The Journal of Open Source Software

# c-lasso - a package for constrained sparse and robust regression and classification in Python

**Léo Simpson**[1], **Patrick L. Combettes**[2], **and Christian L. Müller**[3]

**1** TUM **2** NC State **3** LMU

**DOI:** 10.21105/joss.0XXXX

**Software**

- Review
- Repository
- Archive

**Editor:** Editor Name

**Submitted:** 01 January XXXX
**Published:** 01 January XXXX



## Summary

c-lasso is a Python package that enables sparse and robust linear regression and classification with linear equality constraints.

The forward model is assumed to be:

$$y = X\beta + \sigma\epsilon \qquad \text{s.t.} \qquad C\beta = 0$$

Here, $X$ and $y$ are given outcome and predictor data. The vector $y$ can be continuous (for regression) or binary (for classification). $C$ is a general constraint matrix. The vector $\beta$ comprises the unknown coefficients $\epsilon$ an unknown noise and $\sigma$ an unknown scale.

Depending on the prior we assume on those unknown variables, this forward model can lead to different types of estimation. Our package can solve six of those : four regression-type and two classification-type formulations. Those are all variants of the standard formulation "*R1*" :

$$\arg\min_{\beta\in\mathbb{R}^d} \|X\beta - y\|^2 + \lambda \|\beta\|_1 \qquad s.t. \qquad C\beta = 0$$

### Formulations

- ***R1* Standard constrained Lasso regression**: This is the standard Lasso problem with linear equality constraints on the $\beta$ vector. The objective function combines Least-Squares for model fitting with $L_1$ penalty for sparsity.
- ***R2* Contrained sparse Huber regression**: This regression problem uses the Huber loss as objective function for robust model fitting with $L_1$ and linear equality constraints on the $\beta$ vector. The parameter $\rho$ is set to $1.345$ by default (Aigner, Amemiya, & Poirier, 1976)
- ***R3* Contrained scaled Lasso regression**: This formulation is similar to *R1* but allows for joint estimation of the (constrained) $\beta$ vector and the standard deviation $\sigma$ in a concomitant fashion (Combettes & Müller, 2020; Combettes & Müller, 2020). This is the default problem formulation in c-lasso.
- ***R4* Contrained sparse Huber regression with concomitant scale estimation**: This formulation combines *R2* and *R3* to allow robust joint estimation of the (constrained) $\beta$ vector and the scale $\sigma$ in a concomitant fashion (Combettes & Müller, 2020; Combettes & Müller, 2020).

- ***C1* Contrained sparse classification with Square Hinge loss**: This formulation is similar to *R1* but adapted for classification tasks using the Square Hinge loss with (constrained) sparse $\beta$ vector estimation (Lee & Lin, 2013).
- ***C2* Contrained sparse classification with Huberized Square Hinge loss**: This formulation is similar to *C1* but uses the Huberized Square Hinge loss for robust classification with (constrained) sparse $\beta$ vector estimation (Rosset & Zhu, 2007).

#### Model selections

Different models are implemented together with the optimization schemes, to overcome the difficulty of choosing the penalization free parameter $\lambda$.

- *Fixed Lambda* : This approach is simply letting the user choose the parameter $\lambda$, or to choose $l \in [0, 1]$ such that $\lambda = l \times \lambda_{\max}$. The default value is a scale-dependent tuning parameter that has been proposed in [Combettes:2020.2] and derived in (Shi, Zhang, & Li, 2016).
- *Path Computation* :The package also leaves the possibility to us to compute the solution for a range of $\lambda$ parameters in an interval $[\lambda_{\min}, \lambda_{\max}]$. It can be done using *Path-Alg* or warm-start with any other optimization scheme.
- *Cross Validation* : Then one can use a model selection, to choose the appropriate penalisation. This can be done by using k-fold cross validation to find the best $\lambda \in [\lambda_{\min}, \lambda_{\max}]$ with or without "one-standard-error rule" (Hastie, Tibshirani, & Friedman, 2009).
- *Stability Selection* : Another variable selection model than can be used is stability selection [LIN, SHI, FENG, & LI (2014); Meinshausen & Bühlmann (2010); Combettes:2020.2].

## Statement of need

The package handles several estimators for inferring location and scale, including the constrained Lasso, the constrained scaled Lasso, and sparse Huber M-estimation with linear equality constraints Several algorithmic strategies, including path and proximal splitting algorithms, are implemented to solve the underlying convex optimization problems. We also include two model selection strategies for determining the sparsity of the model parameters: k-fold cross-validation and stability selection. This package is intended to fill the gap between popular python tools such as `scikit-learn` which cannot solve sparse constrained problems and general-purpose optimization solvers such as `cvx` that do not scale well for the considered problems or are inaccurate. We show several use cases of the package, including an application of sparse log-contrast regression tasks for compositional microbiome data. We also highlight the seamless integration of the solver into `R` via the `reticulate` package.

## Basic workflow

Here is a basic example that shows how to run c-lasso on synthetic data.

c-lasso is available on pip, one can install it using `pip install c_lasso`. Then on python, to import the package, one should use `import classo`

Let us now begin the tutorial. Firstly, let us generate a dataset using the routine `random_data` included in the c-lasso package, that allows you to generate instances using normally distributed data.

```
>>> n,d,d_nonzero,k,sigma =100,100,5,1,0.5
>>> (X,C,y),sol = random_data(n,d,d_nonzero,k,sigma,zerosum=True, seed = 123 )
>>> list(numpy.nonzero(sol))
[43, 47, 74, 79, 84]
```

This code snippet generates randomly the vectors $\beta \in R^d$ , $X \in R^{n \times d}$ , $C \in R^{k \times d}$ (here it is the all-one vector instead because of the input `zerosum`), and $y \in R^n$ normally distributed with respect to the model $C\beta = 0$, $y - X\beta \sim N(0, I_n\sigma^2)$ and $\beta$ has only d_nonzero non-null componant (which are plot above).

