ECE228 Machine Learning for Physical Applications (Spring 2025): Assignment 1

Instructor: Yuanyuan Shi, yyshi@ucsd.edu
Co-instructors and Teaching Assistants:
Yuexin Bian, yubian@ucsd.edu
Luke Bhan, lbhan@ucsd.edu
Rich Pai, cpai@ucsd.edu
Zekai Zhao, zez020@ucsd.edu

* Release Time: April 10, 10:00 AM
*** Deadline is April 23, 11:59 PM**

Submission format:

- For this assignment, you will be having 2 parts, the first part has theory questions and the second part is a coding one. All the questions should be done **Individually**.
- You need to submit two different files to Gradescope. One for the Written Questions(Part I) and one for Coding Questions(Part II).
- Please submit your HW as a typed PDF document (not handwritten). It is encouraged that you LaTeX all your work, though you may use another comparable typesetting method. The PDF document should include solutions to **both** the written questions and coding questions (results, discussions, plots, and so on).
- For Coding Questions, you should submit the ipynb (by running all cells in the notebook and including all the outputs) in Gradescope. Additional details have been given at the bottom.

1 Written Questions [35 Pts]

1.1 Linear Regression and ℓ_2 Regularization

In this problem, we explore the theoretical underpinnings of linear regression and the role of ℓ_2 regularization in mitigating overfitting. Starting from the standard least-squares formulation, we will derive the closed-form solution for the optimal model parameters and then extend the analysis to the regularized case. Finally, we will examine how input noise implicitly induces a regularization

effect, offering a deeper understanding of the robustness of linear models.

1. [6 pts] Let $\{(x_i, y_i)\}_{i=1}^N$ be a dataset of N input-output pairs, where $x_i \in \mathbb{R}^k$ is the feature (column) vector and $y_i \in \mathbb{R}$ is the scalar target output. We seek a linear model parameterized by $(w, w_0) \in \mathbb{R}^k \times \mathbb{R}$, i.e.,

$$f_{w,w_0}(x_i) = w^\top x_i + w_0, \quad w = \begin{bmatrix} w_1 & \cdots & w_k \end{bmatrix}^\top,$$

that minimizes the average squared loss

$$L(w, w_0) = \frac{1}{2N} \sum_{i=1}^{N} \left(y_i - f_{w,w_0}(x_i)\right)^2.$$

(a) [2 pts] Let

$$\tilde{x}_i = \begin{bmatrix} x_i \\ 1 \end{bmatrix} \in \mathbb{R}^{k+1}, \quad \tilde{w} = \begin{bmatrix} w \\ w_0 \end{bmatrix} \in \mathbb{R}^{k+1}.$$

Define the input matrix $X \in \mathbb{R}^{N \times (k+1)}$ and response vector $y \in \mathbb{R}^N$ as:

$$X = \begin{bmatrix} \tilde{x}_1^\top \\ \vdots \\ \tilde{x}_N^\top \end{bmatrix}, \qquad y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}.$$

Using these notations, express the average squared loss $L(w, w_0)$ as a matrix expression involving X, y, and $\tilde{w}$.

(b) [4 pts] Using your answer from part (a), derive the closed-form optimal solution $\tilde{w}^\star$ that minimizes the average squared loss, i.e.,

$$\tilde{w}^\star = \text{argmin}_{\tilde{w} \in \mathbb{R}^{k+1}} L(w, w_0).$$

Your answer should be expressed using matrix operations involving X and y, assuming the matrix X is full column rank.

2. [9 pts] A common way to prevent overfitting is to introduce an ℓ_2 regularization term, leading to the regularized loss:

$$L_{\text{reg}}(w, w_0) = \frac{1}{2N} \sum_{i=1}^{N} \left(y_i - f_{w,w_0}(x_i)\right)^2 + \frac{\lambda}{2} \|w\|_2^2,$$

where $\lambda > 0$ is the regularization parameter and $\|w\|_2^2 = w_1^2 + \cdots + w_k^2$.

