Fast, flexible and reproducible optimization for ML

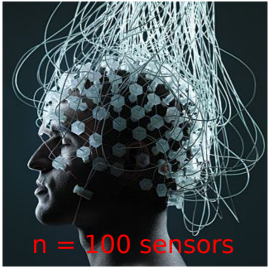
Mathurin Massias

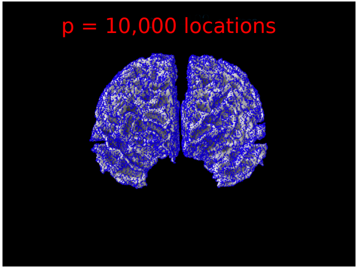
OCKHAM

CEP INRIA, 16/06/23

Early motivation: linear inverse problems

- observe magnetoelectric field outside the scalp (100 sensors)
- reconstruct cerebral activity inside the brain (10,000 locations)





- linear relationship by Maxwell equations

Still working on neuroscience! PhD of Can Pouliquen w. P. Gonçalves, T. Vayer

General setup: overparametrized linear models

Inverse problem: observe $y \approx F(x)$, infer x.
Examples: neural source identification, image denoising, etc.

Linear: $F(x) = Ax$.
ML example: linear regression

How to find x? Introduce data-fidelity divergence D and solve:

$$\boxed{\min_x D(y, F(x))}$$

Most popular divergence: $\frac{1}{2}\|y - F(x)\|^2$ (least squares)

Big issue: in general y much smaller than x (**overparametrization**):

- infinitely many solutions
- sensitivity to noise in observations y

Regularization

Solution to stabilize: introduce regularizer R, solve

$$\boxed{\min_x D(y, F(x)) + R(x)}$$

$\hookrightarrow F(x)$ still close to y but R penalizes **overcomplex** solutions:

- $\frac{1}{2}\|\cdot\|_2^2$ penalizes large norm
- $\|\nabla x\|_2^2$: penalizes high frequency signals/images

Sparse regularization

Since the 2000s, huge popularity for ℓ_1 regularization:

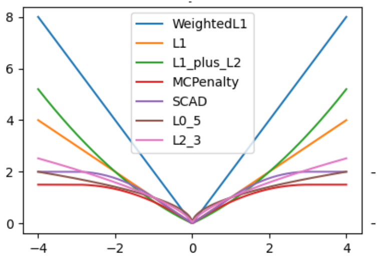
- $R(x) = \sum_j |x_j|$
- convex
- best approximation of the ℓ_0-pseudonorm
- induces sparsity in recovered x

Can be solved efficiently with 1st order (gradient-based) methods called *proximal methods*, e.g. ISTA/proximal gradient descent.

ℓ_1-specific solvers: can solve problems with millions of variables in a few seconds

Non convex penalties

Much better penalties: provide sparser solutions with same predictive power



Issue : no fast algorithm to solve them. Current limits:

- specific to quadratic data fidelity
- only for convex penalties (rely on convex duality)

Introducing skglm

"*Beyond L1: Faster and Better Sparse Models with skglm*", NeurIPS 2022

For generalized linear models, the reference Python package `scikit-learn` suffered from

- slow solvers
- complex development (relying on C/Python hybrid called Cython)
- lack of functionalities

`skglm`'s solution is two-fold

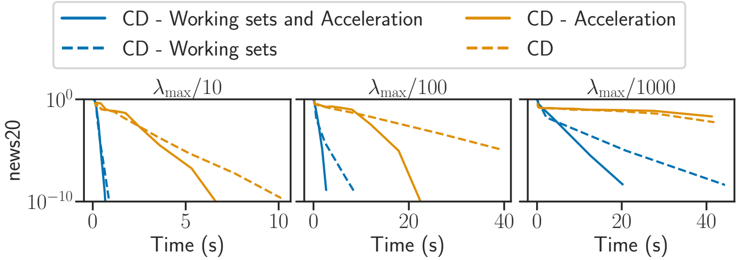
- a fast general purpose algorithm
- a modular implementation, with sklearn API

A new algorithm for sparse non convex problems

$$\boxed{\min_x D(y, Ax) + \sum_j \phi_j(x_j)}$$

2 components:

- a working set solver that identifies important variables
- a nonlinear acceleration procedure called Anderson acceleration, combined with coordinate descent



Solver details

Anderson acceleration: procedure to accelerate the convergence of fixed point iterations

$$x^{k+1} = Tx^k + b$$

Principle:

