feature and threshold). At a parent node, there are N_S samples and the class distribution is $\mathbf{p}$. A split partitions this node into a left child with N_L samples and class distribution $\mathbf{p}_L$ and a right child with $N_R (= N_S - N_L)$ samples and distribution $\mathbf{p}_R$. The information gain is defined as:

  $$IG(N_S, \mathbf{p}, N_L, \mathbf{p}_L, N_R, \mathbf{p}_R) = H(\mathbf{p}) - \left(\frac{N_L}{N_S} H(\mathbf{p}_L) + \frac{N_R}{N_S} H(\mathbf{p}_R)\right)$$

  where H is the impurity function.
- `_split`: Perform the split on a subset of the dataset by identifying all the indices corresponding to each of the two parts separated based on the specified feature and threshold.
- `_gini`: Given a subset of the labels, compute the Gini impurity. The lecture discussed the use of sample entropy as the impurity measure. Here, we introduced a different impurity measure: Gini impurity, with the following formula

  $$\text{Gini}(\mathbf{p}) = 1 - \sum_{i=0}^{C-1} p_i^2$$

  where $\mathbf{p} \in [0, 1]^C$ holds the class distribution in the given subset of labels.
- `_most_common_label`: Given a subset of the labels, find the most common one. If there are many common labels, return the smallest index.

We provide below a description of methods related to the inference of the decision tree classifier.

- `predict` (no `TODO`): Given query samples, make the prediction using the fitted decision tree classifier. This is done by calling the `_traverse_tree` method on each query sample.
- `_traverse_tree`: Given a query sample, recursively traverse the fitted decision tree based on the split at the current node.

[Written] (24 pt)

Answer the following questions:

(a) An impurity measure must be a concave function for the information gain to be non-negative. In this question, we verify that Gini is a qualified impurity measure and explain why concavity is necessary.

i. Prove that the Gini impurity is a strictly concave function on $\mathbb{R}^C$. **Hint** Show that the Hessian matrix, $\nabla^2 \text{Gini}(\mathbf{p})$, is negative definite. (4 pt)
(Side note This implies that the Gini impurity function is concave on our domain of interest, which is the set of class distributions.)

ii. Prove that information gain is always non-negative when the impurity measure is concave. (4 pt)
Hint Find an identity relating $\mathbf{p}$, $\mathbf{p}_L$, and $\mathbf{p}_R$ using N_S, N_L, N_R. Then, use Jensen's inequality in an appropriate way.

(b) As discussed in the lecture, a good impurity measure should be minimal when a node is "pure" (i.e., all data belongs to a single class) and maximal when a node is "maximally impure" (i.e., data is uniformly distributed among all classes). In this question, we verify that the Gini impurity has this desired property.

Let $\mathbf{p} = (p_0, \ldots, p_{C-1})$ be the distribution of classes in a node, where $p_i \geq 0$ and $\sum_{i=0}^{C-1} p_i = 1$.

i. Prove that $\text{Gini}(\mathbf{p}) = 0$ *if and only if* the node is pure (i.e., $p_k = 1$ for some $k \in \{0, \ldots, C-1\}$ and $p_j = 0$ for all $j \neq k$). (4 pt)

ii. Prove that $\text{Gini}(\mathbf{p})$ is maximized when the class distribution is uniform (i.e., $p_i = 1/C$ for all $i \in \{0, \ldots, C-1\}$). What is this maximum value? **Hint** Use the method of Lagrange multipliers. (4 pt)

(c) In this question, we focus on the computational aspect of the decision tree classifier.

i. What is the time complexity of the `_find_best_split` method in big-O notation with respect to N_S (number of data points in the subset S at the node), D (number of features), and C (number of classes)? Explain your answer. (4 pt)

ii. Propose a different implementation that will get a better time complexity. What is the improved time complexity in big-O notation with respect to N_S, D, and C? Explain your answer. (4 pt)

Solution.

(a-i) So for this part they want the Hessian of the Gini impurity

$$G(p) = 1 - \sum_{i=1}^{c} p_i^2.$$

Honestly this one is pretty quick because the function is basically "negative squared norm plus a constant." When I differentiate w.r.t. p_j I get

$$\frac{\partial G}{\partial p_j} = -2p_j.$$

Then for the second derivative:

$$\frac{\partial^2 G}{\partial p_i \partial p_j} = \begin{cases} -2 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

So the Hessian is literally just $-2I_c$. Since it's a negative multiple of the identity, all eigenvalues are -2, which are obviously negative. So the Hessian is negative definite.