(a) [4 pts] Derive the closed-form solution $\tilde{w}^\star$ that minimizes this regularized loss, i.e.,

$$\tilde{w}^\star = \text{argmin}_{\tilde{w} \in \mathbb{R}^{k+1}} L_{\text{reg}}(w, w_0).$$

Your answer should be expressed using matrix operations involving X and y.

(b) [3 pts] What happens to the solution as $\lambda \to 0$ and $\lambda \to \infty$?

(c) [2 pts] Discuss why the regularization term does not apply to the bias term w_0.

3. [5 pts] In the previous problems, we implicitly assumed access to clean input data. However, real-world inputs are often subject to perturbations. This problem investigates how such input noise influences the expected loss and gives rise to an implicit regularization effect.

   Suppose each input $x_i \in \mathbb{R}^k$ is corrupted by additive noise $\eta_i \in \mathbb{R}^k$. We may consider the expected average squared loss under input noise as

$$\tilde{L}(w, w_0) = \mathbb{E}\left[\frac{1}{2N}\sum_{i=1}^{N}\left(y_i - f_{w,w_0}(x_i + \eta_i)\right)^2\right]$$

   where the expectation is over the noises $\eta_1, \ldots, \eta_N$. Assume the additive noise

$$\eta_i = \begin{bmatrix}\eta_{i,1} & \eta_{i,2} & \cdots & \eta_{i,k}\end{bmatrix}^\top, \quad \text{for } i = 1, \ldots, N$$

   are zero-mean random vectors with uncorrelated components, i.e,

$$\mathbb{E}[\eta_{i,j}] = 0, \quad \mathbb{E}[\eta_{i,j}\eta_{i',j'}] = \delta_{i,i'}\delta_{j,j'}\sigma^2,$$

   for $i, i' \in \{1, \ldots, N\}$ and $j, j' \in \{1, \ldots, k\}$ and the Kronecker delta means

$$\delta_{i,i'} = \begin{cases} 1 & \text{if } i = i', \\ 0 & \text{otherwise} \end{cases}.$$

   Please show that

$$\tilde{L}(w, w_0) = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - f_{w,w_0}(x_i)\right)^2 + \frac{\sigma^2}{2}\|w\|_2^2.$$

   That is, minimizing the expected loss under input noise is equivalent to minimizing noise-free loss with an added ℓ_2-regularization term on the weights.

1.2 Generative Classification and Maximum Likelihood Estimation

This problem focuses on the derivation of a generative classification model using maximum likelihood estimation (MLE). You will derive the MLEs for the class prior probabilities, class means, and a shared covariance matrix under the assumption of Gaussian class-conditional densities.

Consider a classification problem with K classes. You are given a labeled training set $\{x_n, t_n\}_{n=1}^N$, where each $x_n \in \mathbb{R}^m$ is a feature vector, and

$$t_n = \begin{bmatrix}t_{n,1} & t_{n,2} & \cdots & t_{n,K}\end{bmatrix}^\top \in \mathbb{R}^K$$

is a one-hot encoded class label vector such that $t_{n,k} = 1$ if sample n belongs to class C_k, and 0 otherwise.

We assume that the data points are generated by a generative model, which specifies:

- Class prior probabilities: $p(C_k) = \pi_k$,
- Class-conditional densities $p(x_n|C_k)$.

We also assume that the data points are drawn independently from this model.

1. [5 pts] We will begin by deriving the MLE for the class prior probabilities $\{\pi_k\}_{k=1}^K$.

   (a) [2 pts] Write down the log-likelihood of all the data points under the categorical prior, i.e., the probability of observing all the data points as **a function of the class prior probabilities**:
   $$p\left(\{\pi_k\}_{k=1}^K; \{x_n, t_n\}_{n=1}^N\right).$$

   (b) [3 pts] Show that the MLE of the prior probability for class C_k is given by
   $$\pi_k = \frac{N_k}{N},$$
   where N_k is the number of training samples labeled as class C_k.

