CS385/CSD3185/CSD3186: Assignment 1

Topics Covered

- Linear Regression
- Gradient Descent Algorithm
- Mean Squared Error (MSE)

Objectives

1. Apply **gradient descent algorithm** to predict the housing prices in Boston city.
2. Get familiarized on determining values of hyperparameters and evaluating the performance of a machine learning algorithm.

Deliverables:

Your submission for this assignment should be **ONE zip** file, which contains the completed *assignment1.ipynb* notebook.
file. Rename your completed zip file like this: **coursecode_A1_your_full_name.zip**,
e.g. CS385_A1_john_doe.zip

Dataset Overview

The **Boston_housing_price.csv** contains information about housing values in the suburbs of Boston. Each record represents a specific suburb, with **13 input features** and **1 output variable** to predict the **median house price** (in thousands of dollars). **Note that some columns in the dataset contain null values, and no manual modifications are allowed to alter the original dataset.**

Name	Description
CRIM	Per capita crime rate by town
ZN	Proportion of residential land zoned for lots over 25,000 square feet
INDUS	Proportion of non-retail business acres per town
CHAS	Charles River dummy variable (1 = tract bounds the river, 0 = otherwise)
NOX	Nitric oxides concentration (parts per 10 million)
RM	Average number of rooms per dwelling
AGE	Proportion of owner-occupied units built prior to 1940
DIS	Weighted distances to five Boston employment centers
RAD	Index of accessibility to radial highways
TAX	Full-value property-tax rate per $10,000
PTRATIO	Pupil-teacher ratio by town
B	1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
LSTAT	Percentage of the population considered lower status

MEDV (output)	Median value of owner-occupied homes (in $1000's)

Your Tasks

You are required to follow the steps in the *Assignment1.ipynb* and use the function names, input types, and output types as outlined. **Tip: You may modify the model codes in Linear_Regression_lectorial.ipynb as needed.*

1. Train-Test Split:
    - Separate X and y then convert them to numpy arrary for slicing.
    - Split dataset into Train (404 rows) and Test (102 rows)

2. def z_score_norm(X): Z-Score Normalization standardizes each feature in X by applying the **z-score formula**:

$$X_{\text{norm}} = \frac{X - \mu}{\sigma}$$

Inputs: X - numpy.ndarray, the input feature values
Return/Outputs:
X_norm - the normalized feature values
(mean, std): the mean and standard deviation of each feature

3. def loss_MSE(X, Y, b, W) : Mean Squared Error Calculation

$$MSE = \frac{\sum_{i=1}^{N}[y_i - (wx_i + b)]^2}{N}$$

Inputs: X - numpy.ndarray, the input feature values
Y - numpy.ndarray, the true output values
b - float/int, bias
W - numpy.ndarray, weights for features
Return/Outputs: MSE – float

4. def update_bias_weights(X, Y, b, W, learning_rate): This function updates the bias and weights iteratively to minimize the loss using **gradient descent**

Gradient of weights: $\frac{\partial L}{\partial w} = \frac{\sum_{i=1}^{N} 2[y_i - (wx_i + b)](-x_i)}{N}$

Gradient of bias: $\frac{\partial L}{\partial b} = \frac{\sum_{i=1}^{N} 2[y_i - (wx_i + b)](-1)}{N}$

Updated weights: $w_{new} = w_{old} - \frac{\partial L}{\partial w} \cdot \alpha$

Updated bias: $b_{new} = b_{old} - \frac{\partial L}{\partial w} \cdot \alpha$

Inputs: X - numpy.ndarray, the input feature values
Y - numpy.ndarray, the true output values
b - float/int, bias

W - numpy.ndarray, weights for features
learning_rate - float, the learning rate

Return/Outputs:
b: updated bias
W: updated weights

5. (a) def train(X, Y, b, W, learning_rate, learning_iterations): This function performs multiple iterations of gradient descent to minimize the loss and returns a history of the loss values, final bias, and weights.

$$\text{loss\_history} = [MSE_1, MSE_2, \ldots, MSE_{\text{learning\_iterations}}]$$

Inputs: X - numpy.ndarray, the input feature values
Y - numpy.ndarray, the true output values
b - float/int, bias
W - numpy.ndarray, weights for features
learning_rate - float, the learning rate

Return/Outputs:
loss_history - list of loss values
b - final bias
W - final weights

(b) def optimal_learning_rate(X, Y, b, W, learning_rates, learning_iterations): This function finds the best learning rate to minimize the MSE

Steps:

- Train the model using different learning rates (α\alphaα) and record the final MSE for each.
- Select the learning rate with the lowest MSE.

Inputs:
X - numpy.ndarray, the input feature values
Y - numpy.ndarray, the true output values
b - float/int, initial bias
W - numpy.ndarray, initial weights for features
learning_rates - list, a list of learning rates to evaluate
learning_iterations - int, the number of training iterations for each learning rate

Return/Outputs:
best_lr - float, the learning rate with the lowest final MSE
best_loss - float, the final MSE for the best learning rate
results – {α: MSE}: A dictionary where α (learning rate) is the key, and the corresponding MSE (Mean Squared Error) is the value

6. Predict on test data: Perform Z-Score normalization on the test set using the mean and standard deviation values from the training set. Subsequently, evaluate the model on the test set using the final bias and weights from the best learning rate. Calculate the MSE on the test set.

Rubrics (Total: 100 Points)

Tasks	Points
Splitting the data	**5**
Normalization of data	**15**
Mean Squared Error (MSE) function	**15**
Bias and weight computation	**25**
Training and optimal learning rate function	**40**