- perform K regular iterations $x^1, \ldots, x^K$
- compute K scalar coefficients $c_1, \ldots, c_k$ from it (solve $K \times K$ linear system)
- restart algorithm from $\sum_{k=1}^K c_k x^k$

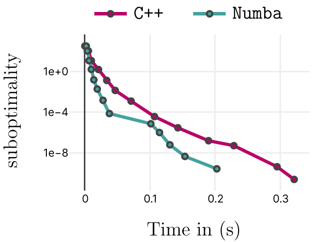
Thm: coordinate descent iteration lead to approximate fixed point iterations and thus are amenable to Anderson acceleration

Implementation choices

We introduce a flexible design, easy to handle new penalties and new datafits

Code flexibility mainly due to **numba**: Just-In-Time compilation of pure python code

- no performance loss compared to C++/Cython
- much easier code writing
- enable modular, object-oriented design



skglm flexibility

Organization around `Solver`, `Datafit` and `Penalty`.

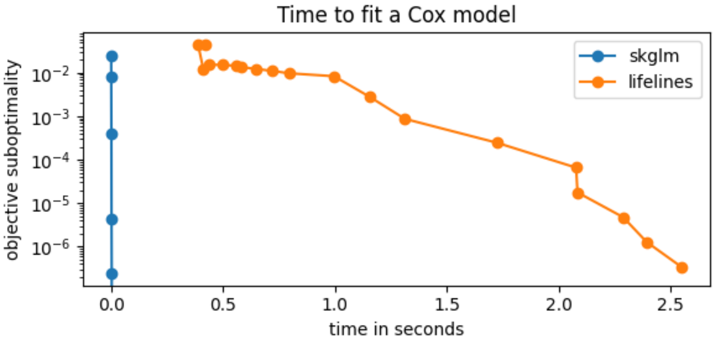
This made it possible to implement easily:

- 12 datafits (regression, classification, robustness)
- 15 penalties (non convex, group, etc)
- 4 state-of-the-art solvers

Most new classes take 50 lines of code

Survival analysis

Recent success story: x500 speedup for survival analysis model (Cox model)



$\hookrightarrow$ planned integration in the `lifelines` package (12 k downloads/day)

Benchopt: making your benchmarks easy and better

“*Benchopt: Reproducible, efficient and collaborative optimization benchmarks*”, NeurIPS 2022.

Benchmarking algorithms today is a pain

Needed: Machine Learning research relies on numerical validation.

Pain points of a benchmark:

- competitors' methods do not work out of the box.
- re-code methods and tools to integrate a new method.
- hard to extend with new settings.

all of this started from scratch by every submission!

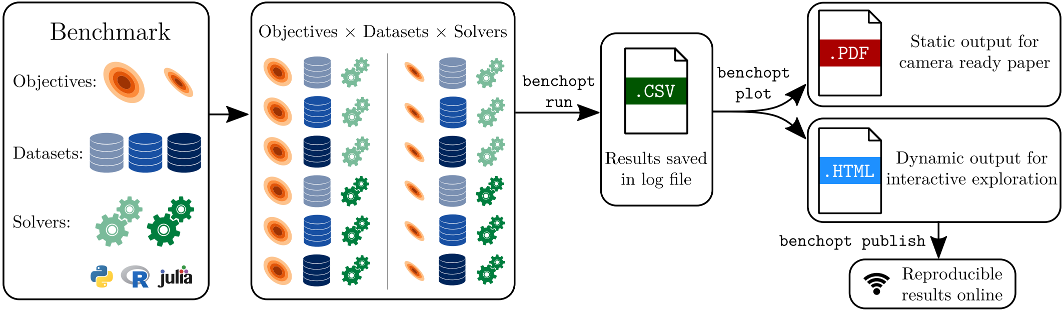
`Benchopt` makes this easier by producing open, reproducible, & extendable benchmarks

How does Benchopt do it?

