Comparative Study: Multivariable Linear Regression Implementations

Aniket Mandal
IIT Varanasi

1 Convergence Time

To ensure a fair comparison, all three models—Pure Python, NumPy, and scikit-learn—were initialized with the same parameters and trained on the same California Housing dataset split (16,346 training samples and 4,087 validation samples).

Implementation Fit Time (seconds)

Implementation	Convergence Time (s)
Pure Python	19.5795
NumPy-based	0.0629
Scikit-learn	0.0802

Observation: The NumPy implementation was significantly faster than the pure Python one, thanks to vectorized matrix operations. The scikit-learn model, though slightly slower than NumPy, remained highly efficient due to its compiled and optimized backend (likely using Cython or Fortran under the hood).

2 Performance Metrics

We evaluated the predictive performance using three standard regression metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), and the R^2 Score.

Metric	Pure Python	NumPy	Scikit-learn
MAE	51404.05	55162.34	51372.67
MSE	5.09×10^9	5.65×10^9	4.92×10^9
R^2 Score	0.6278	0.5870	0.6401

Interpretation

- Scikit-learn achieved the lowest MAE and MSE, and the highest R^2, making it the most accurate of the three.
- Pure Python closely followed sklearn in performance but lagged drastically in speed.
- NumPy, although fastest in convergence, sacrificed some accuracy, likely due to differences in optimization and learning rate tuning.

3 Visualization

A. Cost Function Convergence

Plots were generated for Pure Python and NumPy to visualize how the cost decreased over iterations.

- Pure Python showed a smoother but slower descent in the cost curve.
- NumPy displayed a steeper initial descent, quickly approaching a minimum, aligning with its rapid convergence.

(Include these plots from your notebook in your report.)

B. Metric Comparison Charts

A bar chart comparing MAE, MSE, and R^2 for all three models makes the performance gaps easily interpretable.

(Insert the regression metrics chart here.)

4 Analysis and Discussion

Key Differences and Similarities

- All models trained on the same preprocessed dataset, ensuring fairness in results.
- NumPy's speed advantage came from leveraging vectorized operations over iterative loops.
- Scikit-learn used an internal solver (likely 'auto' or 'lbfgs' for `LinearRegression`) which converged faster and more accurately due to robust optimization algorithms.

Why the Differences?

- Pure Python is inherently slower due to its loop-based computations, which are not optimized for high-volume numerical data.
- NumPy leverages BLAS and LAPACK backends for matrix computations, drastically improving runtime.
- Scikit-learn incorporates data normalization, bias optimization, and parallelization, making it both accurate and efficient.

Scalability & Efficiency

Model	**Scalability**	**Efficiency**
Pure Python	Poor	Low
NumPy	Good	Very High
Scikit-learn	Excellent	High

- For large-scale data, Pure Python is not practical.
- NumPy is a strong middle-ground for understanding internals while maintaining scalability.
- Scikit-learn is production-ready and ideal for real-world ML applications.

Influence of Initialization & Learning Rate

- All models used the same initial weights and learning rate where applicable (except sklearn which does internal optimization).
- NumPy and Pure Python were sensitive to learning rate tuning. Too high a rate led to divergence; too low caused slow convergence.
- Sklearn automatically manages these parameters, providing more consistent results.

5 Conclusion

Each model has its strengths:

- **Pure Python:** Educational, but inefficient.
- **NumPy:** Fast and educational, great for research-level prototyping.
- **Scikit-learn:** The best mix of speed, accuracy, and ease of use.

Thus, while sklearn is recommended for real-world regression tasks, experimenting with NumPy and Python implementations offers deep insight into model internals and optimization behavior.