That basically shows G is strictly concave on the interior of the simplex, because the "curvature" is always bending downward no matter what direction you move in.

(a-ii): Here we need to show the hinge function $h(m) = \max(0, 1-m)$ is convex. I kinda think of it as two lines: one is $1 - m$ (which has slope -1) and the other is the constant 0. The hinge just picks the bigger of the two.

A max of convex functions is always convex, and linear functions are convex anyway. So since both pieces (0 and $1 - m$) are convex, the hinge is also convex overall.
To be a bit more formal: $h(m)$ is the pointwise maximum of $f_1(m) = 1 - m$ and $f_2(m) = 0$, and

$$\max(f_1, f_2)$$

is convex whenever f_1 and f_2 are convex.
So yeah — hinge is convex. Nothing tricky here.

(a-iii) We want to show logistic regression training is convex in $\boldsymbol{w}$ when we're doing binary classification with $y \in \{-1, +1\}$. The loss for a single example is

$$\ell(\boldsymbol{w}) \;=\; \log\bigl(1 + e^{-y\,\boldsymbol{w}^\top x}\bigr).$$

I started by letting $t = y\,\boldsymbol{w}^\top x$ just to make the expression cleaner. Then the loss is basically

$$g(t) = \log(1 + e^{-t}),$$

and you're composing g with an affine function of $\boldsymbol{w}$ (since $\boldsymbol{w}^\top x$ is linear in $\boldsymbol{w}$).
Now, $g(t)$ is a standard convex function (it's basically the "softplus" but flipped), and composing a convex function with an affine thing keeps it convex. That means the whole loss is convex in $\boldsymbol{w}$.
If you want a derivative check: $g''(t)$ is always nonnegative (I checked quickly — it comes out to something like

$$g''(t) = \frac{e^{-t}}{(1 + e^{-t})^2} \geq 0,$$

which is the logistic pdf). So g is convex. Therefore the logistic regression objective

$$\sum_{i=1}^{n} \log\bigl(1 + e^{-y_i\,\boldsymbol{w}^\top x_i}\bigr)$$

is a sum of convex things, and sums of convex functions are convex. Done.

2. Ensemble methods (Bagging, AdaBoost) (34 pt)

Answer the following questions:

(a) Fix a training dataset $\mathcal{D}$ of size N for a regression task, a query point $\mathbf{x}$, and a training algorithm $\mathcal{A}$ (i.e., a function which takes a dataset and returns a trained model). We create B bootstrap samples $\mathcal{D}_1, \ldots, \mathcal{D}_B$ by sampling N times with replacement from $\mathcal{D}$. We then train B models, let $h_b = \mathcal{A}(\mathcal{D}_b)$, i.e., h_b is the model trained on $\mathcal{D}_b$. The bagging predictor is the average:

$$H_B(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} h_b(\mathbf{x})$$

Note that the only source of randomness in this setup is the bootstrap sampling process (not the training dataset nor the query point).

Since each learner h_b is trained on a $\mathcal{D}_b$ drawn from the same bootstrap distribution from $\mathcal{D}$, all $h_b(\mathbf{x})$ are identically distributed. Let this variance be

$$\sigma^2 = \mathrm{Var}[h_b(\mathbf{x})]$$

On the other hand, as discussed in the lecture, any pair of random variables $h_i(\mathbf{x})$ and $h_j(\mathbf{x})$ $(i \neq j)$ are not independent because the training sets $\mathcal{D}_i$ and $\mathcal{D}_j$ are correlated (both are drawn from $\mathcal{D}$). However, all pairs have the same covariance since they are identically distributed. Let this covariance be

$$C = \mathrm{Cov}[h_i(\mathbf{x}), h_j(\mathbf{x})]$$

i. Derive a formula for the variance of the bagging predictor, $\mathrm{Var}[H_B(\mathbf{x})]$, that is dependent on B, σ^2, and C. (2 pt)

ii. What does $\mathrm{Var}[H_B(\mathbf{x})]$ converge to as $B \to \infty$? Answer in terms of σ^2 and C. (2 pt)

