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**[3,4,5]

1 Technische Universität München **2** Department of Mathematics, North Carolina State University **3** Center for Computational Mathematics, Flatiron Institute, New York **4** Institute of Computational Biology, Helmholtz Zentrum München **5** Department of Statistics, Ludwig-Maximilians-Universität München

DOI: 10.21105/joss.0XXXX

Software

- Review
- Repository
- Archive

Editor: Editor Name

Submitted: 01 January XXXX
Published: 01 January XXXX

Summary

We introduce `c-lasso`, a Python package that enables sparse and robust linear regression and classification with linear equality constraints. The underlying statistical forward model is assumed to be of the following form:

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

Here, $X \in R^{n\times d}$ is a given design matrix and the vector $y \in R^n$ is a continuous or binary response vector. The matrix C is a general constraint matrix. The vector $\beta \in R^d$ contains the unknown coefficients and σ an unknown scale. Prominent use cases are (sparse) log-contrast regression with compositional data X, requiring the constraint $1_d^T\beta = 0$ (Aitchion & Bacon-Shone, 1984) and the Generalized Lasso which is a *special case* of the described problem (see, e.g, (James, Paulson, & Rusmevichientong, 2020), Example 3). The `c-lasso` package provides several estimators for inferring unknown coefficients and scale (i.e., perspective M-estimators (Combettes & Müller, 2020a)) of the form

$$\arg\min_{\beta\in\mathbb{R}^d,\sigma\in\mathbb{R}_0} f\left(X\beta - y,\sigma\right) + \lambda\left\|\beta\right\|_1 \qquad s.t. \qquad C\beta = 0$$

for several convex loss functions $f(\cdot,\cdot)$. This includes the constrained Lasso, the constrained scaled Lasso, and sparse Huber M-estimators with linear equality constraints.

Statement of need

Currently, there is no Python package available that can solve these ubiquitous statistical estimation problems in a fast and efficient manner. `c-lasso` provides algorithmic strategies, including path and proximal splitting algorithms, to solve the underlying convex optimization problems with provable convergence guarantees. The `c-lasso` package is intended to fill the gap between popular Python tools such as `scikit-learn` which cannot solve these constrained problems and general-purpose optimization solvers such as `cvxpy` that do not scale well for these problems and/or are inaccurate. `c-lasso` can solve the estimation problems at fixed regularization level and across an entire regularization path and includes three model selection strategies for determining model parameter regularization levels: a theoretically derived fixed regularization, k-fold cross-validation, and stability selection. We show several use cases of the package, including an application of sparse log-contrast regression tasks for compositional microbiome data, and highlight the seamless integration into `R` via `reticulate`.

Functionalities

Installation and basic usage

`c-lasso` is available on pip and can be installed in the shell using

```
pip install c_lasso
```

Below is an example of the basic usage of `c-lasso` in Python.

```python
# Import the main class of the package
from classo import classo_problem

# Define a c-lasso problem instance with default setting,
# given data X, y, and constraints C.
problem  = classo_problem(X,y,C)

# Add possible modifications of the problem instance
...

# Solve the specified problem instance
problem.solve()

# Show the problem specification and the corresponding solution
print(problem)
print(problem.solution)
```

Statistical problem formulations

Depending on the type of and the prior assumptions on the data, the noise ϵ, and the model parameters, `c-lasso` allows for different estimation problem formulations. More specifically, the package can solve the following four regression-type and two classification-type formulations:

R1 Standard constrained Lasso regression:

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

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

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

The Journal of Open Source Software

***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 h_ρ as objective function for robust model fitting with an L_1 penalty and linear equality constraints on the β vector. The default parameter ρ is set to 1.345 (Huber, 1981).

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

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

$$\arg\min_{\beta\in\mathbb{R}^d,\sigma\in\mathbb{R}_0} \frac{\|X\beta - y\|^2}{\sigma} + \frac{n}{2}\sigma + \lambda \|\beta\|_1 \qquad s.t. \qquad C\beta = 0$$

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

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

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

$$\arg\min_{\beta\in\mathbb{R}^d,\sigma\in\mathbb{R}_0} \left(h_\rho\left(\frac{X\beta - y}{\sigma}\right) + n\right)\sigma + \lambda \|\beta\|_1 \qquad s.t. \qquad C\beta = 0$$

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

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

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

$$\arg\min_{\beta\in\mathbb{R}^d} \sum_{i=1}^{n} l(y_i x_i^\top \beta) + \lambda \|\beta\|_1 \qquad s.t. \qquad C\beta = 0$$

where x_i denotes the i^{th} row of X, $y_i \in \{-1, 1\}$, 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).

