Optimization-based statistics with `pyensmallen`

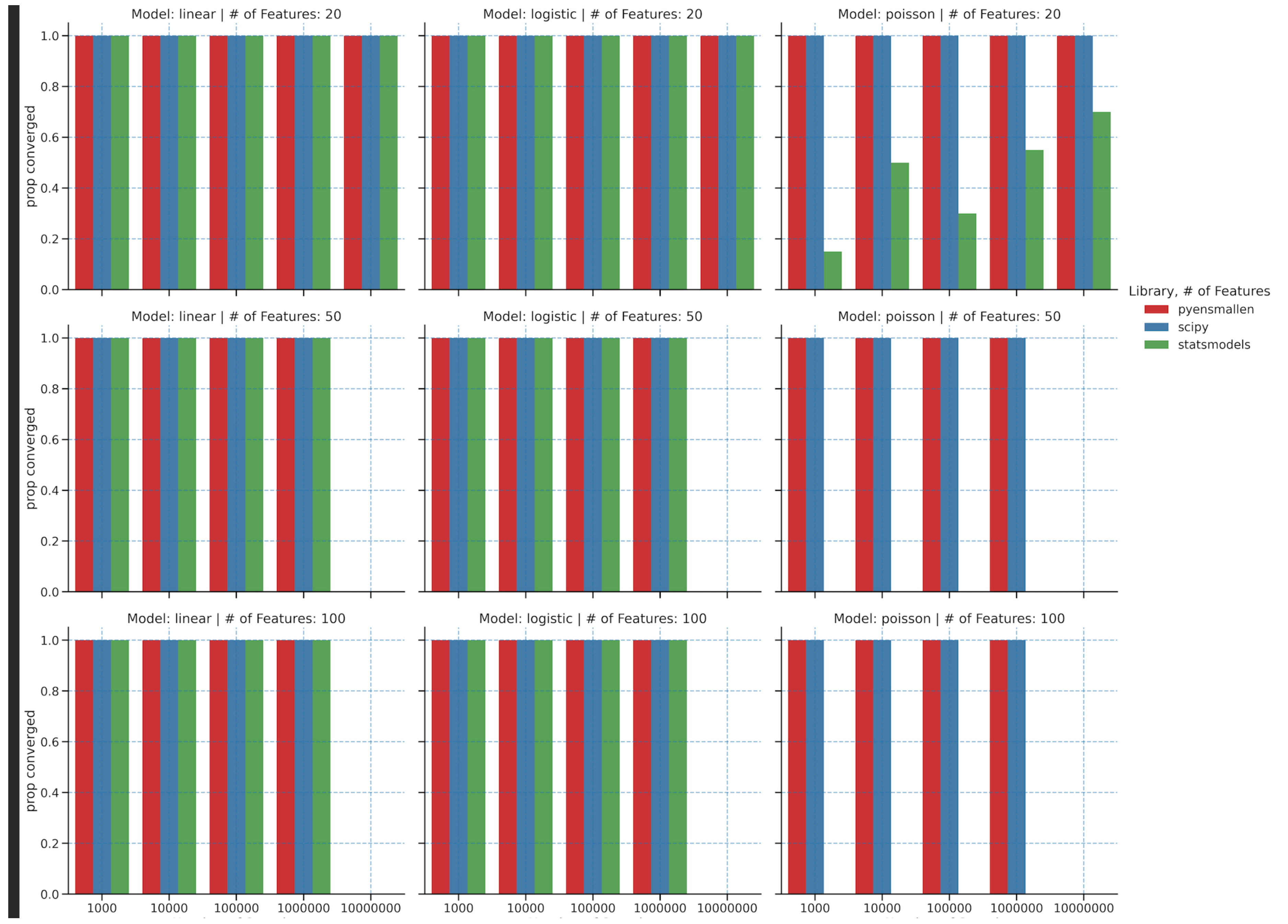
Apoorva Lal

AWS

The Challenge: Scaling Statistical Computing

Modern statistical applications frequently involve large datasets with millions of observations.