iii. Suppose $\mathcal{A}$ is unstable, in the sense that $h_i(\mathbf{x}) \neq h_j(\mathbf{x})$ with non-zero probability. Prove that $C < \sigma^2$ and conclude that $\mathrm{Var}[H_B(\mathbf{x})] < \sigma^2$ for any $B > 1$. **Hint** Consider $Y = h_i(\mathbf{x}) - h_j(\mathbf{x})$ and compute $\mathrm{Var}[Y]$. What can you say about $\mathrm{Var}[Y]$ if $\mathcal{A}$ is unstable? (6 pt)
Side note This means that even with an unstable training algorithm, we will still get reduced variance with bagging.

(b) We learned from the lecture that the AdaBoost algorithm constructs an ensemble of binary classifiers (with outputs in $\{-1, +1\}$). In this exercise, we explain the design of the AdaBoost algorithm in greater detail.

The AdaBoost algorithm builds an ensemble classifier $F_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$ by adding one learner at a time. At step t, it defines $F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \alpha_t h_t(\mathbf{x})$ (with $F_0 = 0$) by finding (h_t, α_t) that minimizes the exponential loss

$$L_t = \sum_{i=1}^{N} \exp(-y^{(i)} F_t(\mathbf{x}^{(i)})) = \sum_{i=1}^{N} \exp(-y^{(i)} F_{t-1}(\mathbf{x}^{(i)})) \cdot \exp(-y^{(i)} \alpha_t h_t(\mathbf{x}^{(i)}))$$

Let $D_t(i)$ be the normalized sample weights from the previous step, defined as

$$D_t(i) = \frac{\exp(-y^{(i)} F_{t-1}(\mathbf{x}^{(i)}))}{\sum_{j=1}^{N} \exp(-y^{(j)} F_{t-1}(\mathbf{x}^{(j)}))} = \frac{\exp(-y^{(i)} F_{t-1}(\mathbf{x}^{(i)}))}{L_{t-1}}$$

(For $t = 1$, $F_0 = 0$, so $D_1(i) = 1/N$.)

Since $L_t = L_{t-1} \cdot \left[\sum_{i=1}^{N} D_t(i) \exp(-y^{(i)} \alpha_t h_t(\mathbf{x}^{(i)}))\right]$ and L_{t-1} is a fixed constant, minimizing L_t is equivalent to minimizing the normalized objective

$$J_t = \sum_{i=1}^{N} D_t(i) \exp(-y^{(i)} \alpha_t h_t(\mathbf{x}^{(i)}))$$

i. Suppose the weak learner h_t has been chosen to be the learner that best minimizes the weighted error ϵ_t defined by

$$\epsilon_t = \sum_{i=1}^{N} D_t(i)\mathbb{I}(y^{(i)} \neq h_t(\mathbf{x}^{(i)})) = \sum_{\substack{1 \leq i \leq N \\ y^{(i)} \neq h_t(\mathbf{x}^{(i)})}} D_t(i)$$

Assume that $\epsilon_t \in (0,1)$. (If $\epsilon_t = 0$, the learner is perfect and we can stop. If $\epsilon_t = 1$, we can flip the learner's signs to achieve $\epsilon_t = 0$.)
Prove that the optimal weight α_t that *minimizes* the objective function J_t is given by:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

(6 pt)
Hint Split L_t into two parts: one for incorrectly classified samples and one for correctly classified samples. Then, try to rewrite it in terms of ϵ_t. Finally, take the derivative and set it to 0.

ii. The AdaBoost algorithm updates the sample weights for the next iteration $(t+1)$ using the rule:

$$D_{t+1}(i) = \frac{1}{Z_t} D_t(i) \exp(-y^{(i)} \alpha_t h_t(\mathbf{x}^{(i)}))$$

where Z_t is a normalization constant (i.e., so that $\sum_{i=1}^{N} D_{t+1}(i) = 1$).

