skglm: Improving scikit-learn for Regularized Generalized Linear Models

Mathurin Massias [1] **Badr Moufad** [1] **Quentin Bertrand** [2]

Abstract

We introduce skglm, an open-source Python package for regularized Generalized Linear Models (GLMs). It solves many limitations of scikit-learn, that have impaired the use of GLMs by practitioners. Thanks to its composable nature, skglm 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, and unlocking new applications. 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.

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). Hence, a variety of complex GLMs are routinely used in applied fields, as an essential modelling step of the data analysis pipeline. Currently, many existing packages offer implementations of regularized GLMs. For the Python machine learning community, scikit-learn (F. 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.

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]. Large scale datasets, which have become very common in the past years, are often out of reach for analysis by the existing implementations.

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 support 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 solv-

[1]Inria, ENS de Lyon, CNRS, Université Claude Bernard Lyon 1, LIP, UMR 5668, 69342, Lyon cedex 07, France [2]Université Jean Monnet Saint-Etienne, CNRS, Institut d'Optique Graduate School, Inria, Laboratoire Hubert Curien UMR 5516, F-42023, SAINT-ETIENNE, France. Correspondence to: mathurin massias, badr moufad, quentin bertrand <firstname.lastname@inria.fr>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

[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`

ing a composite problem:

$$\min_{w} \text{datafit}(w) + \text{penalty}(w)$$

where w are the coefficients of the model. The main design principle of skglm is to view these models as a combination of three objects: *a solver that minimizes the sum 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, contrary to the current design of scikit-learn.

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 (e.g., it allows to specify if the solver needs to access the Lipschitz constant of the datafit, or the proximal operator of the penalty). Once a datafit implements these attributes, it can be used by the solver and mixed with any other penalty that implements the required penalty attributes. So far (v0.4), skglm supports 14 datafits, 16 penalties, and 8 solvers. With these components, it can solve hundreds of different problems, listed in Table 1.

High modularity and extensiblity As illustrated in Figure 1 on the bottom snippet, a problem can be solved by initializing a solver then calling its `solve` method with the desired datafit and penalty, of, more similar to the spirit of sklearn, by wrapping datafit, penalty and solver inside a `GeneralizedLienarEstimator` that behaves like a classical sklearn estimator. This implies that adding support for new problems is synonym to implementing a new datafit, penalty or solver and mixing it with existing components.

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. skglm also includes a GramCD solver, specifically optimized for problems with a Quadratic datafit where the number of samples exceeds the number of features.

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 (e.g. ℓ_1-penalized quantile regression with the pinball loss). When possible, skglm uses Anderson Acceleration (Scieur, d'Aspremont, and Bach, 2016) to accelerate the iterates of some GLMs with a vector auto-regressive behavior. Since Anderson Acceleration does not interfere with the used algorithm, it provides a cheap acceleration procedure applicable for many solvers, in particular coordinate-descent based ones.

Table 1. skglm supported datafits and penalties, as of version `v0.4` released on April 2025. The datafits and penalties can be subdivided into three categories. Any combination of a datafit and a penalty within the subtables is valid.

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
WeightedQuadratic	LogSumPenalty
QuadraticHessian	PositiveConstraint
	SLOPE

Group

Datafit	Penalty
QuadraticGroup	WeightedGroupL2
LogisticGroup	

Multitask

Datafit	Penalty
QuadraticMultiTask	L2_05
	BlockMCPenalty
	BlockSCAD

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

[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

```
# using explicit skglm estimator
from skglm.estimators import MCPRegression

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

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

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

# via sklearn-like estimator
est = GeneralizedLinearEstimator(
    datafit=Quadratic(),
    penalty=MCPenalty(alpha=1, gamma=3),
    solver=AndersonCD(),
).fit(X, y)
```

Figure 1. Code snippets for solving MCP regression on design matrix `X`, and target vector `y`. Given its popularity, MCP regression is implemented directly as an estimator (top snippet), but users can also gain more freedom by directly handling the three components (bottom snippet). For convenience, skglm implements GeneralizedLinearEstimator which allows to wrap the three components in a sklearn-like estimator. The used hyperparameters are arbitrary and given for the sake of illustration.

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. Algorithm specific parts that require intensive computation are isolated and JIT-compiled by `Numba` (Lam, Pitrou, and Seibert, 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 `check_estimator` from `sklearn.utils.estimators_checks`.

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 172 starts, 37 forks, 19 contributors, and more than 5000 downloads per month[5] (110k in total).

skglm provides three levels of abstraction to serve diverse users:

1. efficient off-the-shelf estimators that are drop-in replacements for their scikit-learn counterparts, including estimators with methods for computing solutions over regularization paths using warm-start, such as Lasso and MCPRegression.

