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.

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.

Current functionalities

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.

***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 l1 penalty for sparsity.

***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 l1 and linear equality constraints on the β vector. The parameter $\rho = 1.345$.

***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 (see (P. L. Combettes & Müller, 2020; P. L. Combettes & Müller, 2020) for further info). This is the default problem formulation in c-lasso.

***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 (see (P. L. Combettes & Müller, 2020; P. L. Combettes & Müller, 2020) for further info).

***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.

***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.

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) that vary in generality and are not necessarily applicable to all problems. For each problem type, c-lasso has a default algorithm setting that proved to be the fastest in our numerical experiments.

- **Path algorithms (*Path-Alg*)** : This is the default algorithm for non-concomitant problems *R1,R2,C1,C2*. The algorithm uses the fact that the solution path along is piecewise-affine (as shown, e.g., in (Gaines, Kim, & Zhou, 2018)). When Least-Squares is used as objective function, we derive a novel efficient procedure that allows us to also derive the solution for the concomitant problem *R3* along the path with little extra computational overhead.
- **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. It extends the classical Forward-Backward (FB) (aka proximal gradient descent) algorithm to handle an additional linear equality constraint via projection. In the absence of a linear constraint, the method reduces to FB. This method can solve problem *R1*. For the Huber problem *R2*, P-PDS can solve the mean-shift formulation of the problem (see (Mishra & Müller, 2019)).
- **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. The algorithm does not require projection operators which may be beneficial when C has a more complex structure. In the absence of a linear constraint, the method reduces to the Forward-Backward-Forward scheme. This method can solve problem *R1*. For the Huber problem *R2*, PF-PDS can solve the mean-shift formulation of the problem (see (Mishra & Müller, 2019)).
- **Douglas-Rachford-type splitting method (*DR*)** : This algorithm is the most general algorithm and can solve all regression problems *R1-R4*. It is based on Doulgas Rachford splitting in a higher-dimensional product space. It makes use of the proximity operators of the perspective of the LS objective (see (P. L. Combettes & Müller, 2020; P. L. Combettes & Müller, 2020)) The Huber problem with concomitant scale *R4* is reformulated as scaled Lasso problem with the mean shift (see (Mishra & Müller, 2019)) and thus solved in (n + d) dimensions.

Model selections

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

- *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 [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 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" (see (Hastie, Tibshirani, & Friedman, 2009)).
- *Stability Selection* : Another variable selection model than can be used is stability selection (see [LIN, SHI, FENG, & LI (2014); Meinshausen & Bühlmann (2010); Combettes:2020.2].

Basic workflow

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

Installation

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

```
pip install c_lasso
```

To use the c-lasso package in Python, type

```
from classo import *
```

The c-lasso package depends on several standard Python packages. The dependencies are included in the package. Those are, namely :

`numpy` ; `matplotlib` ; `scipy` ; `pandas` ; `h5py` .

Generate random data

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

```
m,d,d_nonzero,k,sigma =100,100,5,1,0.5
(X,C,y),sol = random_data(m,d,d_nonzero,k,sigma,zerosum=True, seed = 123 )
```

This code snippet generates a problem instance with sparse β in dimension d=100 (sparsity d_nonzero=5). The design matrix X comprises n=100 samples generated from an i.i.d standard normal distribution. The dimension of the constraint matrix C is d x k matrix. The noise level is σ=0.5. The input `zerosum=True` implies that C is the all-ones vector and $C\beta = 0$. The n-dimensional outcome vector y and the regression vector β is then generated to satisfy the given constraints.

Use of c-lasso on the data

Here is an example of problem instance one can create with those set of data.

```
# let's define a c-lasso problem instance with default setting
problem  = classo_problem(X,y,C)


# let's change the formulation of the problem
problem.formulation.huber  = True
problem.formulation.concomitant = False
problem.formulation.rho = 1.5


# let's add a computation of beta for a fixed lambda
problem.model_selection.LAMfixed = True
# and set it to to 0.1*lambdamax
problem.model_selection.LAMfixedparameters.rescaled_lam = True
problem.model_selection.LAMfixedparameters.lam = 0.1

# let's add a computation of the lambda-path
problem.model_selection.PATH = True


# let's solve our problem instance
problem.solve()
```

Here, we have modified the formulation of the problem in order to use *R2*, with $\rho = 1.5$.

