Picasso: A Sparse Learning Library for High Dimensional Data Analysis in R and Python

J. Ge, X. Li, H. Jiang, H. Liu, T. Zhang, M. Wang and T. Zhao*

Abstract

We describe a new library named picasso, which implements a unified framework of pathwise coordinate optimization for a variety of sparse learning problems (e.g., sparse linear regression, sparse logistic regression, sparse Poisson regression and scaled sparse linear regression), combined with efficient active set selection strategies. Besides, the library allows users to choose different sparsity-inducing regularizers, including the convex ℓ_1, nonvoncex MCP and SCAD regularizers. The library is coded in C, has user-friendly R and Python wrappers, and can scale up to large problems efficiently with the memory optimized using sparse matrix output.

1 Introduction

The pathwise coordinate optimization is undoubtedly one the of the most popular solvers for a large variety of sparse learning problems. By leveraging the solution sparsity through a simple but elegant algorithmic structure, it significantly boosts the computational performance in practice (Friedman et al., 2007). Some recent progresses in (Zhao et al., 2017; Li et al., 2017) establish theoretical guarantees to further justify its computational and statistical superiority for both convex and nonvoncex sparse learning, which makes it even more attractive to practitioners.

We recently developed a new library named picasso, which implements a unified toolkit of pathwise coordinate optimization for solving a large class of convex and nonconvex regularized sparse learning problems. Efficient active set selection strategies are provided to guarantee superior statistical and computational preference. Specifically, we implement sparse linear regression, sparse logistic regression, sparse Poisson regression, scaled sparse linear regression, and undirected graphical model estimation (Tibshirani, 1996; Belloni et al., 2011; Sun and Zhang, 2012;

*Xingguo Li and Jason Ge contributed equally; Xingguo Li is affiliated with Department of Electrical and Computer Engineering at University of Minnesota; Tong Zhang is affiliated with Tencent AI Lab; Jason Ge, Mengdi Wang, and Han Liu are affiliated with Department of Operations Research and Financial Engineering at Princeton University; Haoming Jiang and Tuo Zhao are affiliated with Schools of Industrial & Systems Engineering and Computational Science & Engineering at Georgia Institute of Technology, Atlanta, GA 30332; Emails: lixx1661@umn.edu, jiange@princeton.edu, tourzhao@gatech.edu; Tuo Zhao is the corresponding author.

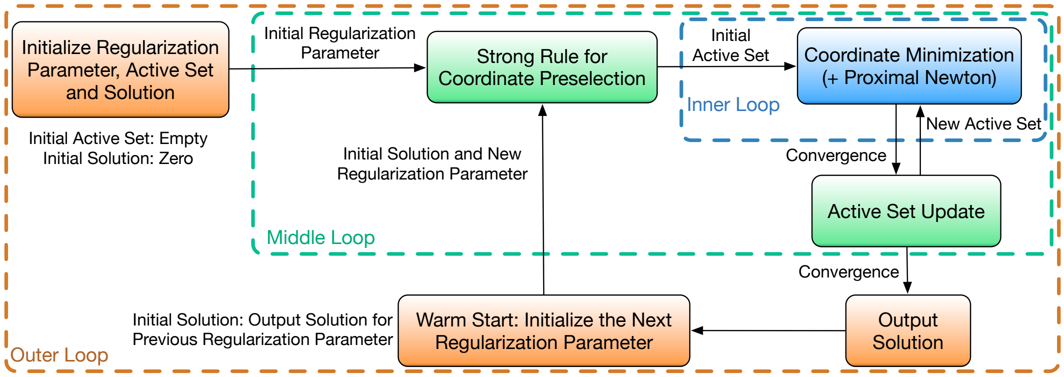


Figure 2.1: The pathwise coordinate optimization framework with 3 nested loops : (1) Warm start initialization; (2) Active set selection, and strong rule for coordinate preselection; (3) Active coordinate minimization.

