INF8245E(Fall 2025) : Machine Learning — Given on: Sep 3rd

Assignment #1 — Due on : Sep 26th, 10 pm

- This is an individual assignment. You are not allowed to discuss the problems with other students.
- Make sure to write your code only in the .py files provided. Avoid creating new files. Do not rename the files or functions as it will interfere with the autograder's ability to evaluate your work correctly. Also, do not change the input or output structure of the functions.
- When Submitting to GradeScope, be sure to submit
  1. A '.zip' file containing all your python codes (in .py format) to the 'Assignment 1 - Code' section on Gradescope.
  2. A 'pdf' file that contains your answers to the questions to the 'Assignment 1 - Report' entry.
- Part of this assignment will be auto-graded by Gradescope. You can use it as immediate feedback to improve your answers. You can resubmit as many times as you want. We provide some tests that you can use to check that your code will be executed properly on Gradescope. These are **not** meant to check the correctness of your answers. We encourage you to write your own tests for this.
- Please provide the title and the labels of the axes for all the figures your are plotting.
- If you have questions regarding the assignment, you can ask for clarifications in Piazza. You should use the corresponding tag for this assignment.
- Before starting the assignment, make sure that you have downloaded all the data and tests related for the assignment and put them in the appropriate locations.
- **Warning:** Throughout the assignment, you will be asked to implement certain algorithms and find optimal values. In the solution you submit, do not simply call a library function which performs the entire algorithm for you, this is forbidden, as it would obviously defeat the purpose. For example, if you were asked to implement the linear regression, do not simply call an outside package (such as scikit-learn) for help.
- You cannot use ChatGPT or any other code assistance tool for the programming part, however if you use ChatGPT to edit grammar in your report, you have to explicitly state it in the report.

Linear and Weighted Ridge Regression (35 Points)

1. In this assignment, you will work with a dataset containing ten features to predict a continuous score for each individual sample. The dataset is provided in four CSV files: `X_train.csv`, `X_test.csv`, `y_train.csv`, and `y_test.csv`.

   Your tasks include deriving and implementing closed-form solutions for different regression models, then comparing their performance on the test set. In **Part 1**, you will implement the required helper functions; in **Part 2**, you will apply them to the dataset. **Make sure to use the exact function names, arguments, and return formats specified in the instructions, otherwise your code will not pass the test cases.**

   (a) (2 Points) Implement a helper function to append a bias term (a column of ones) to the data matrix X. The bias column should be the first column in the augmented matrix. Implement this in `data_matrix_bias()` in **q1_1.py**.

   (b) (5 Points) Implement the function `linear_regression_optimize(X, y)` to compute the ordinary least squares closed-form solution:

   $$w^* = (X^\top X)^{-1} X^\top y$$

   Implement this in `q1_1.py` without using external ML libraries.

   (c) (5 Points) Implement the function `ridge_regression_optimize(X, y, λ)` to compute ridge regression with scalar regularization parameter λ:

   $$w^* = (X^\top X + \lambda I)^{-1} X^\top y$$

   where I is the identity matrix matching $X^\top X$. Implement this in `q1_1.py`.

   (d) (6 Points) **Weighted Ridge Regression Solution:**
   Consider the weighted ridge regression loss:

   $$\mathcal{L}(w) = \|Xw - y\|_2^2 + w^\top \Lambda w$$

   where $X \in \mathbb{R}^{n \times d}$, $y \in \mathbb{R}^n$, and $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_d)$ is a diagonal matrix with non-negative regularization weights.

   - Derive the closed-form solution for the optimal weights w^* minimizing $\mathcal{L}(w)$.
   - Show that:

     $$w^* = (X^\top X + \Lambda)^{-1} X^\top y$$

     Write your detailed derivation in your **.pdf report**.

   (e) (6 Points) Implement the function `weighted_ridge_regression_optimize(X, y,` λ_{vec}`)` that computes weighted ridge regression with diagonal regularization matrix $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_d)$:

   $$w^* = (X^\top X + \Lambda)^{-1} X^\top y$$

   The vector `lambda_vec` has length equal to the number of columns in X (including bias). Implement the `weighted_ridge_regression_optimize(X, y, lambda_vec)` function in `q1_1.py`.

(f) (5 Points) Implement functions `predict(X, w)` which gives you the predicted output, as well as `rmse(y,` $\hat{y}$`)` where:

$$\hat{y} = Xw, \quad RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Implement both in `q1_1.py`.

Hint: Pay special attention to edge cases, such as singleton dimensions and situations where the matrix is not invertible, when implementing the functions in this section.

2. (6 Points) In `q1_2.py`:

   - Load training and test datasets.
   - Append bias to all X matrices.
   - Train and evaluate all three models:
     - Ordinary least square regression
     - Ridge Regression with $\lambda = 1.0$
     - Weighted Ridge Regression with $\lambda\_vec = [0.01, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3]$
   - Compute and report RMSE on the test set for each model.
   - Plot predicted vs actual final performance scores for each model as scatter plots.

   Include the RMSE values and plots in your **.pdf report**. You could use functions you implemented in Part 1.

Cross-Validated Model Selection (30 Points)

1. You are given the same dataset as in Problem 1: `X_train.csv`, `y_train.csv`, `X_test.csv`, and `y_test.csv`. Your task in this problem is to select the best Ridge Regression model (the simple, non-weighted version you implemented in part (c)) by performing k-fold cross-validation using different evaluation metrics. You must implement all methods from scratch using only NumPy.