Then we have chosen the model selections we want to compute : *Fixed Lambda* with $\lambda = 0.1\lambda_{\max}$; *Path computation* and *Stability Selection* which is computed by default.

Finally, those problems are solved using the method `solve` which computes everything.

Visualize the result

One can, before or after having solve the problem, plot the main caracteristics of the problem solved and of its solution:

```
>>> # let's visualize the main parameters set in the problem instance
>>> problem

FORMULATION: R2

MODEL SELECTION COMPUTED:
     Lambda fixed
     Path
     Stability selection

LAMBDA FIXED PARAMETERS:
     numerical_method = DR
     rescaled lam : True
     threshold = 0.177
     lam = 0.1
     theoretical_lam = 0.1994
```

```
PATH PARAMETERS:
     numerical_method : Path-Alg
     Npath = 40
     lamin = 0.013
     lamax = 1.0

STABILITY SELECTION PARAMETERS:
     numerical_method : Path-Alg
     method : first
     B = 50
     q = 10
     percent_nS = 0.5
     threshold = 0.7
     lamin = 0.01
     Nlam = 50

>>> # let's the solutions found
>>> problem.solution

 LAMBDA FIXED :
   Selected variables :  43    47    74    79    84
   Running time :  0.094s

 PATH COMPUTATION :
   Running time :  0.221s

 STABILITY SELECTION :
   Selected variables :  43    47    74    79    84
   Running time :  2.468s
```

The latter command will also plot those graphics :

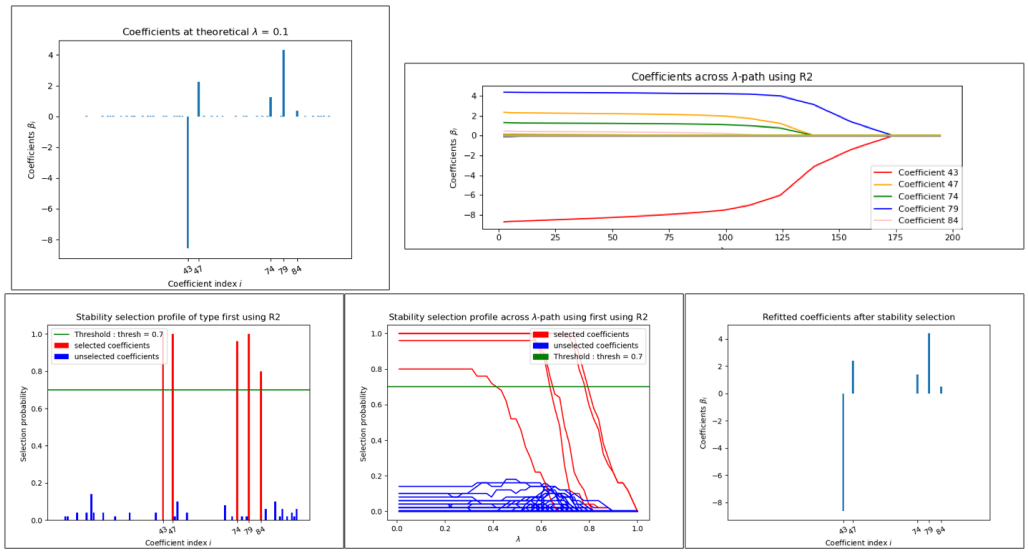

Figure 1: Graphics plotted after calling problem.solution

As this variable selection has been computed for generated data, one can plot the real relevant variables :

```
>>> import numpy
>>> print( list(numpy.nonzero(sol)) )
[43, 47, 74, 79, 84]
```

It is indeed the variables that have been selected with the solution threshold for a fixed lambda, and with stability selection.

Acknowledgements

We acknowledge ...

References

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

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.

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