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

This article illustrates c-lasso, 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 β comprises the unknown coefficients and σ an unknown scale. The typical use case is logarithmic regression with compositional data, that impose the constraint $\sum_{i=1}^{d} \beta_i = 0$, hence $C = 1_d^T$ (AITCHISON & BACON-SHONE, 1984).

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.

Functionalities

c-lasso is available on pip. You can install the package in the shell using

```
pip install c_lasso
```

Here is the typical syntax to use c-lasso on python.

```
# to import the main class of the package
from classo import classo_problem

# to define a c-lasso problem instance with default setting
problem  = classo_problem(X,y,C)

# insert here possible modifications of the problem instance

# to solve our problem instance
problem.solve()

# finally one can visualize the instance we just solved and see solution plots as
print(problem)
print(problem.solution)
```

Formulations

Depending on the prior on the solution β, σ and on the noise ϵ, the previous forward model can lead to different types of estimation problems.

Our package can solve six of those : four regression-type and two classification-type formulations. Here is an overview of those formulation, with a code snippet to see how to use those in the python.

R1 Standard constrained Lasso regression:

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

This is the standard Lasso problem with linear equality constraints on the β vector. The objective function combines Least-Squares for model fitting with L_1 penalty for sparsity.

```
# formulation R1
problem.formulation.huber = False
problem.formulation.concomitant = False
problem.formulation.classification = False
```

R2 Contrained sparse Huber regression:

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

This regression problem uses the Huber loss as objective function for robust model fitting with L_1 and linear equality constraints on the β vector. The parameter ρ is set to 1.345 by default (Aigner, Amemiya, & Poirier, 1976)

```
# formulation R2
problem.formulation.huber = True
problem.formulation.concomitant = False
problem.formulation.classification = False
```

***R3* Contrained scaled Lasso regression:**

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

This formulation is similar to *R1* but allows for joint estimation of the (constrained) β vector and the standard deviation σ in a concomitant fashion (P. L. Combettes & Müller, 2020; P. L. Combettes & Müller, 2020). This is the default problem formulation in c-lasso.

```
# formulation R3
problem.formulation.huber = False
problem.formulation.concomitant = True
problem.formulation.classification = False
```

***R4* Contrained sparse Huber regression with concomitant scale estimation:**

$$\arg\min_{\beta\in\mathbb{R}^d} \left(h_\rho\left(\frac{X\beta - y}{\sigma}\right) + n\right)\sigma + \lambda\|\beta\|_1 \qquad s.t. \qquad C\beta = 0$$

This formulation combines *R2* and *R3* to allow robust joint estimation of the (constrained) β vector and the scale σ in a concomitant fashion (P. L. Combettes & Müller, 2020; P. L. Combettes & Müller, 2020).

```
# formulation R4
problem.formulation.huber = True
problem.formulation.concomitant = True
problem.formulation.classification = False
```

***C1* Contrained sparse classification with Square Hinge loss:**

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

where $L\left((r_1, ..., r_n)^T\right) := \sum_{i=1}^n l(r_i)$ and l is defined as :

$$l(r) = \begin{cases} (1-r)^2 & if \quad r \leq 1 \\ 0 & if \quad r \geq 1 \end{cases}$$

This formulation is similar to *R1* but adapted for classification tasks using the Square Hinge loss with (constrained) sparse β vector estimation (Lee & Lin, 2013).

```
# formulation C1
problem.formulation.huber = False
problem.formulation.concomitant = False
problem.formulation.classification = True
```

***C2* Contrained sparse classification with Huberized Square Hinge loss :**

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

where $L_\rho\left((r_1, ..., r_n)^T\right) := \sum_{i=1}^n l_\rho(r_i)$ and l_ρ is defined as :

$$l_\rho(r) = \begin{cases} (1-r)^2 & if \quad \rho \leq r \leq 1 \\ (1-\rho)(1+\rho-2r) & if \quad r \leq \rho \\ 0 & if \quad r \geq 1 \end{cases}$$

This formulation is similar to *C1* but uses the Huberized Square Hinge loss for robust classification with (constrained) sparse β vector estimation (Rosset & Zhu, 2007).

```
# formulation C2
problem.formulation.huber = True
problem.formulation.concomitant = False
problem.formulation.classification = True
```

Optimization schemes

The available problem formulations *R1*-*C2* require different algorithmic strategies for efficiently solving the underlying optimization problem. We have implemented four algorithms (with provable convergence guarantees). Those algorithms originally exist for standard regression (formulation *R1*), but we have adapted it to different formulations when it was possible, for example, using the mean-shift formulation (Mishra & Müller, 2019) for Huber regression.

