Journal of Machine Learning Research 26 (2025) 1-6 Submitted 1/24; Revised 4/25; Published 8/25

skglm: Improving scikit-learn for Regularized Generalized Linear Models

Badr Moufad[1] BADR.MOUFAD@INRIA.FR
Pierre-Antoine Bannier PIERREANTOINE.BANNIER@GMAIL.COM
Quentin Bertrand[3] QUENTIN.BERTRAND@MILA.QUEBEC
Quentin Klopfenstein[2] QUENTIN.KLOPFENSTEIN@UNI.LU
Mathurin Massias[1] MATHURIN.MASSIAS@INRIA.FR

[1] *Univ Lyon, Inria, CNRS, ENS de Lyon, UCB Lyon 1, LIP UMR 5668, F-69342, Lyon, France*
[2] *University of Luxembourg, LCSB, Esch-sur-Alzette, Luxembourg*
[3] *Mila & UdeM, Canada*

Editor: Joaquin Vanschoren

Abstract

We introduce skglm, an open-source Python package for regularized Generalized Linear Models. Thanks to its composable nature, it supports combining datafits, penalties, and solvers to fit a wide range of models, many of them not included in scikit-learn (e.g. Group Lasso and variants). It uses state-of-the-art algorithms to solve problems involving high-dimensional datasets, providing large speed-ups compared to existing implementations. It is fully compliant with the scikit-learn API and acts as a drop-in replacement for its estimators. Finally, it abides by the standards of open source development and is integrated in the scikit-learn-contrib GitHub organization.

Keywords: generalized linear models, regularization, high-dimensional data, scikit-learn

1. Introduction

Generalized Linear Models (GLMs) are simple yet powerful models. They are highly interpretable as they assume the output is a function of a linear combination of features. They are often coupled with a regularization term endowing their coefficients with additional properties such as sparsity or group structure. From an optimization perspective, learning these coefficients requires solving an optimization problem with a composite objective, the sum of a datafit and a penalty: the datafit embodies the model specifications whereas the penalty enforces a given prior on the solution.

There exists a wealth of datafits and penalties covering a broad range of applications such as inverse problems in neuroscience (Strohmeier et al., 2016) or survival analysis (Efron, 1977) and having tailored properties, for instance robustness to outliers (Barron, 2019) or bias reduction (Fan and Li, 2001). Many existing packages offer implementations of regularized GLMs. For the Python machine learning community, scikit-learn (Pedregosa et al., 2011) is the de facto standard as it exposes an efficient implementation of these models through a user-friendly API easy to use and adopt even by non-experts.

```python
# Using skglm estimator
from skglm.estimators import MCPRegression

estimator = MCPRegression()
estimator.fit(X, y)
```

```python
# using composition
from skglm.datafits import Quadratic
from skglm.penalties import MCPenalty
from skglm.solvers import AndersonCD

solver = AndersonCD()
solver.solve(X, y, Quadratic(), MCPenalty(1,3))
```

Figure 1: Code snippets for solving MCP regression on design matrix `X`, and output `y`.

However, several challenges impede the prevalence of off-the-shelf regularized GLMs and prevent the community from leveraging them. First, existing packages support a limited number of GLMs, as they have a non-modular design that makes handling new datafits and penalties time-consuming[1]. Second, some reference packages may fall behind in terms of speed and efficiency, as the high implementation cost of a new method prevents them from leveraging the most recent research advances[2].

We introduce skglm, a Python package specifically designed to solve regularized GLMs. It supports many models, including those missing from standard libraries, and most importantly, can be easily extended to new penalties, datafits or solvers. It implements state-of-the-art algorithms that enable it to efficiently tackle high-dimensional datasets, making it the fastest in the current ecosystem when the number of features is large.. Finally, it complies with software development standards (Buitinck et al., 2013) hence promoting its persistence and encouraging its collaborative development.

2. Package Implementation

Design choices. Despite the diversity of regularized GLMs, from an optimization point of view, they all reduce to solving a composite problem. The main principle of skglm is to view these models as a solver that minimizes a combination of a datafit and a penalty. With that, skglm treats solvers, datafits and penalties as three separate components and combines them to solve regularized GLMs. Hence, it achieves high flexibility and extensibility by leveraging reusable independent components.

