CS 273P: Machine Learning and Data Mining
Homework 3
Due date: **See Canvas**
Instructor: Xiaohui Xie

This homework is graded automatically on **Gradescope** using an autograder. All questions are designed to be unit-testable: you will implement specific functions that return specified outputs that satisfy the autograder.

Allowed libraries: You may use `numpy`, `scipy`, `pandas`, and `scikit-learn`.

Submission rules (READ CAREFULLY):

1. Download the starter files from the GitHub link.
2. Complete your work by editing only the provided Python files: `problem1.py` and `problem2.py`.
3. Create a plain text file called `statement.txt` containing your collaboration statement.
4. Submit to Gradescope by uploading a single **ZIP file** containing exactly:
   - `problem1.py`
   - `problem2.py`
   - `statement.txt`

   No folders/subdirectories; all files must be at the top level of the zip.
5. You may resubmit before the deadline; your **highest score** counts.

If you run into issues with the autograder or submission format, post on Ed Discussion so others can benefit from the discussion.

Points: This homework adds up to a total of **100 points** with **10 points extra credit**, as follows:

Problem 1: Logistic Regression (from scratch + analysis utilities)	75 (+10) points
Problem 2: Shattering and VC Dimension (computational check)	20 points
Statement of Collaboration	5 points

Problem 1: Logistic Regression (75 + 10 points)

In this problem, you will implement a binary logistic regression learner and test it on two binary tasks constructed from the Iris dataset.

Data. Load `data/iris.txt`. Use only the first two features: X = `iris[:,0:2]` and labels Y = `iris[:,-1]`. Standardize features (zero mean, unit variance) before training (recommended for optimization stability).

Create two binary classification datasets:

- Dataset A: classes $\{0, 1\}$ (original labels 0 vs 1)
- Dataset B: classes $\{1, 2\}$ (original labels 1 vs 2)

Use **all** points for training (no train/val split for this homework).

All required functions for Problem 1 are in `problem1.py`. The autograder calls them directly.

1.1 Data preparation and sanity checks (10 points)

Implement:

1. `load_iris_binary(path, pair) -> (X, y)` (6 pts)
   `pair` is either `(0,1)` or `(1,2)`. Return $X \in R^{N\times 2}$ and $y \in \{0,1\}^N$ using the specified class pair, where the smaller class is mapped to 0 and the larger class to 1. Standardize X using `only the returned data`.

2. `is_linearly_separable(X, y, tol=1e-6) -> bool` (4 pts)
   Return `True` if there exists a linear separator that achieves **zero** training error, else `False`. (*Hint:* you may use `sklearn.svm.LinearSVC` with a large C and check training error.)

1.2 Logistic regression: core math (25 points)

Implement a logistic regression model trained with (stochastic or mini-batch) gradient descent. Your implementation must be deterministic given a seed.

Implement:

1. `sigmoid(z) -> ndarray` (3 pts): elementwise $\sigma(z) = \frac{1}{1+e^{-z}}$ with numerical stability.

2. `logistic_loss(X, y, theta, reg=0.0) -> float` (7 pts)
   Negative log-likelihood averaged over data, with optional L2 regularization:

$$J(\theta) = \frac{1}{N}\sum_{j=1}^{N}\left[-y^{(j)}\log\sigma(r^{(j)}) - (1-y^{(j)})\log(1-\sigma(r^{(j)}))\right] + \frac{\text{reg}}{2}\|\theta_{1:}\|_2^2,$$

   where $r^{(j)} = \theta_0 + x^{(j)}\cdot\theta_{1:}$ and the bias term θ_0 is `not` regularized.

3. `logistic_grad(X, y, theta, reg=0.0) -> ndarray` (10 pts)
   Return the gradient of the above objective with respect to `theta`.

4. `predict_proba(X, theta) -> ndarray` (2 pts): return $p(y{=}1 \mid x)$ for each row of X.

5. `predict(X, theta, threshold=0.5) -> ndarray` (3 pts): return predictions in $\{0,1\}$.