2. [10 pts] Now suppose the class-conditional densities are multivariate Gaussian with a shared covariance matrix, i.e., for $n \in \{1, \ldots, N\}$ and $k \in \{1, \ldots, K\}$
   $$\begin{aligned} p(x_n|C_k) &= \mathcal{N}(x_n|\mu_k, \Sigma) \\ &= \frac{1}{(2\pi)^{m/2}\det(\Sigma)^{1/2}} \exp\left(-\frac{1}{2}(x_n - \mu_k)^\top \Sigma^{-1}(x_n - \mu_k)\right), \end{aligned}$$
   where $\mu_k \in \mathbb{R}^m$ is the mean vector for class C_k and $\Sigma \in \mathbb{R}^{m\times m}$ is the shared covariance matrix across all K classes assumed to be non-singular.

   (a) [4 pts] Show that the MLE of the mean vector μ_k for class C_k is
   $$\mu_k = \frac{1}{N_k}\sum_{n=1}^N t_{n,k} x_n.$$

   (b) [6 pts] Define the within-class scatter matrix for class C_k as
   $$S_k = \frac{1}{N_k}\sum_{n=1}^N t_{n,k}(x_n - \mu_k)(x_n - \mu_k)^\top.$$
   Show that the MLE of the shared covariance matrix is
   $$\Sigma = \sum_{k=1}^K \frac{N_k}{N} S_k.$$
   Hint: Consider taking derivative with respect to Σ^{-1} and using the following identities:

   (1) $\frac{\partial \log\det\left(\Sigma^{-1}\right)}{\partial \Sigma^{-1}} = \Sigma^\top = \Sigma$

   (2) $\frac{\partial \mathrm{tr}(A\Sigma^{-1})}{\partial \Sigma^{-1}} = A^\top$ for any matrix A of an appropriate dimension

2 Coding Questions: Sparse Linear Regression [65 Pts]

In previous problems, we examined linear regression with ℓ_2-regularization (Ridge regression), where the objective is:

$$L_{\text{reg}}(w, w_0) = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - f_{w,w_0}(x_i)\right)^2 + \frac{\lambda}{2}\|w\|_2^2,$$

with $f_{w,w_0}(x_i) = w^\top x_i + w_0$. Ridge regression encourages small coefficients w but does not produce sparse solutions (most weights remain nonzero). In this problem, we will study the **LASSO** (Least Absolute Shrinkage and Selection Operator), which uses an ℓ_1-penalty to promote sparsity:

$$L_{\text{lasso}}(w, w_0) = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - w^\top x_i - w_0\right)^2 + \lambda\|w\|_1.$$

Here, $\lambda > 0$ controls the strength of regularization.

Unlike Ridge regression, LASSO does not admit a closed-form solution, due to the nonsmoothness of the ℓ_1-norm penalty. Despite this, LASSO can be solved efficiently using first-order optimization methods. In particular, we will consider the **ISTA** (Iterative Shrinkage-Thresholding Algorithm), which applies **proximal gradient descent** to handle the nonsmooth ℓ_1-regularizer. ISTA is simple to implement and often effective in practice. It is a special case of the **forward-backward splitting** framework for composite optimization problems. This approach decomposes the objective into a smooth term and a nonsmooth term, and alternates between two steps: a gradient descent step on the smooth part (the *forward step*), and the application of a proximal operator to the nonsmooth part (the *backward step*). We will now examine each of these steps in more detail.