Here is a summary of which algorithm can be used for each problem :

	Path-Alg	*DR*	*P-PDS*	*PF-PDS*
R1	recommanded for high λ and for path computation	recommanded for small λ	possible	recommanded for complex constraints
R2	recommanded for high λ and for path computation	recommanded for small λ	possible	recommanded for complex constraints
R3	recommanded for high λ or when path computation is require	recommanded for small λ		
R4		recommanded (only option)		
C1	recommanded (only option)			
C2	recommanded (only option)			

The python syntax to use an algorithm different than recommanded is the following :

```
problem.numerical_method = "Path-Alg"
# also works with "DR", "P-PDS" or "PF-PDS"
```

- Path algorithms (*Path-Alg*) : The algorithm uses the fact that the solution path along is piecewise-affine as shown, in (Gaines, Kim, & Zhou, 2018).
- Douglas-Rachford-type splitting method (*DR*) : This algorithm is based on Doulgas Rachford splitting in a higher-dimensional product space (P. L. Combettes & Müller, 2020; P. L. Combettes & Müller, 2020).
- Projected primal-dual splitting method (*P-PDS*) : This algorithm is derived from (Briceño-Arias & López Rivera, 2019) and belongs to the class of proximal splitting algorithms.
- Projection-free primal-dual splitting method (*PF-PDS*) : This algorithm is a special case of an algorithm proposed in (Combettes & Pesquet, 2011) (Eq.4.5) and also belongs to the class of proximal splitting algorithms.

Model selections

Different models are implemented together with the optimization schemes, to overcome the difficulty of choosing the penalization free parameter λ. When using the package, several of those model selection can be computed with the same problem-instance.

The python syntax to use some specific model selection is the following

```
# to perform Cross-Validation and Path computation :
problem.model_selection.LAMfixed = False
problem.model_selection.PATH = True
problem.model_selection.CV = True
problem.model_selection.StabSel = False
# obviously any other combination also works
```

- *Fixed Lambda* : This approach is simply letting the user choose the parameter λ, 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 [Muller:2020] 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 λ 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); Muller:2020].

Example on R

As an alternative, one can use this package in R instead of python by calling the python package with the Rlibrary `reticulate`. As an example, this code snippet used in R will perform regression with a fixed lambda set to $\lambda = 0.1\lambda_{\max}$.

One should be careful with the inputs : X should be a `matrix`, C as well, but y should be an `array` (if one set y to be matrix $1 \times n$ for example, c-lasso will not work).

```r
problem<- classo$classo_problem(X=X,C=C,y=y)
problem$model_selection$LAMfixed <- TRUE
problem$model_selection$StabSel <- FALSE
problem$model_selection$LAMfixedparameters$rescaled_lam <- TRUE
problem$model_selection$LAMfixedparameters$lam <- 0.1
problem$solve()

# extract outputs
beta <- as.matrix(map_dfc(problem$solution$LAMfixed$beta, as.numeric))
```

Example on synthetic data

The c-lasso package includes the routine `random_data` that allows you to generate problem instances using normally distributed data.

It allows perform some functionality of the package on synthetic data as an example.

```python
from classo import classo_problem, random_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 )
print("Relevant variables  : {}".format(list(numpy.nonzero(sol)) ) )

problem  = classo_problem(X,y,C)

problem.formulation.huber  = True
problem.formulation.concomitant = False
problem.formulation.rho = 1.5

problem.model_selection.LAMfixed = True
problem.model_selection.PATH = True
problem.model_selection.LAMfixedparameters.rescaled_lam = True
problem.model_selection.LAMfixedparameters.lam = 0.1

problem.solve()

print(problem.solution)
```

The formulation used is *R2*, with $\rho = 1.5$. The model selections used are *Fixed Lambda* with $\lambda = 0.1\lambda_{\max}$, *Path computation* and *Stability Selection* which is computed by default.