1.3 Training with early stopping + diagnostics (30 points)

Implement:

1. `train_logreg(X, y, step_size=0.1, max_epochs=2000, tol=1e-6, batch_size=0, reg=0.0, seed=0) -> dict` (20 pts)
   Train logistic regression starting from `theta=0`. If `batch_size=0`, use full-batch GD; otherwise use mini-batches of that size. Stop when either:

   - `max_epochs` is reached, or
   - the absolute change in loss between consecutive epochs is $<$ `tol`.

   Return a dictionary with at least: `{ "theta": theta, "loss_history": loss_hist, "err_history": err_hist, "epochs": E }`. Error is the training misclassification rate using threshold 0.5.

2. `decision_boundary(theta, x1_grid) -> ndarray` (5 pts)
   Given `x1_grid` (1D array), return the corresponding boundary values x_2 where $\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$. If $\theta_2 = 0$, return `np.full_like(x1_grid, np.nan)`.

3. `evaluate_trained(X, y, theta) -> dict` (5 pts)
   Return a dict with `loss`, `error_rate`, and `confusion` (a 2×2 confusion matrix).

1.4 More challenging: calibration + threshold tuning (10 points)

In practice, we often tune the decision threshold. Implement:

1. `best_threshold(y_true, p_hat, metric="f1") -> float` (10 pts)
   Given true labels and predicted probabilities, search thresholds in $\{0.05, 0.10, \ldots, 0.95\}$ and return the threshold that maximizes the chosen metric: `"f1"` or `"balanced_accuracy"`. Break ties by returning the `smallest` threshold.

Extra Credit: regularization path (10 points)

Implement:

1. `regularization_path(X, y, regs, **train_kwargs) -> dict` (10 pts)
   For each reg in `regs` (a list of nonnegative floats), train a model and return a dict with: `{ "regs": regs, "thetas": list_of_theta, "losses": list_of_final_loss }`.

Problem 2: Shattering and VC Dimension (20 points)

In this problem, you will determine computationally whether small point sets can be shattered by different hypothesis classes in R^2.

Datasets. We provide four small point sets (A–D). Each dataset is a $n \times 2$ array of points. No three points are collinear.

Binary labelings. Given n points, there are 2^n possible labelings in $\{-1, +1\}$. A hypothesis class `shatters` a dataset if it can realize `every` labeling.

All required functions for Problem 2 are in `problem2.py`.

2.1 Hypothesis classes as feasibility checks (12 points)

Implement:

1. `all_labelings(n) -> ndarray` (3 pts)
   Return an array of shape $(2^n, n)$ with entries in $\{-1, +1\}$ enumerating all binary labelings (in lexicographic order over bit patterns is fine).

2. `shattered_by_linear_threshold(X) -> bool` (5 pts)
   Return whether the class $T(a + bx_1 + cx_2)$ shatters X. (*Hint:* for each labeling, check feasibility of a strict linear separation; you may use `sklearn.svm.LinearSVC(C=1e6)` and verify it fits perfectly.)

3. `shattered_by_axis_threshold(X) -> bool` (4 pts)
   Return whether the class $T(a + bx_1)$ shatters X (a 1D threshold on x_1, ignoring x_2).

2.2 More challenging hypothesis classes (8 points)

Implement:

1. `shattered_by_circle(X) -> bool` (4 pts)
   Return whether the class $T((x_1 - a)^2 + (x_2 - b)^2 + c)$ shatters X. Your implementation must be deterministic. For unit testing, you may assume $n \leq 4$ and perform a bounded grid search over (a, b, c) using a fixed grid defined in the starter code.

2. `shattered_by_two_parallel_lines(X) -> bool` (4 pts)
   Return whether the classifier $T(a + bx_1 + cx_2) \times T(d + bx_1 + cx_2)$ shatters X (product of two thresholds with shared normal vector, i.e., two parallel lines). Your implementation must be deterministic. For unit testing, you may assume $n \leq 4$ and perform a bounded search over a fixed set of directions (b, c) and offsets (a, d) defined in the starter code.

No writeup required. Your submission is graded entirely by the correctness of your returned values.

Statement of Collaboration (5 points)

Add a plain text file `statement.txt` to your submission. List the names of collaborators and the nature of collaboration, or write "No collaboration." if you worked alone. Do not share code. Academic honesty policies apply.