A. Using α_t from the previous part, show that $Z_t = J_t$. Derive a formula for Z_t that is only dependent on ϵ_t. (6 pt)

B. Derive a formula for $\exp(-y^{(i)} \alpha_t h_t(\mathbf{x}^{(i)}))$ in the two cases $y^{(i)} = h_t(\mathbf{x}^{(i)})$ and $y^{(i)} \neq h_t(\mathbf{x}^{(i)})$ that is only dependent on α_t. (4 pt)

iii. The training error of the final ensemble $H_T(\mathbf{x}) = \text{sign}(F_T(\mathbf{x}))$ is

$$\epsilon_{\text{train}} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(H_T(\mathbf{x}^{(i)}) \neq y^{(i)})$$

Prove that the training error of AdaBoost is exponentially bounded:

$$\epsilon_{\text{train}} \leq \prod_{t=1}^{T} Z_t$$

(8 pt)
Hint Start by showing $\mathbb{I}(H_T(\mathbf{x}^{(i)}) \neq y^{(i)}) \leq \exp(-y^{(i)} F_T(\mathbf{x}^{(i)}))$. Then, see if you can relate L_T to the product of Z_t.
Side note This implies that, if each weak learner does better than just random chance (i.e. $\epsilon_t < 0.5$), then $Z_t < 1$, and $\epsilon_{\text{train}} \to 0$ as $T \to \infty$.

Solution.

(a-i) We have the bagging predictor

$$H_B(\boldsymbol{x}) = \frac{1}{B} \sum_{b=1}^{B} h_b(\boldsymbol{x}).$$

All $h_b(\boldsymbol{x})$ have variance σ^2 and pairwise covariance C. Compute

$$(H_B) = \Big(\frac{1}{B}\sum_{b=1}^{B} h_b\Big) = \frac{1}{B^2}\Big(\sum_{b=1}^{B}(h_b) + \sum_{b\neq b'}(h_b, h_{b'})\Big).$$

So

$$\boxed{(H_B) = \frac{\sigma^2}{B} + \frac{B-1}{B}\,C}.$$

(a-ii) Limit as $B \to \infty$

Take $B \to \infty$:

$$\lim_{B\to\infty}(H_B) = \lim_{B\to\infty}\Big(\frac{\sigma^2}{B} + \frac{B-1}{B}C\Big) = C.$$

(a-iii) Show $C < \sigma^2$ for unstable learners and variance reduction

Let $Y = h_i - h_j$. Then

$$(Y) = (h_i) + (h_j) - 2(h_i, h_j) = 2\sigma^2 - 2C.$$

If the algorithm is unstable then $h_i \neq h_j$ with positive probability, so $(Y) > 0$. Thus $2\sigma^2 - 2C > 0$ and hence $C < \sigma^2$.

Plugging this into the expression for (H_B) gives

$$(H_B) = \frac{\sigma^2}{B} + \frac{B-1}{B}C < \frac{\sigma^2}{B} + \frac{B-1}{B}\sigma^2 = \sigma^2,$$

for any $B > 1$. So bagging reduces variance.

(b) AdaBoost - derivations

We use $D_t(i)$ as the normalized weights and let

$$\epsilon_t = \sum_{i:y^{(i)}\neq h_t(\boldsymbol{x}^{(i)})} D_t(i).$$

(i) Optimal α_t. Split the objective

$$J_t = \sum_i D_t(i)\exp(-y^{(i)}\alpha_t h_t(\boldsymbol{x}^{(i)}))$$

into correctly and incorrectly classified examples. For correctly classified ones $yh_t = +1$, term is $e^{-\alpha_t}$; for incorrect ones $yh_t = -1$, term is $e^{+\alpha_t}$. Thus

$$J_t = (1-\epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t}.$$

Differentiate w.r.t. α_t and set to zero:

$$-(1-\epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t} = 0 \quad\Rightarrow\quad e^{2\alpha_t} = \frac{1-\epsilon_t}{\epsilon_t}.$$