   (a) (9 Points) In `q2_1.py`, implement a function `cv_splitter(X_train, y_train, k)` that takes the training dataset (`X_train, y_train`) and an integer k, and returns k train-validation splits.

   You must shuffle the data before partitioning into folds. The output of this function should be a **list of length** k, where each element is a **tuple containing four arrays**: `X_train_fold`, `y_train_fold`, `X_val_fold`, and `y_val_fold`.

   Requirements:

   1. Shuffle the data randomly before splitting into folds.
   2. Ensure that each row of the dataset appears exactly once as part of a validation fold.
   3. Distribute any remainder samples in the first fold if the dataset is not divisible by k.
   4. Use only NumPy functions; do not use scikit-learn utilities.

   (b) (6 Points) Implement the following functions in `q2_1.py`, assuming y and $\hat{y}$ are one-dimensional NumPy arrays of equal length:

   1. `MAE()` - Mean Absolute Error (MAE):

   $$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

   2. `MaxError()` - Maximum Absolute Error:

   $$\text{MaxError} = \max_i |y_i - \hat{y}_i|$$

   3. Import the `rmse()` function you implemented in `q1_1.py`.

   (c) (7 Points) Implement a function `cross_validate_ridge(X, y,` *lambda*_list`, k, metric)` in `q2_1.py` that:

   - Trains a Ridge Regression model (closed-form solution) on $k-1$ folds for each $\lambda \in \lambda\_list$.
   - Computes the prediction error on the held-out fold using the given metric (as a Python function).
   - Averages the metric across folds for each λ.
   - Returns the λ with the best average validation score.

- The `metric` argument can only take one of the following values: `MAE, MaxError, RMSE`. Be sure to use the same names in your function; otherwise, test cases will not pass.
- You can use the `ridge_regression_optimize()` function you implemented in `q1_1.py`

2. (8 Points) In `q2_2.py`:

   Use your function to perform 5-fold cross-validation over the following hyperparameter values on the dataset from previous section:

$$\lambda \in \{0.01, 0.1, 1, 10, 100\}$$

   For each of the three evaluation metrics (MAE, MaxError, and RMSE), report:

   - The best λ based on mean validation score
   - The corresponding mean score across folds

   Include your results in a table in your **.pdf report**.

Gradient Descent for Ridge Regression with Learning Rate Schedules (35 pts)

1. In this question, you will implement gradient descent for Ridge Regression from scratch.

   From the lectures, we know that the ridge regression gradient can be calculated as:

$$\nabla_{\mathbf{w}}\mathcal{L} = -\frac{2}{n}X^{\top}(y - X\mathbf{w}) + 2\lambda\mathbf{w}$$

   (a) (4 pts) In `q3_1.py`, implement a function named `ridge_gradient(.)` to compute the Ridge regression gradient. The function must take the arrays X, y, and w, and the float `lambda` and returns the gradient using the above formula. annealing.

   (b) (3 pts) **Learning Rate Schedule.** In class, you have already seen the case of a constant learning rate, where the step size remains fixed throughout training. While simple, this approach can lead to slow convergence or instability if the chosen value is not well-tuned. To address this, we often use a learning rate schedule, where the learning rate changes over time. The schedule allows for larger steps early in training (to quickly move toward good regions of the parameter space) and smaller steps later (to refine convergence). Common schedules include exponential decay and cosine

   In `q3_1`, implement a function named `learning_rate_exp_decay(.)` for the exponential decay learning rate schedules:

$$\eta_t = \eta_0 \cdot \exp(-kt)$$

   The function should take the initial learning rate η_0, current iteration step t, and the decay constant k and return the learning rate at each step.

   (c) (3 pts) In `q3_1`, implement a function named `learning_rate_cosine_annealing(.)` for the cosine annealing learning rate schedules:

$$\eta_t = \eta_0 \cdot \frac{1}{2}(1 + \cos(\pi t/T))$$

   The function should take the initial learning rate η_0, current iteration step t, and the total number of iterations T, and return the learning rate at each step.

   (d) (7 pts) In `q3_1`, implement a single gradient descent step function that updates the weights using the gradient and learning rate. Name your function as `gradient_step(.)`. Your function should take X, y, w, λ and η as input and return the updated w.

   (e) (8 pts) In `q3_1.py`, implement the function `gradient_descent_ridge(.)` that performs the full gradient descent training loop including:

   - Gradient computation
   - Weight update step
   - Selected learning rate schedule

The function should take: X, y, λ, initial learning rate η_0, total number of iterations T, schedule type, and decay constant k. Note that the schedule type should be either 'constant', 'exp_decay', or 'cosine'.The function should have two outputs: The updated final weights w as well as a vector L of size T that includes the training loss at every iteration. We need this vector to track the training loss.

2. (10 pts) In `q3_2.py`, implement the ridge regression with gradient descent on the dataset from the previous sections. Train using each learning rate schedule with the same initial conditions ($\eta_0 = 0.001$, $k = 0.001$, $T = 100$, $\lambda = 1.0$). For each schedule, plot the training loss and report the RMSE error on the test set.

   In your **.pdf report**, describe in 3-4 sentences which schedule converged fastest and achieved the best generalization (lowest test loss).