Step 1: Objective Structure and the Forward Gradient Descent Step

Consider a dataset $\{(x_i, y_i)\}_{i=1}^N$ be of N input-output pairs, where $x_i \in \mathbb{R}^k$ is the feature (column) vector and $y_i \in \mathbb{R}$ is the scalar target output. Similarly as before, let $\tilde{x}_i = [x_i^\top \;\; 1]^\top \in \mathbb{R}^{k+1}$ and $\tilde{w} = [w^\top \;\; w_0]^\top \in \mathbb{R}^{k+1}$. Also define the data matrix $X \in \mathbb{R}^{N\times(k+1)}$ and label vector $y \in \mathbb{R}^N$ as $X = [\tilde{x}_1 \;\cdots\; \tilde{x}_N]^\top \in \mathbb{R}^{N\times(k+1)}$ and $y = [y_1 \;\cdots\; y_N]^\top \in \mathbb{R}^N$. Then, the LASSO objective becomes:

$$L_{\text{lasso}}(\tilde{w}) = \frac{1}{2}\|y - X\tilde{w}\|_2^2 + \lambda\|w\|_1.$$

To apply ISTA, we first rewrite the objective in the form of a composite optimization problem: $L_{\text{lasso}}(\tilde{w}) = g(\tilde{w}) + h(\tilde{w})$, where g is a smooth differentiable function and h is a convex but nonsmooth function. In our case:

$$g(\tilde{w}) = \frac{1}{2}\|y - X\tilde{w}\|_2^2, \quad h(\tilde{w}) = \lambda\|w\|_1.$$

Note that the ℓ_1-penalty is applied only to the first k components of $\tilde{w}$, corresponding to the weights w, not the bias term w_0.

The ISTA algorithm, as we will see shortly, uses a fixed step size α for the forward (gradient descent) step. To ensure convergence, α must satisfy $\alpha \leq \frac{1}{L}$, where L is the Lipschitz constant of ∇g. It is not difficult to derive:

$$\nabla g(\tilde{w}) = X^\top (X\tilde{w} - y), \quad L = \|X\|_2^2,$$

where $\|X\|_2$ is the spectral norm (i.e., the largest singular value) of X. As a result, at each iteration k, the forward step of ISTA updates the iterate by taking a gradient descent step:

$$z^{(t)} = \tilde{w}^{(t)} - \alpha \nabla g(\tilde{w}^{(t)}) = \tilde{w}^{(t)} - \alpha X^\top (X\tilde{w}^{(t)} - y),$$

where we choose $\alpha = \frac{1}{\|X\|_2^2}$ as the fixed step size. The intermediate vector $z^{(t)}$ is then passed to the backward step (proximal operator), which we will describe next.

Step 2: Backward Step: Proximal Operator for the ℓ_1-Norm

In ISTA, the backward step involves applying the proximal operator of the ℓ_1-regularizer. For our objective, the regularization term is

$$h(\tilde{w}) = \lambda \|w\|_1,$$

where $\tilde{w} = [w^\top \ w_0]^\top \in \mathbb{R}^{k+1}$, and the ℓ_1-penalty is applied only to the weights w, not the bias w_0.

The proximal operator of h, with step size $\alpha > 0$, is defined as:

$$\text{prox}_{\alpha h}(z) := \text{argmin}_{\tilde{w} \in \mathbb{R}^{k+1}} \left\{ \lambda \|w\|_1 + \frac{1}{2\alpha} \|\tilde{w} - z\|_2^2 \right\},$$

where $z \in \mathbb{R}^{k+1}$ is a vector of the same dimension as $\tilde{w}$. The solution to this problem is given coordinate-wise by:

$$[\text{prox}_{\alpha h}(z)]_i = \begin{cases} \text{SoftThreshold}(z_i, \alpha\lambda) & \text{if } i \leq k \quad \text{(weight entries)} \\ z_i & \text{if } i = k+1 \quad \text{(bias term)} \end{cases},$$

where the **soft-thresholding operator** is defined elementwise by:

$$\text{SoftThreshold}(z_i, \alpha\lambda) := \text{sign}(z_i) \cdot \max(|z_i| - \alpha\lambda,\, 0).$$

In other words, the proximal operator applies the soft-thresholding operator to each coordinate of the weight vector w, while leaving the bias term w_0 unchanged. More explicitly, we can write:

$$\text{SoftThreshold}(z_i, \alpha\lambda) = \begin{cases} z_i - \alpha\lambda & \text{if } z_i > \alpha\lambda \\ 0 & \text{if } |z_i| \leq \alpha\lambda \\ z_i + \alpha\lambda & \text{if } z_i < -\alpha\lambda \end{cases}.$$

Step 3: ISTA algorithm for LASSO

ISTA solves the LASSO problem via proximal gradient descent. At each iteration, it performs:

- A gradient descent step on the smooth loss
- A proximal step (soft-thresholding) on the weights w, but not on the bias w_0

Let the step size be $\alpha = \frac{1}{L}$, where $L = \|X\|_2^2$ (spectral norm squared). Below, we present two common variants of ISTA, differing in their stopping criteria.