In terms of code, a solver is an object implementing a `solve` method and that has two fields to specify the datafit and the penalty required attributes. Once a datafit implements these attributes, it can be used by the solver and mixed with any other penalty that checks the required penalty attributes. So far, skglm supports 12 datafits, 16 penalties, and 8 solvers. With these components, it can solve hundreds of different problems (Table 1).

High modularity and extensiblity. As illustrated in Figure 1 on the right-hand snippet, a problem can be solved by initializing a solver then calling its `solve` method with the desired datafit and penalty. This implies that adding support for new problems is synonym to implementing a new datafit, penalty or solver and mixing it with existing components.

[1] See for example this 6 year old pull request to make scikit-learn solvers more extensible: https://github.com/scikit-learn/scikit-learn/pull/10745.

[2] An issue that highlights a lack of performance in `lifelines`, which is a reference package for survival analysis: https://github.com/CamDavidsonPilon/lifelines/issues/1531.

Single task

Datafit	Penalty
Quadratic	L1
Logistic	L1_plus_L2
QuadraticSVC	WeightedL1
Huber	MCPenalty
Poisson	WeightedMCPenalty
Gamma	SCAD
Cox	IndicatorBox
Pinball	L0_5
SqrtQuadratic	L2_3
	LogSumPenalty
	PositiveConstraint
	SLOPE

Group

Datafit	Penalty
QuadraticGroup	WeightedGroupL2
LogisticGroup	

Multitask

Datafit	Penalty
QuadraticMultiTask	L2_05
	BlockMCPenalty
	BlockSCAD

Table 1: skglm supported datafits and penalties, as of `v0.3.1` (December 2023). Any combination of a datafit and a penalty within the subtables is valid.

Fast algorithms. skglm uses state-of-the-art algorithm to solve regularized GLMs. It is built around a well-founded theory that takes advantage of the properties of problems.

In particular, for sparse GLMs, skglm leverages the small support of the solution wherein few of the coefficients are non-zero. skglm builds a working set that progressively approaches the support hence reducing considerably the optimization variables (Bertrand et al., 2022). For non quadratic datafit, taking into account the curvature through the Hessian is critical, and skglm implements a fast Prox-Newton solver.

Examples of other solvers include a wrapper for `Scipy`'s LBFGS solver, and a Primal-Dual solver for non-smooth datafits used with non-smooth penalties.

Finally, thanks to the flexibility of the design, it is possible to add new solvers to account for problems specificities while leveraging previously implemented datafits and penalties.

Figure 2 showcases the speed of skglm on three benchmarks[3]. For transparent and reproducible benchmarks, we used benchopt (Moreau et al., 2022). Primarily interested in sparse penalties, we focus on the case where the number of features is much greater than the number of samples; in the reversed configuration the results may vary.

Underlining technologies. skglm is entirely written in Python. It is a design choice in order to make code accessible and avoid the often high development time costs that result from relying on extensions, for instance written in `Cython` (Behnel et al., 2010). Although written completely in Python, skglm does not sacrifice performance and can achieve speed comparable to those achieved with extensions. skglm relies on `Numpy` (Harris et al., 2020) and `Scipy` (Virtanen et al., 2020) for dense and sparse arrays operations.

[3] Reproduce and extend the benchmarks here https://github.com/benchopt/benchmark_lasso for Lasso, https://github.com/benchopt/benchmark_cox for sparse Cox, and https://github.com/benchopt/benchmark_group_lasso for Group Lasso.

