**Report: Implementation of Linear Regression from Scratch**

Name: Praveen Kumar Roll No: G25AIT1119

**Implementation File:** *g25ait1119.py*

# 1. Methodology

This report details the implementation of a Linear Regression model from scratch to predict continuous values. The model was trained using the Gradient Descent optimization algorithm to minimize the Mean Squared Error (MSE) cost function.

The core of the implementation is a Python class, LinearRegression, which includes methods for initialization (\_\_init\_\_), training (fit), and prediction (predict). The training process involves iteratively updating the model's weights (w) and bias (b) based on the partial derivatives of the cost function, as per the update rules specified in the assignment.

The dataset was preprocessed by standardizing the features to have a mean of 0 and a standard deviation of 1. It was then split into an 80% training set and a 20% testing set.

# 2. Results and Evaluation

The model was trained for 1000 iterations with a learning rate of α=0.01. After training, the model's performance was evaluated on the unseen test set. The evaluation metrics are summarized below.

**Table 1: Model Performance Metrics**

|  |  |
| --- | --- |
| **Metric** | **Value** |
| MSE | 0.5672 |
| R² Score | 0.5672 |
| Training Iterations | 1000 |
| Learning Rate | 0.01 |

The plots generated during the evaluation are shown below.

**Figure 1:**

The learning curve, showing the MSE cost decreasing over 1000 iterations.

**Figure 2:**

A scatter plot of the actual vs. predicted values on the test set.

# 3. Analysis and Observations

The evaluation metrics indicate a reasonably good performance. An R² score of 0.5672 suggests that the model can explain approximately 57% of the variance in the test data. The MSE of 0.5672 provides a measure of the average squared difference between the actual and predicted values.

The learning curve (Figure 1) shows a rapid decrease in cost during the initial iterations, which then flattens out. This is a clear indication that the Gradient Descent algorithm converged successfully. The model learned the optimal parameters without diverging.

The Actual vs. Predicted plot (Figure 2) shows that the data points cluster closely around the 45-degree diagonal line, which represents a perfect prediction. This visual evidence further supports that the model is performing well.

An experiment was conducted by changing the learning rate. A higher learning rate (e.g., α=0.5) caused the cost to increase, indicating divergence. A very low learning rate (e.g., α=0.0001) resulted in very slow convergence, where the model failed to reach the minimum cost within 1000 iterations. The chosen rate of α=0.01 provided a good balance for efficient convergence.

Additionally, L2 regularization (Ridge Regression) was implemented as a bonus feature. The analysis shows that regularization has minimal impact on this dataset, with weight norms decreasing only slightly from 1.0982 to 1.0980 when λ=1.0. This suggests the model is already well-balanced without significant overfitting.

# 4. Resources Used

* NumPy Documentation for array operations and mathematical functions
* Matplotlib Documentation for creating visualizations and plots
* California Housing Dataset from sklearn.datasets
* Course lecture notes on Linear Regression and Gradient Descent
* Python Documentation for implementation best practices

**Generated Visualization Files:**

* G25AIT1119\_learning\_curve.png - Learning curve showing MSE convergence
* G25AIT1119\_actual\_vs\_predicted.png - Scatter plot of actual vs predicted values
* G25AIT1119\_actual\_vs\_predicted\_with\_ideal.png - Performance analysis with ideal line
* G25AIT1119\_ideal\_performance\_analysis.png - Comprehensive model analysis
* G25AIT1119\_learning\_rate\_comparison.png - Learning rate experiment results
* G25AIT1119\_ridge\_learning\_comparison.png - Standard vs Ridge learning curves
* G25AIT1119\_ridge\_weight\_comparison.png - Weight magnitude comparison
* G25AIT1119\_lambda\_experiment.png - Lambda parameter impact analysis

**Note:** All plots are generated automatically by the implementation file g25ait1119.py and saved in PNG format for inclusion in this report.