- Algorithm 1 uses a fixed number of iterations, terminating after T updates regardless of convergence.
- Algorithm 2 stops when the change in loss between consecutive iterates falls below a user-defined tolerance ϵ.

Algorithm 1 ISTA for LASSO with Bias (Max iteration Stopping)

Require: Augmented data matrix $X \in \mathbb{R}^{N\times(k+1)}$, target vector $y \in \mathbb{R}^N$, regularization $\lambda > 0$, step size $\alpha > 0$, initial guess $\tilde{w}^{(0)}$, maximum number of iterations T

for $t = 0$ to $T-1$ **do**
  $z^{(t)} \leftarrow \tilde{w}^{(t)} - \alpha X^\top (X\tilde{w}^{(t)} - y)$ ▷ Forward step: gradient descent
  $\tilde{w}^{(t+1)} \leftarrow \text{prox}_{\alpha h}(z^{(t)})$ ▷ Backward step: soft thresholding

Ensure: Approximate solution $\tilde{w}^{(T)}$

Algorithm 2 ISTA for LASSO with Bias (Loss-Based Stopping)

Require: Augmented data matrix $X \in \mathbb{R}^{N\times(k+1)}$, target vector $y \in \mathbb{R}^N$, regularization $\lambda > 0$, step size $\alpha > 0$, initial guess $\tilde{w}^{(0)}$, tolerance $\varepsilon > 0$

Initialize $t \leftarrow 0$, $L_{\text{prev}} \leftarrow L_{\text{lasso}}(\tilde{w}^{(0)})$
while true **do**
  $z^{(t)} \leftarrow \tilde{w}^{(t)} - \alpha X^\top (X\tilde{w}^{(t)} - y)$ ▷ Forward step: gradient descent
  $\tilde{w}^{(t+1)} \leftarrow \text{prox}_{\alpha\lambda}(z^{(t)})$ ▷ Backward step: soft thresholding
  $L_{\text{new}} \leftarrow L_{\text{lasso}}(w^{(t+1)})$
  if $|L_{\text{new}} - L_{\text{prev}}| < \varepsilon$ **then break**
  $t \leftarrow t+1$, $L_{\text{prev}} \leftarrow L_{\text{new}}$

Ensure: Approximate solution $\hat{w} := \tilde{w}^{(t)}$

In the following tasks, you will:

- Implement the ISTA algorithm for solving LASSO.
- Test your implementation on a synthetic dataset.
- Apply your code to a real-world dataset.

You will receive a zip file containing a Jupyter notebook and the necessary supporting materials. Follow the instructions provided in the notebook to complete the tasks. Upon completion, please submit both your completed `.ipynb` file and the generated PDF to Gradescope.

2.1 Implement ISTA for Solving LASSO [15 Pts]

Implement ISTA in Algorithm 2. Here are some hints that you may find helpful:

- An important check for your implementation correctness is to ensure the loss objective is nonincreasing with each step.
- In some applications, it is necessary to solve the Lasso problem on the same dataset for multiple values of λ. This process is known as computing a *regularization path*. A common and efficient approach is to begin with a large value of λ, where the solution is typically sparse or entirely zero, and then iteratively decrease λ by a constant ratio.
- The largest value of λ for which the solution w is entirely zero is given by

$$\lambda_{\max} = \|X^T(y - \overline{y})\|_\infty, \tag{1}$$

  where $\bar{y}$ denotes the mean of the response vector y. This value provides a useful starting point when selecting the initial λ in a regularization path, as any $\lambda \geq \lambda_{\max}$ yields a zero solution.