So

$$\boxed{\alpha_t = \tfrac{1}{2}\ln\Big(\frac{1-\epsilon_t}{\epsilon_t}\Big)}.$$

(ii) Z_t equals J_t and closed form in ϵ_t. By definition of the weight update,

$$D_{t+1}(i) = \frac{1}{Z_t}D_t(i)\exp(-y^{(i)}\alpha_t h_t(\boldsymbol{x}^{(i)})),$$

so Z_t is exactly the normalizer

$$Z_t = \sum_i D_t(i) \exp(-y^{(i)} \alpha_t h_t(\boldsymbol{x}^{(i)})) = J_t.$$

Using the split form and the optimal α_t:

$$Z_t = (1 - \epsilon_t) e^{-\alpha_t} + \epsilon_t e^{\alpha_t}.$$

Plugging $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$ gives (after a small algebra step)

$$\boxed{Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}}.$$

(ii) Explicit exponent values. For any example:

$$\exp(-y^{(i)} \alpha_t h_t(\boldsymbol{x}^{(i)})) = \begin{cases} e^{-\alpha_t}, & \text{if } y^{(i)} = h_t(\boldsymbol{x}^{(i)}) \text{ (correct)}, \\ e^{+\alpha_t}, & \text{if } y^{(i)} \neq h_t(\boldsymbol{x}^{(i)}) \text{ (incorrect)}. \end{cases}$$

(iii) Exponential bound on training error. For any training example i,

$$\mathbb{I}\big(\text{sign}(F_T(\boldsymbol{x}^{(i)})) \neq y^{(i)}\big) \leq \exp\big(-y^{(i)} F_T(\boldsymbol{x}^{(i)})\big),$$

because if the sign is wrong then $y^{(i)} F_T(\boldsymbol{x}^{(i)}) \leq 0$ and the exponential is at least 1. Summing over i and dividing by N gives

$$\epsilon_{\text{train}} \leq \frac{1}{N} \sum_{i=1}^{N} \exp\big(-y^{(i)} F_T(\boldsymbol{x}^{(i)})\big) = \frac{L_T}{N},$$

where $L_T = \sum_i \exp(-y^{(i)} F_T(\boldsymbol{x}^{(i)}))$. But AdaBoost updates satisfy

$$L_T = L_0 \prod_{t=1}^{T} Z_t,$$

and with $L_0 = N$ (since $F_0 = 0$) we get

$$\epsilon_{\text{train}} \leq \frac{L_T}{N} = \prod_{t=1}^{T} Z_t.$$

So

$$\boxed{\epsilon_{\text{train}} \leq \prod_{t=1}^{T} Z_t},$$

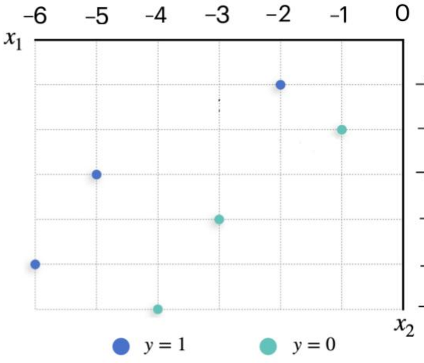
which shows training error is driven down exponentially when each $\epsilon_t < 1/2$ (because then $Z_t < 1$).

3. K-Means (20 pt)

(a) Suppose we want to perform K-Means clustering on the data samples from the **below** example Fig 3a to cluster them into two groups. Using the algorithm in the slides, perform two runs of the algorithm in the two following settings by showing your work for each step:

i. Setting the initial centers to $\boldsymbol{\mu}_1 = [-1, 0.5]$ and $\boldsymbol{\mu}_2 = [-5.5, -4.5]$. (5pt)

ii. Setting the initial centers to $\boldsymbol{\mu}_1 = [0, -4]$ and $\boldsymbol{\mu}_2 = [-5, 0]$. (5pt)



(b) Implement the K-means Algorithm (Lloyd's algorithm) introduced in the lecture, detailed as follows. (10 pt)

Algorithm 1 K-means Clustering (Lloyd's Algorithm)

```
Require: Dataset $\{x^{(i)}\}_{i=1}^{N}$, number of clusters $k$ (i.e., n_clusters)
Ensure: Cluster centers $\{\mu_1, \mu_2, \ldots, \mu_k\}$ and assignments $\{C_1, C_2, \ldots, C_k\}$
1: Initialization: Randomly choose $k$ examples as initial cluster centers
       $\mu_1^{(0)}, \mu_2^{(0)}, \ldots, \mu_k^{(0)}$.
2: Iterate for $t = 0, 1, 2, \ldots$ until convergence:
     (1) Assignment Step:
         Assign each data point $x^{(i)}$ to its nearest cluster center:
           $C_\ell = \left\{ x^{(i)} : \|x^{(i)} - \mu_\ell^{(t)}\|_2^2 \le \|x^{(i)} - \mu_j^{(t)}\|_2^2,\ \forall j \in \{1, \ldots, k\} \right\}$
     (2) Update Step:
         Recompute each cluster centroid as the mean of its assigned points:
           $\mu_\ell^{(t+1)} = \frac{1}{|C_\ell|} \sum_{x^{(i)} \in C_\ell} x^{(i)}, \quad \ell = 1, 2, \ldots, k$
3: Convergence: Stop when cluster assignments no longer change.
4: Output: Final cluster centers $\{\mu_\ell\}$ and assignments $\{C_\ell\}$.
```