Then, let us define a `classo_problem` instance with the generated dataset in order to formulate the optimization problem we want to solve.

```
# to define a c-lasso problem instance with default setting :
>>> problem  = classo_problem(X,y,C)
# to change the formulation of the problem :
>>> problem.formulation.huber  = True
>>> problem.formulation.concomitant = False
>>> problem.formulation.rho = 1.5
# to add the computation for a fixed lambda :
>>> problem.model_selection.LAMfixed = True
# to set lambda to 0.1*lambdamax :
>>> problem.model_selection.LAMfixedparameters.rescaled_lam = True
>>> problem.model_selection.LAMfixedparameters.lam = 0.1
# to add the computation of the lambda-path :
>>> problem.model_selection.PATH = True
# to solve our optimization problem :
>>> problem.solve()
```

Here, we have modified the formulation of the problem in order to use *R2*, with $\rho = 1.5$. We have chosen the following model selections : *Fixed Lambda* with $\lambda = 0.1\lambda_{\max}$ ; *Path computation* and *Stability Selection* which is computed by default. Then, those problems are solved using the recommanded optimization scheme on each model according to the formulation and the size of the parameter $\lambda$

Finally, one can visualize the solutions and see the running time, and the name of the selected variables by calling the instance `problem.solution`. Note that by calling directly the instance `problem` one could also visualize the main parameters of the optimization problems one is solving. In our case, the running time is in the order of 0.1sec for the fixed lambda and path computation, but vary from 2sec to 4sec for the stability selection computation.

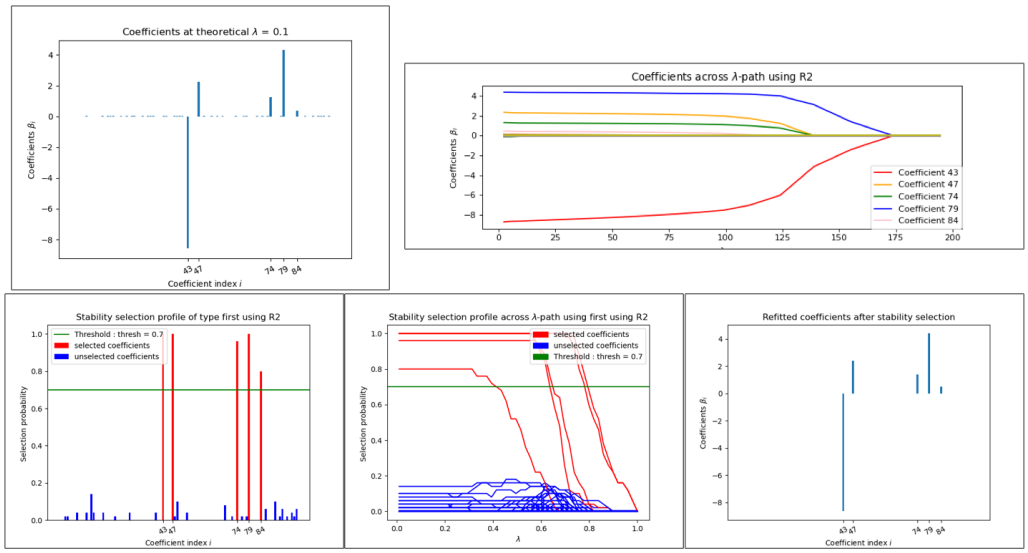


**Figure 1:** Graphics plotted after calling `problem.solution`

## References


Aigner, D. J., Amemiya, T., & Poirier, D. J. (1976). On the estimation of production frontiers: Maximum likelihood estimation of the parameters of a discontinuous density function. *International Economic Review*, *17*(2), 377–396. Retrieved from http://www.jstor.org/stable/2525708

Combettes, P. L., & Müller, C. L. (2020). Perspective maximum likelihood-type estimation via proximal decomposition. *Electron. J. Statist.*, *14*(1), 207–238. doi:10.1214/19-EJS1662

Combettes, P. L., & Müller, C. L. (2020). Regression models for compositional data: General log-contrast formulations, proximal optimization, and microbiome data applications. *Statistics in Biosciences*. doi:10.1007/s12561-020-09283-2

Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). *The elements of statistical learning: Data mining, inference, and prediction*. Springer series in statistics. Springer. ISBN: 9780387848846

Lee, C.-P., & Lin, C.-J. (2013). A study on l2-loss (squared hinge-loss) multiclass svm. *Neural computation*, *25*. doi:10.1162/NECO_a_00434

LIN, W., SHI, P., FENG, R., & LI, H. (2014). Variable selection in regression with compositional covariates. *Biometrika*, *101*(4), 785–797. Retrieved from http://www.jstor.org/stable/43304688

Meinshausen, N., & Bühlmann, P. (2010). Stability selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *72*(4), 417–473. doi:10.1111/j.1467-9868.2010.00740.x

Rosset, S., & Zhu, J. (2007). Piecewise linear regularized solution paths. *Ann. Statist.*, *35*(3), 1012–1030. doi:10.1214/009053606000001370

Shi, P., Zhang, A., & Li, H. (2016). Regression analysis for microbiome compositional data. *Ann. Appl. Stat.*, *10*(2), 1019–1040. doi:10.1214/16-AOAS928