Many popular Python libraries for statistical modeling, such as SciPy and statsmodels, were not designed for these scales. This results in excessive computation times for large problems, poor convergence properties, and limits the use of computationally intensive methods like the nonparametric bootstrap.



convergence rates of `pyensmallen`, `scipy`, and `statsmodels` across linear, logistic, and Poisson regression models with growing sample size (x-axis) and problem size (rows). 'statsmodels' implementation of Poisson fails to converge with alarming regularity.

Our Solution: `pyensmallen`

`pyensmallen` solves this problem by providing Python bindings to the highly optimized, header-only C++ `ensmallen` library. [1] It leverages high-performance linear algebra via the Armadillo library. It can be used in combination with JAX [2] gradients for large problems with many parameters.

This enables Python users to access powerful, fast optimization algorithms for statistical estimation, bridging the gap between high-level prototyping and high-performance execution.

Optimization for M-Estimation

Many statistical estimators can be framed as M-estimators, which seek to find parameters θ that minimize an objective function $Q_N(\theta)$:

$$\hat{\theta} = \arg\min_{\theta\in\Theta} Q_N(\theta) = \frac{1}{N}\sum_{i=1}^{N} m(y_i, x_i, \theta)$$

Examples include Maximum Likelihood Estimation (MLE), M-Estimation, and Generalized Method of Moments.

Unconstrained Optimization: L-BFGS

For smooth, unconstrained problems like MLE, `pyensmallen` offers L-BFGS, a quasi-Newton method. It iteratively approximates the inverse Hessian matrix to find an optimal search direction. The parameter update follows:

$$\theta_{k+1} = \theta_k - \alpha_k H_k \nabla Q(\theta_k)$$

where H_k is the approximate inverse Hessian and α_k is the step size. L-BFGS is highly effective due to its low memory footprint and fast convergence.

Constrained Optimization: Frank-Wolfe

For optimization over a constrained set $\mathcal{S}$, the Frank-Wolfe (conditional gradient) algorithm is available. It avoids projection by iteratively solving a linear approximation over the constraint set. At each step k:

1. Find s_k that minimizes the linear approximation:
$$s_k = \arg\min_{s\in\mathcal{S}} s^T \nabla Q(\theta_k)$$
2. Update the parameters by moving towards s_k:
$$\theta_{k+1} = (1-\gamma_k)\theta_k + \gamma_k s_k$$

This is ideal for problems with lp-ball or simplex constraints. This optimizer is used in an applications library `synthlearners`, which focusses on causal inference with panel data and provides fast synthetic control and synthetic-difference-in-differences implementations powered by ensmallen.

Availability

`pyensmallen` is open-source and available on PyPI and GitHub. [2, 4]

