INF8245E(Fall 2024) : Machine Learning — Given on: Sep 4^{th}

Assignment #1 — Due on : Sept 27, 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.

Simple Linear Regression (20 Points)

1. In this assignment, you will take a simple dataset and implement linear and ridge regression by solving for their **analytical** solutions. You will then perform a simple hyperparameter search to determine regression coefficients that best suit the data. In the last part, you will solve the linear and ridge regression problems using gradient descent.

   We'll be using the Salary Dataset in this assignment. Each employee's salary is determined based on their experience as well as their score on a professional test. In this dataset, you are given the *experience* (in years), *test_score* and should predict the *salary* (in dollars).

   The data is split into train and test sets and given in a .csv file. The *X_train.csv* and *X_test.csv* contain the data matrix and the *y_train* and *y_test* contain the ground truth salary. The provided code loads the data properly in the format you need. In this case, we load our data as a data matrix X where

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nd} \end{bmatrix}$$

   In order to implement the linear regression from scratch, we first need to implement the following helper functions. Please implement them in the **q1_1.py** file which is given.

   (a) (3 Points) Write a helper function that appends the bias to the data matrix X. The columns should be appended to the beginning of the matrix. Provide your answer in the designated space in the *data_matrix_bias* function in **q1_1.py**.

   (b) (2 Points) Implement a function that computes $y = Xw$ given a data matrix X and parameters w. Provide your answer in the designated space in the *linear_regression_predict* function in **q1_1.py**.

   (c) (7 Points) Implement a function that given y and X, computes the optimal w using the closed form solution. Provide your answer in the designated space in the *linear_regression_optimize* function in **q1_1.py**. **Do not use library functions which given y and X return the optimal w.**

   (d) (3 Points) Write a function that calculates the root mean square error (RMSE) of a predicted output vector $\hat{y}$ relative to the ground truth vector y. Provide your answer in the designated space in the *rmse* function in **q1_1.py**.

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

2. (5 Points) Using the functions you have implemented, fit and evaluate a linear regression model in **q1_2.py**. You should find the optimal parameters using the training data. We have provided part of the code and you need to complete it. Report the RMSE value

on the test data, and plot the predicted salary and the actual salary on the test set in one graph as a function of both variables: experience and test score. You can plot two separate scatter plots, one per variable. Show the resulting plots in your **.pdf report** file.

Ridge Regression and Cross Validation (40 Points)

1. (10 Points) If you recall, ridge regression is a regularized version of linear regression where loss is calculated as

$$\tilde{L}(X, y, w) = L(X, y, w) + \lambda \cdot \|w\|_2^2$$

where L is the loss for linear regression and w is the current weight vector.

Implement the closed form solution for Ridge Regression, also called Linear Regression with an L2 regularizer. Please provide your code in the *ridge_regression_optimize* function in the **q2_1.py** file.

2. (15 Points) Here, we'll perform a k-fold cross-validation over the samples to estimate the best value of λ that enables the best transfer from the training dataset to the testing dataset. Since there are many ways to perform the splitting, we'll be doing it in a very simple manner. To avoid any random seeding issues, we'll be creating each fold's train/validation sets deterministically.

First, calculate a 'fold_size' by determining the best size which splits the training set evenly between the desired number of folds. Leave this value as a decimal as rounding will be taken care of later.

Then for the i-th fold, simply use the $fold\_size \times i$ to $fold\_size \times (i + 1)$ entries of the training data as your validation set. To avoid confusion, if either value is not an integer, simply round the values to the nearest one.

Please provide your implementation in the *cross_validation_linear_regression* function which can be found in the **q2_2.py** file. Your code should take the training data, hyperparameter value(s), and the number of folds as inputs. The output should include the average RMSE across all folds for each hyperparameter value, along with the best hyperparameter value and its corresponding RMSE. Note that the test set should not be used in this process.

3. (5 Points) Using the functions you implemented, perform a hyperparameter search on the regularization term of the Ridge Regression. Plot the RMSE as a function of the hyperparameter value. How can your observations be theoretically explained? Please provide your code in the **q2_3.py** file and provide your plots and explanation in the **.pdf report** file.

4. (5 Points) For the ridge regression to work, the matrix $(XX^T + \lambda I)^{-1}$ has to exist. Assuming $\lambda > 0$, show that $XX^T + \lambda I$ is an invertible matrix. Provide your answer in the **.pdf report** file.

5. (5 Points) For regression problems, we may prefer to leave the bias term unregularized. One approach is to change the loss function so that the bias is separated out from the other parameters and left unregularized. Another approach that can achieve approximately the same thing is to use a very large number B, rather than 1, for the extra bias dimension. Explain why making B large decreases the effective regularization on the bias term, and how we can make that regularization as weak as we like (though not zero). Provide your answer in the **.pdf report** file.

Full-Batch Gradient Descent (40 Points)

1. (15 Points) In this section, we will optimize our Regression model with gradient descent instead of the closed-form solution. As the first step, we should calculate the gradients. Implement functions that compute the gradient for both simple and ridge regression. Implement your code in the *compute_gradient_simple* and *compute_gradient_ridge* functions in the **q3_1.py** file.

2. (15 Points) Now write a function for solving a given regression task using full-batch gradient descent instead of the analytical solution. In this case, the function will take in a parameter 'reg_type' to determine the type of regression we're using, a 'hyperparameter' in case we are doing regularization, a 'learning_rate' parameter which sets the size of the steps we make in the gradient directions and 'num_epochs' which defines how many gradient descent steps we wish to perform. Provide your answer in the *gradient_descent_regression* function of the **q3_2.py** file.

3. (5 Points) Now run the above function for both linear and ridge regression on the given dataset. Initialize the initial weight vector using 'np.random.normal'. Write the code to find the solution weights for each type of regression. After each epoch, plot the training loss. Write your code in **q3_3.py** and provide your plots and explanations in the **.pdf report** file.

4. (5 Points) Now, try different values for 'learning_rate' hyperparameter. Plot the training loss after each epoch, and the RMSE versus the 'learning_rate' for both linear and ridge regression on the test dataset. Explain how changing this hyperparameter affects the training process. Write your code in **q3_4.py** and provide your plots and explanations in the **.pdf report** file.