Report: Fundamental Machine Learning Implementations

Youri HALMAERT

June 12, 2025

1 Introduction

This report summarizes the work performed in two Jupyter notebooks, `lin_reg_solver.ipynb` and `log_reg_solver.ipynb`, which focus on implementing foundational machine learning algorithms: Linear Regression and Logistic Regression, respectively. A core theme throughout both exercises is the use of Gradient Descent, a powerful iterative optimization algorithm, to find the optimal parameters for these models, rather than relying on closed-form solutions. The labs provide a hands-on approach to understanding the underlying mechanics of these algorithms, including data preprocessing, loss function definition, and parameter updates.

2 Part 1: Linear Regression with Gradient Descent

The first part of this work delves into the implementation of Linear Regression. Unlike typical approaches that might use the normal equation (a closed-form solution), this lab emphasizes the use of gradient descent for parameter estimation.

2.1 Model and Loss Function

The objective of linear regression is to model the relationship between a dependent variable y and one or more independent variables x by fitting a linear equation. The assumed linear model is given by:

$$\hat{y} = w^T x + w_0$$

where $\hat{y}$ is the predicted output, w is the vector of weights for the features, x is the input feature vector, and w_0 is the bias term (or intercept).

To quantify the model's performance, the sum of squares loss function (also known as Mean Squared Error) is employed:

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

Here, N is the number of samples, y_i is the actual output for the i-th sample, and $\hat{y}_i$ is the predicted output for the i-th sample. The goal of gradient descent is to minimize this loss function.

2.2 Gradient Descent Implementation

The `gradient_descent` function was implemented to iteratively update the model's weights. The key steps within each iteration are:

1. **Estimate** $\hat{y}$: The predicted output $\hat{y}$ is calculated using the current weights and input data.
2. **Estimate the Gradient**: The gradient of the loss function with respect to the weights is computed:

   $$\nabla L = -\frac{2}{N} X^T (y - \hat{y})$$

   where X is the input matrix (augmented with a column of ones for the bias term).
3. **Check Stopping Criterion**: The algorithm stops when $\|\nabla L\| < \epsilon$.
4. **Update Weights**: The weights are updated as:

   $$w \leftarrow w + \frac{2\alpha}{N} X^T (y - \hat{y})$$

2.3 Results

The implementation was tested with $\alpha = 0.0001$, max iterations = 1000, and $\epsilon = 0.00001$. The model successfully converged, demonstrating the effectiveness of gradient descent.

3 Part 2: Logistic Regression from Scratch

The second part focuses on implementing Logistic Regression using gradient descent.

3.1 Data Generation

The data was generated using `sklearn.datasets.make_classification` with parameters:

- `n_features = 10`
- `n_informative = 6`

- `n_redundant = 0`
- `n_repeated = 0`

Number of useless features:

$$10 - (6 + 0 + 0) = 4$$

3.2 Feature Scaling

Features were standardized using:

$$x_i' = \frac{x_i - \mu}{\sigma}$$

3.3 Logistic Regression Class

Key components:

3.3.1 Sigmoid Function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

3.3.2 Loss Function (Cross-Entropy)

$$L(w) = -\sum_{i=1}^{N} [y_i \log(\sigma(X_i w)) + (1 - y_i) \log(1 - \sigma(X_i w))]$$

3.3.3 Gradient Update

$$w \leftarrow w + \frac{\alpha}{N} X^T (y - \sigma(Xw))$$

3.4 Training

- Learning rate (α) = 0.01
- Iterations = 500
- Train-test split = 90%-10%

4 Conclusion

This work demonstrated successful implementations of Linear and Logistic Regression using gradient descent, highlighting core concepts like loss functions, gradient updates, and feature preprocessing.