Ravikumar et al., 2010; Liu and Wang, 2012; Sun and Zhang, 2013). The options of regularizers include the ℓ_1, MCP, and SCAD regularizers (Fan and Li, 2001; Zhang, 2010). Unlike existing libraries implementing heuristic optimization algorithms such as `ncvreg` or `glmnet` (Breheny, 2013; Friedman et al., 2010), our implemented algorithm `picasso` have strong theoretical guarantees that it attains a global linear convergence to a unique sparse local optimum with optimal statistical properties (e.g. minimax optimality and oracle properties). See more technical details in Zhao et al. (2017); Li et al. (2017).

2 Algorithm Design and Implementation

The algorithm implemented in `picasso` is mostly based on the generic pathwise coordinate optimization framework proposed by Zhao et al. (2017); Li et al. (2017), which integrates the warm start initialization, active set selection strategy, and strong rule for coordinate preselection into the classical coordinate optimization. The algorithm contains three structurally nested loops as shown in Figure 2.1:

(1) Outer loop: The warm start initialization, also referred to as the pathwise optimization scheme, is applied to minimize the objective function in a multistage manner using a sequence of decreasing regularization parameters, which yields a sequence of solutions from sparse to dense. At each stage, the algorithm uses the solution from the previous stage as initialization.

(2) Middle loop: The algorithm first divides all coordinates into active ones (active set) and inactive ones (inactive set) by a so-called strong rule based on coordinate gradient thresholding (Tibshirani et al., 2012). Then the algorithm calls an inner loop to optimize the objective, and update the active set based on efficient active set selection strategies. Such a routine is

repeated until the active set no longer changes

(3) Inner loop: The algorithm conducts coordinate optimization (for sparse linear regression) or proximal Newton optimization combined with coordinate optimization (for sparse logistic regression, Possion regression, scaled sparse linear regression, sparse undirected graph estimation) only over active coordinates until convergence, with all inactive coordinates staying zero values. The active coordinates are updated efficiently using an efficient "naive update" rule that only operates on the non-zero coefficients. Better efficiency is achieved by the "covariance update" rule. See more details in (Friedman et al., 2010). The inner loop terminates when the successive descent is within a predefined numerical precision.

The warm start initialization, active set selection strategies, and strong rule for coordinate preselection significantly boost the computational performance, making pathwise coordinate optimization one of the most important computational frameworks for sparse learning. The library is implemented in `C` with the memory optimized using sparse matrix output, and called from `R` and `Python` by user-friendly interfaces. The numerical evaluations show that `picasso` is efficient and can scale to large problems.

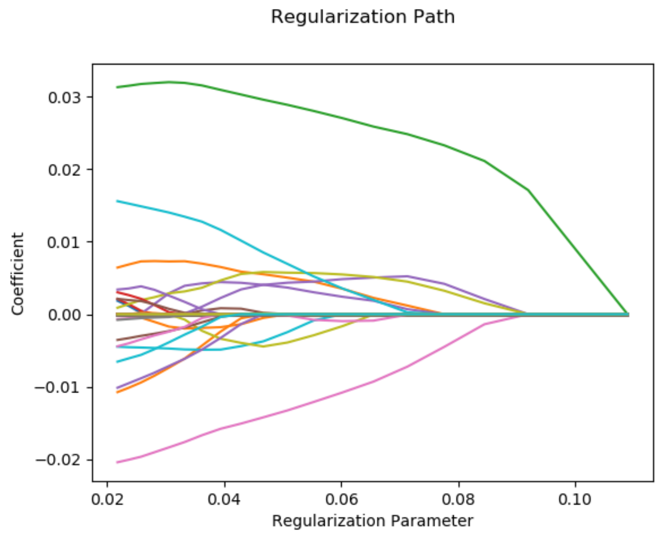
3 Examples

We illustrate the user interface by analyzing the eye disease data set in `picasso`.