2.2 Test LASSO Solver on a Synthetic Dataset [20 Pts]

We will now test your Lasso solver on a synthetic dataset. One of the key advantages of Lasso is its ability to promote sparsity in the solution. When many features are irrelevant for predicting the output, Lasso can help identify the truly important ones by shrinking the irrelevant coefficients to zero. This experiment will demonstrate how well Lasso can recover a sparse model from data with both relevant and irrelevant features.

Suppose the data points (x_i, y_i) are generated independently according to the model

$$y_i = w_0^* + w_1^* x_{i,1} + w_2^* x_{i,2} + \cdots + w_d^* x_{i,d} + \epsilon_i,$$

where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ is Gaussian noise. Here, k denotes the total number of features, but only the first d features are relevant, with $d \ll k$. In other words, the remaining features $x_{i,d+1}, \ldots, x_{i,k}$ are irrelevant for predicting y_i. You are now tasked with the following:

1. [5 pts] Let $N = 50$, $k = 80$, $d = 6$, and $\sigma = 0.1$. Generate a synthetic dataset as follows:
   - Construct a data matrix $X \in \mathbb{R}^{N \times (k+1)}$, where the first k columns are sampled independently from the standard normal distribution $\mathcal{N}(0, 1)$, and the last column is set to all ones to account for the bias term.

- Set the true bias $w_0^* = 0$, and define the ground-truth weight vector $w^* \in \mathbb{R}^k$ by assigning the first d entries to ± 10 (you may choose any sign pattern), and setting the remaining $k - d$ entries to zero.
- Generate a noise vector $\epsilon \in \mathbb{R}^N$ with entries drawn independently from $\mathcal{N}(0, \sigma^2)$.
- Form the response vector as:

$$y = X\tilde{w}^* + \epsilon,$$

  where $X \in \mathbb{R}^{N \times (k+1)}$ includes an appended column of ones to account for the bias term, and the augmented weight vector (including bias) is $\tilde{w}^* = [(w^*)^\top \ w_0^*]^\top \in \mathbb{R}^{k+1}$.

2. [12 pts] Using the synthetic dataset you generated in part (a), you will now solve a sequence of LASSO problems using different levels of regularization. We have provided code that constructs a sequence of ten decreasing λ values. These are generated starting from a computed $\lambda_{\max}$, and decay geometrically by a fixed ratio of 0.5:

$$\lambda_i = \lambda_{\max} \cdot (0.5)^i, \quad i = 0, 1, \ldots, 9.$$

   - For each λ, run your ISTA algorithm using the same initial point $\tilde{w}^{(0)} = 0$ to obtain the final solution $\tilde{w}_{\text{hat}}$ (the final convergent $\tilde{w}^{(t)}$ from algorithm 2).
   - For each λ, record and print the following quantities:
     - The data fitting loss $\frac{1}{2N}\|y - X\tilde{w}_{\text{hat}}\|_2^2$
     - The number of nonzero weights (excluding the bias term)
     - *Precision* and *recall*, as defined in eqs. (2) and (3).

     We have provided the code to generate the summary table. You only need to compute and store these values when you run ISTA for each λ.
   - How to choose λ?
     - Compare the solutions $\tilde{w}_{\text{hat}}$ obtained for different λ with the ground truth $\tilde{w}^*$.
     - Identify the values of λ that achieve both high recall and high precision, and among these, select the one that results in the lowest fitting loss.
     - Finally, display the corresponding $\tilde{w}_{\text{hat}}$ alongside the ground truth $\tilde{w}^*$.