The output is then :

```
Relevant variables  : [43 47 74 79 84]

 LAMBDA FIXED :
   Selected variables :  43    47    74    79    84
   Running time :  0.294s

 PATH COMPUTATION :
   Running time :  0.566s

 STABILITY SELECTION :
   Selected variables :  43    47    74    79    84
   Running time :  5.3s
```

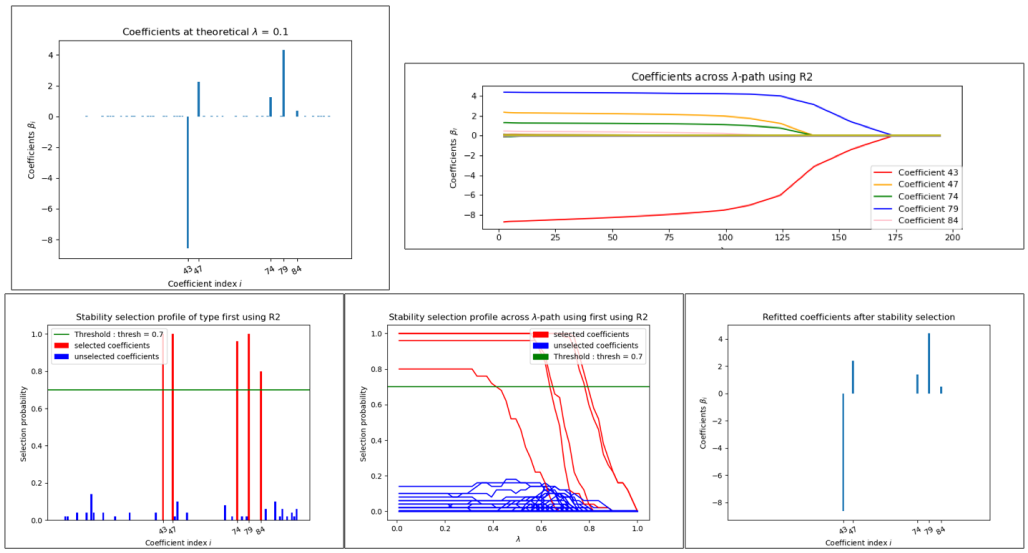

Figure 1: Graphics plotted after calling problem.solution

It is indeed the variables that have been selected with the solution threshold for a fixed lambda, and with stability selection. Let us also note that the running time is still very low in our example. Those remarks are comforting, but not surprising because in this example the noise is little and the number of variable is still small.

Acknowledgements

This work was supported by the Flatiron Institute and the Helmholtz Zentrum Munchen.

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

AITCHISON, J., & BACON-SHONE, J. (1984). Log contrast models for experiments with mixtures. *Biometrika*, *71*(2), 323–330. doi:10.1093/biomet/71.2.323

Briceño-Arias, L., & López Rivera, S. (2019). A projected primal–dual method for solving constrained monotone inclusions. *Journal of Optimization Theory and Applications*, *180*(3), 907–924. doi:10.1007/s10957-018-1430-2

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

Combettes, P., & Pesquet, J.-C. (2011). Primal-dual splitting algorithm for solving inclusions with mixtures of composite, lipschitzian, and parallel-sum type monotone operators. *Set-Valued and Variational Analysis*, *20*, 1–24. doi:10.1007/s11228-011-0191-y

Gaines, B. R., Kim, J., & Zhou, H. (2018). Algorithms for fitting the constrained lasso. *Journal of Computational and Graphical Statistics*, *27*(4), 861–871. doi:10.1080/10618600.2018.1473777

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

Mishra, A., & Müller, C. (2019). Robust regression with compositional covariates.

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