Solution.

(a) Manual runs (short guidance)

Run 1: initial centers $\boldsymbol{\mu}_1 = [-1, 0.5]$, $\boldsymbol{\mu}_2 = [-5.5, -4.5]$.

- *Assignment step*: compute distances from each point to $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$, assign each point to the nearer center.
- *Update step*: recompute $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ as the mean of assigned points.
- *Repeat* assignment and update until assignments stop changing.

Run 2: initial centers $\boldsymbol{\mu}_1 = [0, -4]$, $\boldsymbol{\mu}_2 = [-5, 0]$.

- Same steps: assignment then update then repeat until convergence.
- you can see how different initial centers can lead to different convergence paths; still, K-means will reach some local optimum for the objective.

(b) Lloyd's algorithm (short description / justification)

Algorithm (Lloyd): initialize k centers, then repeat:

$$(1)\ \text{Assign } x^{(i)} \to \arg\min_{\ell} \|x^{(i)} - \mu_\ell\|_2^2, \qquad (2)\ \text{Update } \mu_\ell \leftarrow \frac{1}{|C_\ell|} \sum_{x^{(i)} \in C_\ell} x^{(i)}.$$

Stop when assignments don't change. This monotonically decreases the K-means objective

$$\sum_{\ell=1}^{k} \sum_{x^{(i)} \in C_\ell} \|x^{(i)} - \mu_\ell\|_2^2,$$

so the algorithm converges to a (possibly local) minimum. Your implementation lines up with this: compute distances, assign, recompute means, repeat until stable.

4. Kernels (44 pt)

In this problem, we want to show that new valid kernels can be made by combining existing ones. Suppose k_1, k_2 are valid kernel functions. You need to show:

(a) i. The set of valid kernels is closed with respect to the scalar product. Formally: $ck_1(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)})$ for any $c \geq 0$ would be a valid kernel. (4 pt)

ii. The set of valid kernels is closed with respect to addition. Formally: $k_1(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}) + k_2(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)})$ would be valid kernel. (6 pt)

iii. The set of valid kernels is closed with respect to multiplication. Formally, $k_1(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)})k_2(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)})$ would be valid kernel. (6 pt)

iv. The set of valid kernels is closed with composition mapping. Formally, $k_1(f(\boldsymbol{x}^{(1)}), f(\boldsymbol{x}^{(2)}))$ would be valid kernel, where $f : \mathbb{R}^d \to \mathbb{R}^d$ is a mapping function. (8 pt)

v. The set of valid kernels is closed with exponentiation. Formally, $\exp(k_1(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}))$ would be valid kernel. (8 pt)

Hint: You can use either of the two following methods for showing that a function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a valid kernel function:

- The matrix
$$K_{ij} = k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$$
is symmetric and positive semidefinite for any set of vectors $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(n)} \in \mathbb{R}^d$
- $k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \phi(\boldsymbol{x}^{(i)})^T \phi(\boldsymbol{x}^{(j)})$ for some transformation ϕ

Hint:
$$\exp(x) = \sum_{m=0}^{\infty} \frac{x^m}{m!}.$$

(b) Recall that the dual problem of a hard-margin SVM is

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i}^{N} \sum_{j}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} \boldsymbol{x}^{(i)\top} \boldsymbol{x}^{(j)} \qquad \text{s.t.} \quad \alpha_i \geq 0, \ i = 1, \ldots, N; \ \sum_{j=1}^{N} \alpha_j y^{(j)} = 0$$