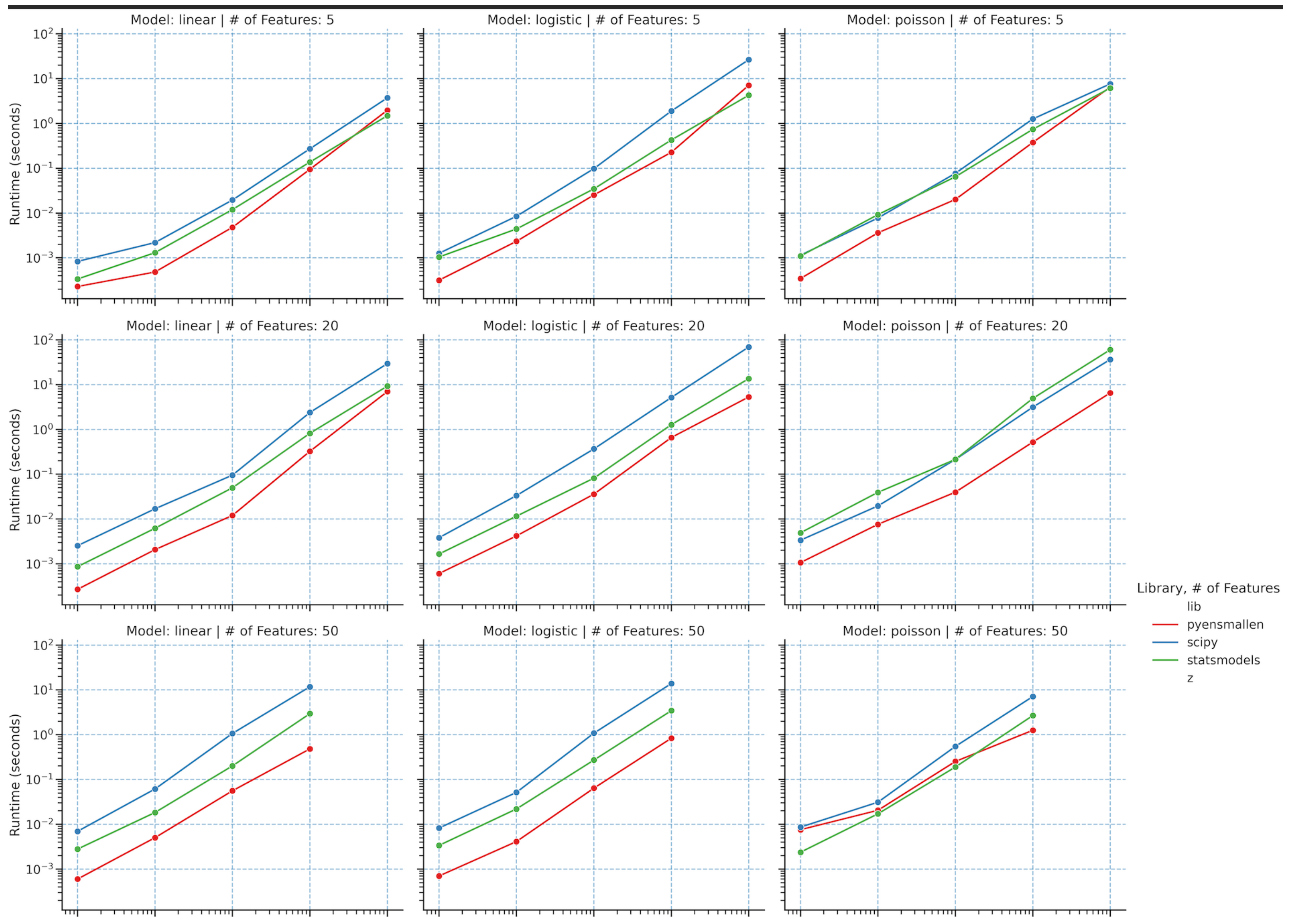
```
uv pip install pyensmallen
```

Benchmark Highlights

Our benchmarks show `pyensmallen` consistently outperforms SciPy and statsmodels, with the advantage growing as dataset size increases.

- **Linear Regression :** 5-11x faster than SciPy & 3-4x faster than statsmodels for 10M obs.
- **Logistic Regression :** 11-15x speedup over SciPy & 2-4.5x faster than statsmodels for 1m+ obs
- **Poisson Regression:** Up to 13x faster than SciPy & 30x faster than statsmodels.

Performance Comparison



Performance comparison of `pyensmallen`, `scipy`, and `statsmodels` across linear, logistic, and Poisson regression models with k=5 and k=20 features. Full benchmarks are reproducible from the code repository.

References

1. Bhardwaj, S., Curtin, R. R., Edel, M., Mentekidis, Y., & Sanderson, C. (2018). Ensmallen: A Flexible c++ Library for Efficient Function Optimization. *Workshop on Systems for ML and Open Source Software at NeurIPS*.
2. Bradbury, J., Frostig, R., Hawkins, P., et al. (2018). JAX: Composable Transformations of Python+NumPy Programs.
3. Sanderson, C., & Curtin, R. R. (2016). The Design and Implementation of the Armadillo c++ Linear Algebra Library. *Mathematical Software-ICMS 2016*.