3. [3 pts] You are also required to generate three plots (using a log scale for λ):
   - Data fitting loss vs. λ
   - Precision vs. λ: Plot the precision rate as a function of λ. Precision is defined as:

$$\text{Precision} = \frac{\text{\# of correctly identified nonzeros in } \tilde{w}_{\text{hat}}}{\text{\# of nonzeros in } \tilde{w}_{\text{hat}}}. \tag{2}$$

- Recall vs. λ: Plot the recall rate as a function of λ. Recall is defined as:

$$\text{Recall} = \frac{\text{\# of correctly identified nonzeros in } \tilde{w}_{\text{hat}}}{d}, \tag{3}$$

where d is the number of true nonzero coefficients in $\tilde{w}^*$.

2.3 Solve a Real-world Problem: Water Quality Prediction [30 Pts]

In this assignment, we aim to forecast *pH values* at a particular monitoring station based on a set of **spatiotemporal environmental features**. The dataset consists of daily measurements from 37 water stations, each recording 11 features (e.g., temperature, dissolved oxygen, specific conductance). Your goal is to predict the pH value at the first station (Location ID = 0) using a sparse LASSO model.

In general, there are two ways to construct feature vectors for this task:

- Low-dimensional (local-only) features: Use only the 11 features from a single location (e.g., Location ID $= 0$). In this case, each input vector $x_t \in \mathbb{R}^{11}$ contains local measurements only.
- High-dimensional (spatially-augmented) features: Alternatively, one can leverage information from all 37 spatial locations. Each $x_t \in \mathbb{R}^{407}$ (i.e., a flattened 37×11 matrix).

We will focus on the second, high-dimensional setting. This allows us to explore how Lasso can identify the most informative spatial features and promote sparsity in a large input space.

The dataset consists of temporal measurements collected over multiple days. Specifically, the training data contains $T = 423$ time steps (days), while the test data includes $T = 282$ time steps. The dataset is provided in a `.mat` file and contains the following variables:

Variable Name	Type	Size	Description
`features`	array of strings	1×11	Names of the 11 water quality features.
`location_ids`	array of integers	37×1	IDs of the 37 water stations.
`X_te`	array of matrices	1×282	Test set input data: 282 days of 37×11 spatial feature matrices. Each day represents 37 locations by 11 features.
`X_tr`	array of matrices	1×423	Training set input data: 423 days of 37×11 spatial feature matrices.
`Y_te`	array of matrices	37×282	Test output: pH values for all 37 locations across 282 days.
`Y_tr`	array of matrices	37×423	Training output: pH values for all 37 locations across 423 days.
`location_group`	array of cells	1×3	Spatial groups of stations (not used in this task).

2.3.1 Data Loading and Preprocessing [10 Pts]

We have provided the following code to help you load the dataset in Python:

```
import scipy.io
data_file = "water_dataset.mat"
data = scipy.io.loadmat(data_file)
X_train_raw, X_test_raw = data['X_tr'], data['X_te']
y_train_raw, y_test_raw = data['Y_tr'], data['Y_te']
```

Here, `X_train_raw` and `X_test_raw` are arrays of shape `(1, T)` containing 37×11 matrices at each time step, and `y_train_raw`, `y_test_raw` contain the corresponding pH target values for each of the 37 spatial locations.

You are required to complete and use the function `prepare_dataset()` to preprocess the data into a format compatible with the regression model. Specifically,

- Extract a feature matrix where each row represents the flattened full 37×11 spatial grid at a single time step, and the target is the pH value at a specific location.
- After flattening the inputs and selecting the corresponding target values, you should split the training data into 80% training and 20% validation sets.
- The full list of variables that your `prepare_dataset()` function should return is summarized in Table 1.
  - To help you verify that your code is correct, we have provided print statements that display the shapes of all returned variables. These checks will confirm that the feature and target arrays are properly aligned for training, validation, and testing.
  - Additionally, the code includes a command to print the names of the 11 input features used at each location. This will help you interpret your final Lasso model.
- After completing your `prepare_dataset()` function, you will need to perform a few additional steps to ensure the data is ready for use in your Lasso implementation.
  - **Add bias term:** Use the provided `add_bias_column()` function to augment each feature matrix (`X_train`, `X_val`, `X_test`, and `X_train_full`) with a bias column of all ones. This allows your regression model to learn a bias term w_0.
  - **Flatten the targets:** For convenience and compatibility with NumPy operations and plotting, you may need to flatten each of the target vectors (`y_train`, `y_val`, `y_test`, and `y_train_full`) into 1D arrays.