Benchopt is a framework to organize and run benchmarks:

- one repository per benchmark
- one base open source Python CLI to run them

3 components: Objective, Dataset, Solver



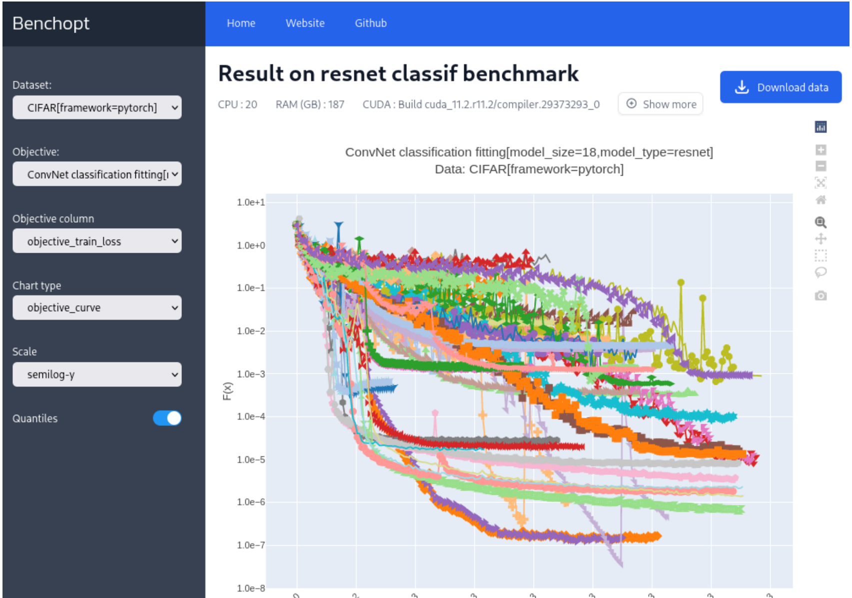
Structure of a benchmark

```
benchmark/
├── objective.py
├── datasets/
│   ├── dataset1.py
│   └── dataset2.py
└── solvers/
    ├── solver1.py
    └── solver2.py
```

Modular & extendable

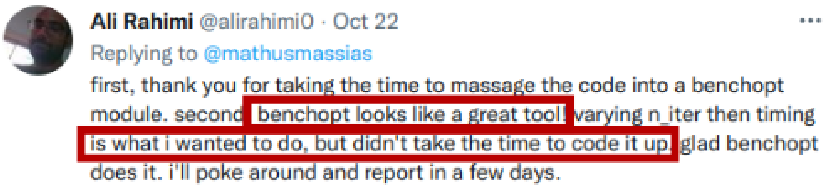
New solver? add a file
New dataset? add a file
New metric? modify objective

Interactive results exploration



Benchopt **makes your life easy**

- build on previous benchmarks
- use solvers in Python, R, Julia, binaries...
- monitor any metric you want altogether (test/train loss, ...)
- add parameters to solvers
- share and publish HTML results
- run all benchmarks in parallel
- cache results
- and much more!



Existing benchmarks

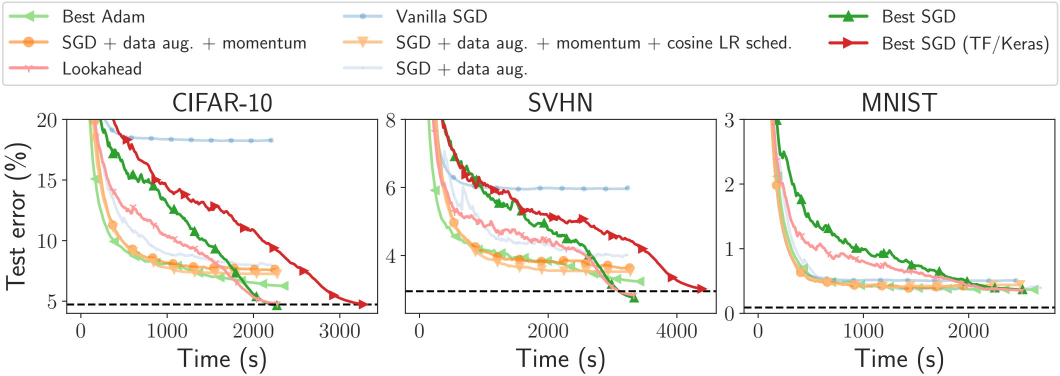
Examples of existing benchmarks:

- Resnet18
- Lasso
- ICA
- Logistic regression
- Total Variation
- Ordinary Least Squares
- Non convex sparse regression
- linear SVM

Start yours with `https://github.com/benchopt/template_benchmark`!

Deep learning benchmark

- image classification with resnet18
- various optimization strategies
- compare `pytorch` and `tensorflow`
- publish reproducible SOTA for baselines



`https://github.com/benchopt/benchmark_resnet_classif/`