2. building blocks for composing new scikit-learn compatible estimators via GeneralizedLinearEstimator; the latter seamlessly integrates with scikit-learn's Pipeline and GridSearchCV.

3. Flexible API for optimization and statistical researchers to implement new datafits, penalties, and solvers.

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.

Real-world impact skglm has demonstrated significant real-world impact through its integration into the energy forecasting pipelines of E-REDES[7], Portugal's largest electricity distribution company. skglm was specifically chosen for the superior speed and efficiency of its ElasticNet estimator, particularly in computationally intensive tasks like

[4]Repository of skglm `https://github.com/scikit-learn-contrib/skglm`

[5]Download statistics `https://www.pepy.tech/projects/skglm`

[6]Documentation of skglm `https://contrib.scikit-learn.org/skglm/`

[7]`https://www.e-redes.pt`

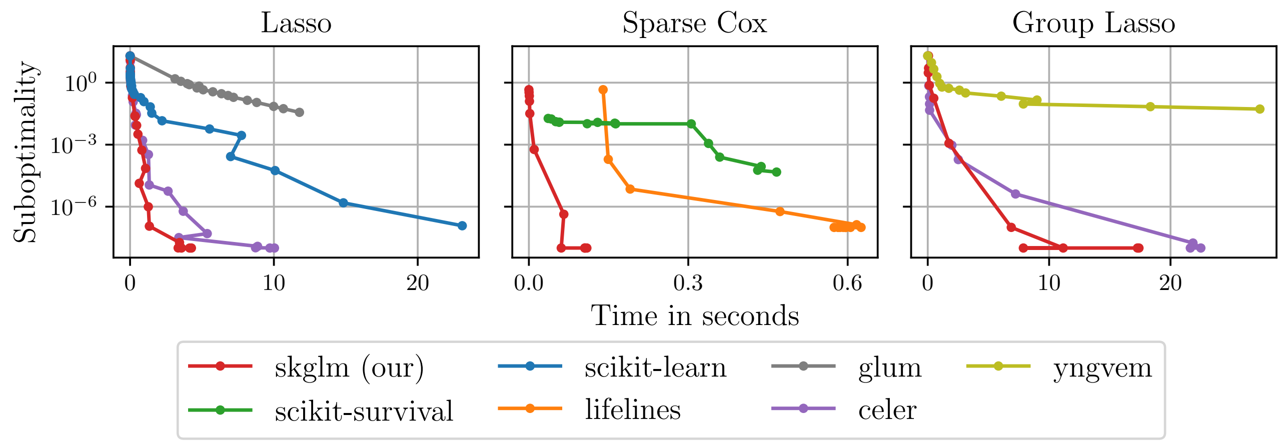

Figure 2. Timing comparison on three problems: Lasso, Group Lasso and sparse Cox; on the datasets: MEG, Breast-Cancer, and Drug Potency. The benchmark was performed using a laptop with specifications: CPU 12th Gen Intel® Core™ i7-12700H @ 2.7GHz, 20 cores, 32GB of RAM.

cross-validation. skglm's versatility, notably through its GramCD solver, enabled it to adapt effectively to the "tall" problem setup, characterized by many samples relative to features.

4. Conclusion

skglm is a modular and scikit-learn compatible package that exposes both a high-level and low-level API to solve sparse GLMs. It is flexible and support a wide range of problems and more importantly can be easily extended while being efficient. 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.

References

[1] Jonathan T Barron. "A general and adaptive robust loss function". In: *Proceedings of the IEEE CVF*. 2019.

[2] Stefan Behnel et al. "Cython: The best of both worlds". In: *CiSE* (2010).

[3] Quentin Bertrand et al. "Beyond l1: Faster and better sparse models with skglm". In: *NeurIPS* (2022).

[4] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: *European Conference on Machine Learning and Principles and Practices of Knowledge Discovery in Databases* (2013).

[5] Bradley Efron. "The efficiency of Cox's likelihood function for censored data". In: *JASA* (1977).

[6] Jianqing Fan and Runze Li. "Variable selection via nonconcave penalized likelihood and its oracle properties". In: *JASA* (2001).

[7] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* (2020).

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

[9] Thomas Moreau et al. "Benchopt: Reproducible, efficient and collaborative optimization benchmarks". In: *NeurIPS*. 2022.

[10] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *JMLR* (2011).

[11] Damien Scieur, Alexandre d'Aspremont, and Francis Bach. "Regularized nonlinear acceleration". In: *NeurIPS* (2016).

[12] Daniel Strohmeier et al. "The iterative reweighted mixed-norm estimate for spatio-temporal MEG/EEG source reconstruction". In: *IEEE TMI* (2016).

[13] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* (2020).