Preprocessing the dataset correctly is a critical step in this assignment. If you have completed the steps above, congratulations! You now have the data prepared in a standard form suitable for applying regression models such as Lasso. At this point, your dataset aligns with the mathematical model described earlier, and you are ready to proceed with implementation and analysis.

Variable	Shape	Description
`X_train`	(train_size, 407)	Training feature matrix (flattened 37×11 grid), obtained from an 80% split of the original training data.
`X_val`	(val_size, 407)	Validation feature matrix corresponding to the held-out 20% of the original training data.
`X_test`	(282, 407)	Test feature matrix; each row corresponds to one day's full grid of input features.
`y_train`	(train_size, 1)	Target pH values at location `loc` for the training set.
`y_val`	(val_size, 1)	Target pH values at location `loc` for the validation set.
`y_test`	(282, 1)	Target pH values at location `loc` for the test set.
`X_train_full`	(423, 407)	Full training feature matrix before splitting; each row is a flattened 37×11 spatial grid.
`y_train_full`	(423, 1)	Full vector of pH target values at location `loc` for all training days.

Table 1: Description of variables returned by `prepare_dataset()`.

2.3.2 Training, Model Selection, and Testing [20 Pts]

Apply your LASSO solver (set $\epsilon = 10^{-6}$ in algorithm 2) to the real-world water quality dataset. In this part, you will experiment with different values of the regularization parameter λ, and evaluate model performance based on the Root Mean Squared Error (RMSE).

Definition of RMSE: Given predictions $\hat{y}_1, \ldots, \hat{y}_N$ and ground truth values $y_1, \ldots, y_N$,

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}.$$

Note that the RMSE is essentially equivalent to the data fitting loss (the least squares part) used in the LASSO objective, up to scaling and a square root. Although not exactly the same, the RMSE is often used as a more interpretable metric because it has the same units as the output variable.

1. [5 pts] Train with multiple λ: Use your LASSO solver to train models with a range of provided λ values. For each λ, record:

   - The **validation RMSE**
   - The **number of nonzero weights** in the learned model (excluding the bias).

   Generate two plots: validation RMSE vs. λ and sparsity vs. λ.

   Note: For each new λ, use the precious converged solution $\hat{w}$ (from previous λ) as the initial point for ISTA. This warm start can help the algorithm converge faster.

2. [5 pts] Select two candidate λ values:

- Option A: The λ that achieves the lowest validation RMSE
- Option B: The λ that gives the most sparse solution (fewest nonzero weights).

Important: You must use the *validation set only* to select the best λ. The test set should be used strictly for final evaluation after model selection is complete.

3. [5 pts] Retrain on the full training set: For each selected λ, retrain your model using the **entire** training set (training + validation). Record the following for each model:
   - The data fitting loss: $\frac{1}{2N}\|y - X\tilde{w}_{\text{hat}}\|_2^2$
   - The ℓ_1 regularization term: $\lambda\|w_{\text{hat}}\|_1$
   - The total LASSO loss: $L_{\text{lasso}}(\tilde{w}_{\text{hat}}) = \frac{1}{2N}\|y - X\tilde{w}_{\text{hat}}\|_2^2 + \lambda\|w_{\text{hat}}\|_1$
   - The number of nonzero weights (excluding bias)
4. [2 pts] Plot training curves: For both selected λ values, plot the total LASSO loss (log scale) versus iteration to visualize convergence behavior during ISTA.
5. [3 pts] Evaluate on test data: Apply both retrained models to the test set. Compute and report the test RMSE for each. Briefly discuss which model generalizes better to unseen data.

These steps will help you understand how to select a model using validation performance, and how to balance the trade-off between prediction accuracy and sparsity in practice.