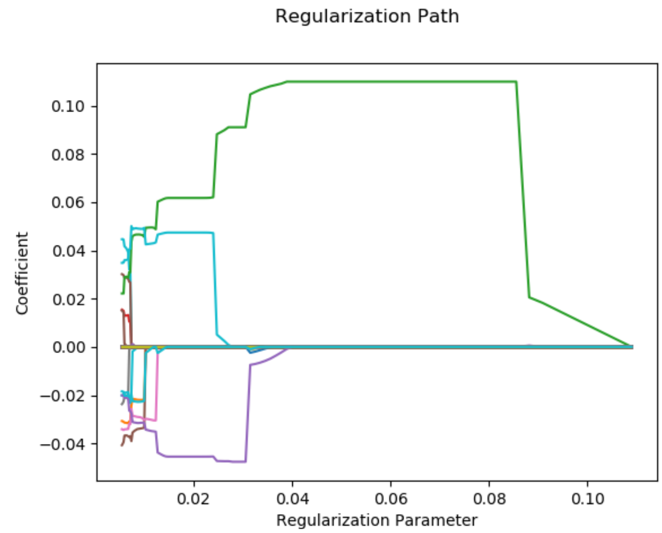
3.1 R User Interface

```
> # Load the data set
> library(picasso); data(eyedata)
> # Lasso
> out1 = picasso(x,y,method="l1",type.gaussian="naive",nlambda=20,
+                lambda.min.ratio=0.2)
> # MCP regularization
> out2 = picasso(x,y,method="mcp", gamma = 1.25, prec=1e-4)
> # Plot solution paths
> plot(out1); plot(out2)
```

The program automatically generates a sequence of regularization parameters and estimate the corresponding solution paths based on the ℓ_1 and MCP regularizers respectively. For the ℓ_1 regularizer, the number of regularization parameters as 20, and the minimum regularization parameter as `0.2*lambda.max`. Here `lambda.max` is the smallest regularization parameter yielding an all zero solution (automatically calculated by the library). For the MCP regularizer, we set the concavity parameter as $\gamma = 1.25$, and the pre-defined accuracy as 10^{-4}. Here `nlambda` and `lambda.min.ratio` are omitted, and therefore set by the default values (`nlambda=100` and `lambda.min.ratio=0.01`). We further plot two solution paths in Figure 3.1.



(a) ℓ_1



(b) MCP

Figure 3.1: The solution paths of the ℓ_1-regularized and MCP-regularized sparse linear regression.

3.2 Python User Interface

The library is named ”Pycasso” for Python to avoid the conflict with https://pypi.python.org/pypi/Picasso/0.0.2..

```
> # Load the library and the data set
> from pycasso import core
> import numpy as np
> data = np.load("eyedata.npy").item()
> x = data["data"]
> y = data["label"]
> # Lasso
> s1 = core.Solver(x,y,penalty="l1",type_gaussian="naive",nlambda=20,\
>       lambda_min_ratio=0.2)
> # MCP regularization
> s2 = core.Solver(x,y,penalty="mcp", gamma = 1.25, prec=1e-4)
> # Plot solution paths
> s1.plot(); s2.plot()
```

The results are stored in out1 and out2. We further plot two solution paths and see the same result in Figure 3.1.

4 Numerical Simulation

To demonstrate the superior efficiency of our library, we compare `picasso` with a popular `R` library `ncvreg` (version 3.9.1) for nonconvex regularized sparse regression, the most popular `R` library `glmnet` (version 2.0-13) for convex regularized sparse regression, and two `R` libraries `scalreg` (version 1.0) and `flare` (version 1.5.0) for scaled sparse linear regression. All experiments are evaluated on an Intel Xeon CPU E5-2667 v4 3.20GHz and under `R` version 3.4.3. Timings of the CPU execution are recored in seconds and averaged over 10 replications on a sequence of 100 regularization parameters. All libraries are compared based on the same regularization path and the convergence threshold are adjusted so that similar objective gaps are achieved (if possible).