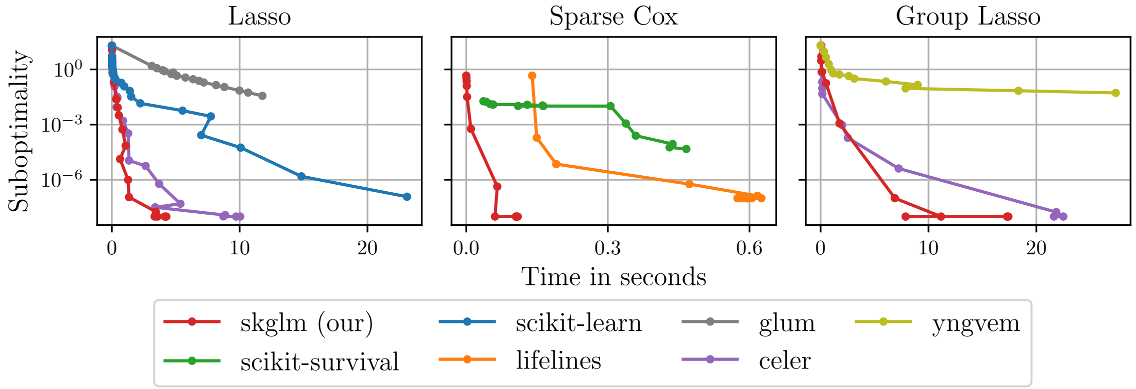

Figure 2: Timing comparison on three problems: Lasso, Sparse Cox, and Group Lasso; on the datasets: MEG, Breast-Cancer, and Drug Potency. The benchmark was performed using a laptop with specifications: CPU 12th Gen Intel® Core$^{\text{TM}}$ i7-12700H @ 2.7GHz, 20 cores, 32GB of RAM.

Algorithm specific parts that require intensive computation are isolated and JIT-compiled by `Numba` (Lam et al., 2015). Similarly, objects that perform intensive computations, namely datafits and penalties, are decorated by `Numba`'s `jitclass`. Finally, skglm estimators are fully-compliant with `scikit-learn`: they inherit from `scikit-learn`'s base classes and pass the test function `sklearn.utils.estimators_checks.check_estimator`.

3. Community

skglm is an open-source package licensed under BSD 3-Clause and hosted on GitHub[4]. It is part of the scikit-learn-contrib GitHub organization, an organization created and managed by scikit-learn core developers that gathers high quality scikit-learn compatible projects. Since the first release of skglm in May 2022, the package has gathered 100 starts, 20 forks, 10 contributors, and more than 5000 downloads per month[5].

skglm abides by the software development standards. It features meticulous testing suits comprising around 300 unit and integration tests. Besides, it has detailed and comprehensive documentation[6] with a gallery of hands-on examples and tutorials for new users. The documentation has two version: *stable* for the released code and *dev* for the one under development; both continuously built and deployed throughout skglm development cycle. Finally, to ensure the smooth onboarding of new contributors, the project has contribution guidelines as well as PR and issues templates.

4. Conclusion

skglm is an ongoing effort. It has proven its great potential in terms of speed and extensibility. With every new release, new scikit-learn compatible estimators are added, new datafits and penalties are supported, and state-of-art solvers are implemented.

[4] The repository of skglm `https://github.com/scikit-learn-contrib/skglm`.
[5] The download statistics `https://www.pepy.tech/projects/skglm`.
[6] The documentation of skglm `https://contrib.scikit-learn.org/skglm/`.

References

Jonathan T Barron. A general and adaptive robust loss function. In *Proceedings of the IEEE CVF*, 2019.

Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The best of both worlds. *CiSE*, 2010.

Quentin Bertrand, Quentin Klopfenstein, Pierre-Antoine Bannier, Gauthier Gidel, and Mathurin Massias. Beyond l1: Faster and better sparse models with skglm. *NeurIPS*, 2022.

Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, et al. Api design for machine learning software: experiences from the scikit-learn project. *European Conference on Machine Learning and Principles and Practices of Knowledge Discovery in Databases*, 2013.

Bradley Efron. The efficiency of cox's likelihood function for censored data. *JASA*, 1977.

Jianqing Fan and Runze Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *JASA*, 2001.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 2020.

Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of LLVM-HPC*, 2015.

Thomas Moreau, Mathurin Massias, Alexandre Gramfort, Pierre Ablin, Pierre-Antoine Bannier, Benjamin Charlier, Mathieu Dagréou, Tom Dupré la Tour, Ghislain Durif, Cassio F. Dantas, Quentin Klopfenstein, Johan Larsson, En Lai, Tanguy Lefort, Benoit Malézieux, Badr Moufad, Binh T. Nguyen, Alain Rakotomamonjy, Zaccharie Ramzi, Joseph Salmon, and Samuel Vaiter. Benchopt: Reproducible, efficient and collaborative optimization benchmarks. In *NeurIPS*, 2022.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *JMLR*, 2011.

Daniel Strohmeier, Yousra Bekhti, Jens Haueisen, and Alexandre Gramfort. The iterative reweighted mixed-norm estimate for spatio-temporal meg/eeg source reconstruction. *IEEE TMI*, 2016.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 2020.