Evaluation of Multiple Linear Regression Implementations Using the California Housing Data

Vaishnavi
24124048
Mathematics and Computing

Objective This project involves implementing multivariable linear regression using three different approaches:

- **Part 1**: Pure Python (using core language features and basic lists)
- **Part 2**: NumPy (vectorized operations)
- **Part 3**: Scikit-learn (`LinearRegression` class)

The goal is to compare the performance of these methods in terms of convergence time, regression metrics, and computational efficiency.

Palavras-chave. Instruções, LaTeX, Trabalhos Completos, SBMAC, CNMAC (entre 3-6 palavras-chave)

1 Introduction

This project aims to implement multivariable linear regression using three different approaches: pure Python (with minimal use of external libraries), optimized NumPy-based vectorization, and the high-level interface provided by Scikit-learn. The objective is to assess and compare the performance, convergence behavior, and accuracy of each method under a uniform experimental setup.

The dataset used is the California Housing dataset, which has been preprocessed to include normalized features and engineered variables such as one-hot encoded categorical attributes and price-to-income ratios. All implementations were trained and evaluated on this uniformly processed dataset.

2 Methodology

Handling Missing Values

The `total_bedrooms` column had missing values, which were filled using the median value of the column. This approach preserves the data distribution without being affected by outliers.

One-Hot Encoding

The categorical feature `ocean_proximity` was converted into multiple binary columns using one-hot encoding. This allowed the model to process location data without assuming any order or hierarchy among categories.

Feature Engineering

A new feature, `price_to_income_ratio`, was created by dividing the median house value by the median income. This captures housing affordability, providing a useful signal for predicting housing prices.

3 Evaluation Metrics

We evaluated the models on both the training and testing sets using the following metrics:

- **Mean Absolute Error (MAE)**
- **Root Mean Squared Error (RMSE)**
- **R-squared Score (R^2)**

In addition, we measured:

- **Convergence Time**: For the iterative gradient descent algorithms (Parts 1 and 2).
- **Cost Convergence**: Visualization of cost function over epochs for Parts 1 and 2.
- **Metric Comparison**: Bar plots to contrast MAE, RMSE, and R^2 across all three methods.

4 Result

Tabela 1: Training Time and Regression Metrics for Different Implementations

Implementation	Training Time (s)	MAE	RMSE	R^2
Training Set				
Pure Python	51.9708	29030.9602	46827.1826	0.8360
NumPy	0.3128	29390.7532	47437.7053	0.8305
Scikit-learn	0.2030	27835.5500	44849.4900	0.8495
Test Set				
Pure Python	–	29132.2050	49530.2820	0.8128
NumPy	–	30057.2458	47265.5186	0.8340
Scikit-learn	–	28492.4600	47868.4600	0.8251

5 Conclusion

- The **Pure Python** model is computationally expensive but valuable for understanding the fundamentals.
- **NumPy's** vectorization offers a huge speedup with only a minor drop in accuracy.
- **Scikit-learn** provides the best balance of speed and accuracy, making it suitable for real-world applications.

6 Visualization of Evaluation Metrics