For (scaled) sparse linear regression, we choose the (n, d) pairs as $(500, 5000)$ and $(1000, 10000)$ respectively, where n is the number of observation and d is the dimension. For the design matrix $X \in \mathbb{R}^{n\times d}$, we generate each row independently from a d-dimensional normal distribution $\mathcal{N}(\mathbf{0}, \Sigma)$. For well-conditioned design, we choose $\Sigma_{ij} = 0.25$ for $i \neq j$ and $\Sigma_{ii} = 1$. For ill-conditioned design, we choose $\Sigma_{ij} = 0.75$ for $i \neq j$ and $\Sigma_{ii} = 1$. Then we generate $y = X\theta + \varepsilon$, where all entries θ are 0 except the first 20 entries independently drawn from Uniform$(0, 1)$, and ε is drawn from a n dimensional normal distribution $\mathcal{N}(\mathbf{0}, I)$. For sparse logistic regression, we use the same data generation processes for X and θ. The response y_i is independent drawn from Bernoulli$\left(1/(1 + \exp(-X_{i*}^\top\theta))\right)$, where X_{i*} denotes the i-th row of the design matrix.

We summarize the numerical results in Tables 5.1 – 5.3:

(1) Sparse linear regression: `picasso` achieves similar timing and optimization performance to `glmnet` and `ncvreg`.

(2) Sparse logistic regression: When using the ℓ_1 regularizer, `picasso`, `glmnet` and `ncvreg` achieves similar optimization performance. When using the nonconvex regularizers, `picasso` achieves significantly better optimization performance than `ncvreg`, especially in ill-conditioned cases.

(3) Scaled sparse linear regression: `picasso` significantly outperforms `scalreg` and `flare` in timing performance. In Table 5.3, `picasso` is $20 - 100$ times faster and achieves smaller objective function values.

Moreover, we remark that `picasso` performs stably for various settings and tuning parameters. However, `ncvreg` is vulnerable to ill-conditioned problems, especially when the tuning parameters are relatively small (corresponding to denser solutions). We often observe that it converges either very slow or fails to converge (not shown in Tables 5.1 – 5.3).

5 Conclusion

The `picasso` library demonstrates significantly improved computational and statistical performance over existing libraries for nonconvex regularized sparse learning such as `ncvreg`. Besides, `picasso` also shows improvement over the popular libraries for convex regularized sparse learning such as `glmnet`. Overall, the `picasso` library has the potential to serve as a powerful toolbox for high dimensional sparse learning. We will continue to maintain and support this library.

Table 5.1: Average timing performance (in seconds) and objective values with standard errors in the parentheses for sparse linear regression.

Well-conditioned Design					
Method	Library	$n = 500, d = 5000$		$n = 1000, d = 10000$	
		Time	Obj. Value	Time	Obj. Value
ℓ_1 norm	`picasso`	0.176(0.068)	21.072	0.466(0.003)	24.030
	`glmnet`	0.190(0.082)	21.112	0.438(0.003)	24.013
	`ncvreg`	0.220(0.079)	21.113	0.548(0.006)	24.012
MCP	`picasso`	0.290(0.088)	17.676	0.470(0.020)	22.067
	`ncvreg`	0.342(0.015)	17.620	0.594(0.014)	22.066
SCAD	`picasso`	0.252(0.045)	20.641	0.650(0.008)	23.773
	`ncvreg`	0.302(0.071)	20.610	0.746(0.014)	23.809
Ill-conditioned Design					
ℓ_1 norm	`picasso`	0.128(0.021)	29.655	0.492(0.011)	30.256
	`glmnet`	0.232(0.024)	29.658	0.900(0.017)	30.259
	`ncvreg`	0.188(0.031)	29.654	0.692(0.011)	30.257
MCP	`picasso`	0.064(0.008)	29.915	0.262(0.003)	30.348
	`ncvreg`	0.080(0.007)	30.609	0.272(0.040)	30.349
SCAD	`picasso`	0.124(0.015)	29.655	0.508(0.006)	30.256
	`ncvreg`	0.188(0.009)	29.654	0.680(0.034)	30.257

Table 5.2: Average timing performance (in seconds) and objective values with standard errors in the parentheses for sparse logistic regression.