Denote the optimal primal solution as $\boldsymbol{w}^*$. We define the domain $\mathcal{C} = [0, \infty]^N = \{\boldsymbol{\alpha} : \alpha_i \geq 0\}$ for a hard-margin SVM, and $\mathcal{C} = [0, C]^N = \{\boldsymbol{\alpha} : 0 \leq \alpha_i \leq C\}$ for a soft-margin SVM. We can solve this dual problem by projected gradient descent, which starts from some $\boldsymbol{\alpha}_0 \in \mathcal{C}$ (e.g., $\mathbf{0}$) and updates as follows:

$$\boldsymbol{\alpha}_{t+1} = \Pi_{\mathcal{C}} \left[\boldsymbol{\alpha}_t - \eta \nabla f(\boldsymbol{\alpha}_t) \right].$$

Here $\Pi_{\mathcal{C}}[\boldsymbol{\alpha}]$ is the *projection* of $\boldsymbol{\alpha}$ onto $\mathcal{C}$, defined as the closest point to $\boldsymbol{\alpha}$ in $\mathcal{C}$:

$$\Pi_{\mathcal{C}}[\boldsymbol{\alpha}] := \arg\min_{\boldsymbol{\alpha}' \in \mathcal{C}} \|\boldsymbol{\alpha}' - \boldsymbol{\alpha}\|_2.$$

If $\mathcal{C}$ is convex, in Homework 3, we already prove that

$$\left(\Pi_{[0,\infty)^N}[\boldsymbol{\alpha}]\right)_i = \max\{\alpha_i, 0\},$$
$$\left(\Pi_{[0,C]^N}[\boldsymbol{\alpha}]\right)_i = \min\{\max\{0, \alpha_i\}, C\}.$$

i. Assume we use radial basis kernel function $k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \exp(-\frac{1}{2}\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\|_2^2)$. If the test point $\boldsymbol{x}$ is far away from any training point $\boldsymbol{x}^{(i)}$, prove that $\boldsymbol{w}^{*\top} \phi(\boldsymbol{x}) + b \approx b$. The definition of $\boldsymbol{w}^*$ is the same as above. (6 pt)

ii. On the area $[-8, 8] \times [-8, 8]$, plot the contour lines of the following kernel SVMs, trained on the XOR data. Different kernels and XOR data are **provided in** hw5_utils.py. Learning rate 0.1 and 10000 steps should be enough. To draw the contour lines, you can use `hw5_utils.svm_contour()`.

- The polynomial kernel with degree 3.
- The RBF kernel with $\sigma = 1$.
- The RBF kernel with $\sigma = 2$.
- The RBF kernel with $\sigma = 5$.

Include these four plots in your **written submission**. (8 pt)

Solution.

Q4 - Kernels

(a) Kernel "is this still a kernel?" checks

(i) Multiply by a positive constant If k_1 works as a kernel, scaling it by $c \geq 0$ doesn't break anything - you're just stretching the similarity scores. Everything stays legal.

(ii) Adding two kernels If two kernels measure similarity in two different ways, adding them is like blending those two views. Since both were valid before, the combo stays valid.

(iii) Multiplying two kernels This one sounds weird at first, but it's allowed. There's a matrix fact ("Schur product theorem") that basically says: if both similarity matrices are good, multiplying them entry-by-entry keeps them good. So product kernels are fine.

(iv) Kernels after applying a function f If you take a kernel and just plug $f(x)$ into it instead of x, you're basically pretending your data actually lives in the "f-transformed" world. Kernels don't mind; this is still a legit kernel.

(v) Exponential of a kernel This one looks fancy but the idea is simple: the exponential can be expanded into an infinite sum of powers, and we already said adding kernels and multiplying kernels is fine. So the whole thing stays a kernel.

(b) RBF + SVM intuition + plots

(i) Why a far-away point gives score $\approx b$ The RBF kernel gets tiny when points are far apart. So if a test point x is nowhere near any training point, every term

$$\alpha_i y_i k(x_i, x)$$

shrinks basically to zero. So the whole SVM prediction collapses to just the bias term b. In English: "if I've never seen anything like this point before, the machine just shrugs and goes with the default."

(ii) The contour plots You just train four SVMs; one poly (degree 3) and three RBFs with different widths, and call the provided plotting function. The poly model usually gives very global, smooth curves, while the RBF ones look more "blobby," with tighter or looser bends depending on σ. Just screenshot the plots and insert them.