Well-conditioned Design					
Method	Library	$n = 500, d = 5000$		$n = 1000, d = 10000$	
		Time	Obj. Value	Time	Obj. Value
ℓ_1 norm	picasso	0.138(0.093)	0.346	0.324(0.006)	0.363
	glmnet	0.186(0.088)	0.346	0.600(0.011)	0.363
	ncvreg	0.168(0.051)	0.346	0.276(0.006)	0.363
MCP	picasso	0.098(0.093)	0.242	0.102(0.003)	0.215
	ncvreg	0.112(0.085)	0.292	0.126(0.003)	0.244
SCAD	picasso	0.100(0.079)	0.248	0.098(0.003)	0.221
	ncvreg	0.114(0.076)	0.314	0.162(0.003)	0.271
Ill-conditioned Design					
ℓ_1 norm	picasso	0.086(0.003)	0.325	0.438(0.037)	0.335
	glmnet	0.208(0.013)	0.325	1.236(0.153)	0.335
	ncvreg	0.156(0.031)	0.325	1.104(0.096)	0.335
MCP	picasso	0.026(0.003)	0.175	0.100(0.021)	0.170
	ncvreg	0.052(0.012)	0.222	0.264(0.006)	0.228
SCAD	picasso	0.028(0.009)	0.183	0.106(0.003)	0.181
	ncvreg	0.104(0.007)	0.253	1.028(0.019)	0.275

Table 5.3: Average timing performance (in seconds) and objective values with standard errors in the parentheses for scaled sparse linear regression.

Well-conditioned Design				
Library	$n = 500, d = 5000$		$n = 1000, d = 10000$	
	Time	Obj. Value	Time	Obj. Value
picasso	0.368(0.045)	2.677	0.360(0.000)	4.454
flare	1.512(0.040)	3.336	5.324(0.062)	5.188
scalreg	1.680(0.034)	2.867	40.202(0.608)	4.492
Ill-conditioned Design				
picasso	0.040(0.002)	5.388	0.146(0.003)	5.495
flare	13.092(0.113)	5.979	297.356(2.772)	5.959
scalreg	3.354(0.427)	5.395	49.120(10.986)	5.507

References

Belloni, A., Chernozhukov, V. and Wang, L. (2011). Square-root lasso: pivotal recovery of sparse signals via conic programming. *Biometrika* **98** 791–806.

Breheny, P. (2013). ncvreg: Regularization paths for scad-and mcp-penalized regression models. *R package version* 2–6.

Fan, J. and Li, R. (2001). Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association* **96** 1348–1360.

Friedman, J., Hastie, T., Höfling, H. and Tibshirani, R. (2007). Pathwise coordinate optimization. *The Annals of Applied Statistics* **1** 302–332.

Friedman, J., Hastie, T. and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software* **33** 1–13.

Li, X., Ge, J., Jiang, H., Wang, M., Hong, M. and Zhao, T. (2017). Boosting pathwise coordinate optimization in high dimensions: Sequential screening and proximal sub-sampled newton algorithm. Tech. rep., Georgia Tech.

Liu, H. and Wang, L. (2012). Tiger: A tuning-insensitive approach for optimally estimating gaussian graphical models. Tech. rep., Massachusett Institute of Technology.

Ravikumar, P., Wainwright, M. J., Lafferty, J. D. et al. (2010). High-dimensional ising model selection using 1-regularized logistic regression. *The Annals of Statistics* **38** 1287–1319.

Sun, T. and Zhang, C. (2012). Scaled sparse linear regression. *Biometrika* To appear.

Sun, T. and Zhang, C.-H. (2013). Sparse matrix inversion with scaled lasso. *Journal of Machine Learning Research* **14** 3385–3418.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B* **58** 267–288.

Tibshirani, R., Bien, J., Friedman, J., Hastie, T., Simon, N., Taylor, J. and Tibshirani, R. (2012). Strong rules for discarding predictors in lasso-type problems. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **74** 245–266.

Zhang, C. (2010). Nearly unbiased variable selection under minimax concave penalty. *The Annals of Statistics* **38** 894–942.

Zhao, T., Liu, H. and Zhang, T. (2017). Pathwise coordinate optimization for nonconvex sparse learning: Algorithm and theory. *Annals of Statistics* .