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

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

$$\arg\min_{\beta\in\mathbb{R}^d} \sum_{i=1}^{n} l_\rho(y_i x_i^\top \beta) + \lambda \left\|\beta\right\|_1 \qquad s.t. \qquad C\beta = 0\,.$$

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

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

This formulation can be activated in `c-lasso` as follows:

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

Optimization schemes

The problem formulations *R1*-*C2* require different algorithmic strategies for efficiently solving the underlying optimization problems. The `c-lasso` package implements four published algorithms with provable convergence guarantees. The package also includes novel algorithmic extensions to solve Huber-type problems efficiently using the mean-shift formulation (Mishra & Müller, 2019). The following algorithmic schemes are implemented:

- Path algorithms (*Path-Alg*): This algorithm follows the proposal in (Gaines, Kim, & Zhou, 2018; Jeon, Kim, Won, & Choi, 2020) and uses the fact that the solution path along is piecewise-affine (Rosset & Zhu, 2007). We also provide a novel efficient procedure that allows to derive the solution for the concomitant problem *R3* along the path with little computational overhead.
- Douglas-Rachford-type splitting method (*DR*): This algorithm can solve all regression problems *R1*-*R4*. It is based on Doulgas-Rachford splitting in a higher-dimensional product space and makes use of the proximity operators of the perspective of the LS objective (Combettes & Müller, 2020a, 2020b). The Huber problem with concomitant scale *R4* is reformulated as scaled Lasso problem with mean shift vector (Mishra & Müller, 2019) and thus solved in (n + d) dimensions.
- 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, extending 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.

- Projection-free primal-dual splitting method (*PF-PDS*): This algorithm is a special case of an algorithm proposed in (Combettes & Pesquet, 2012) (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.

The following table summarizes the available algorithms and their recommended use for each problem:

	Path-Alg	*DR*	*P-PDS*	*PF-PDS*
R1	use for large λ and path computation	use for small λ	possible	use for complex constraints
R2	use for large λ and path computation	use for small λ	possible	use for complex constraints
R3	use for large λ and path computation	use for small λ	-	-
R4	-	only option	-	-
C1	only option	-	-	-
C2	only option	-	-	-

The following Python snippet shows how to select a specific algorithm:

```
problem.numerical_method = "Path-Alg"
# Alternative options: "DR", "P-PDS", and "PF-PDS"
```

Computation modes and model selection

The c-lasso package provides several computation modes and model selection schemes for tuning the regularization parameter.

- *Fixed Lambda*: This setting lets the user choose a fixed parameter λ or a proportion $l \in [0,1]$ such that $\lambda = l \times \lambda_{\max}$. The default value is a scale-dependent tuning parameter that has been derived in (Shi, Zhang, & Li, 2016) and applied in (Combettes & Müller, 2020b).
- *Path Computation*: This setting allows the computation of a solution path for λ parameters in an interval $[\lambda_{\min}, \lambda_{\max}]$. The solution path is computed via the *Path-Alg* scheme or via warm-starts for other optimization schemes.
- *Cross Validation*: This setting allows the selection of the regularization parameter λ via k-fold cross validation for $\lambda \in [\lambda_{\min}, \lambda_{\max}]$. Both the Minimum Mean Squared Error (or Deviance) (MSE) and the "One-Standard-Error rule" (1SE) are available (Hastie, Tibshirani, & Friedman, 2009).
- *Stability Selection*: This setting allows the selection of the λ via stability selection (Combettes & Müller, 2020b; Lin, Shi, Feng, & Li, 2014; Meinshausen & Bühlmann, 2010). Three modes are available for the selection of variables over subsamples: selection at a fixed λ (Combettes & Müller, 2020b), selection of the q first variables entering the path (the default setting in c-lasso), and selection of the q largest coefficients (in absolute value) across the path (Meinshausen & Bühlmann, 2010).

The Python syntax to use a specific computation mode and model selection is exemplified below:

The Journal of Open Source Software

```
# Example how to do only 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

# c-lasso allows to specify multiple model selection schemes,
# e.g., adding stability selection
problem.model_selection.StabSel = True
```

Each model selection procedure has additional meta-parameters that are described in the Documentation.

Computational examples

Basic workflow using synthetic data

We illustrate the workflow of the `c-lasso` package on synthetic data using the built-in routine `random_data` which enables the generation of test problem instances with normally distributed data X, sparse coefficient vectors β, and constraints $C \in R^{k\times d}$.

Here, we use a problem instance with $n = 100$, $d = 100$, a β with five non-zero components, $\sigma = 0.5$, and a zero-sum contraint.

```
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)
```

We use formulation *R2* with $\rho = 1.5$, computation mode and model selections *Fixed Lambda* with $\lambda = 0.1\lambda_{\max}$, *Path computation*, and *Stability Selection* (as per default).

The corresponding output reads:

```
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
```

c-lasso allows standard visualization of the computed solutions, e.g., coefficient plots at fixed λ, the solution path, the stability selection profile at the selected λ, and the stability selection profile across the entire path.

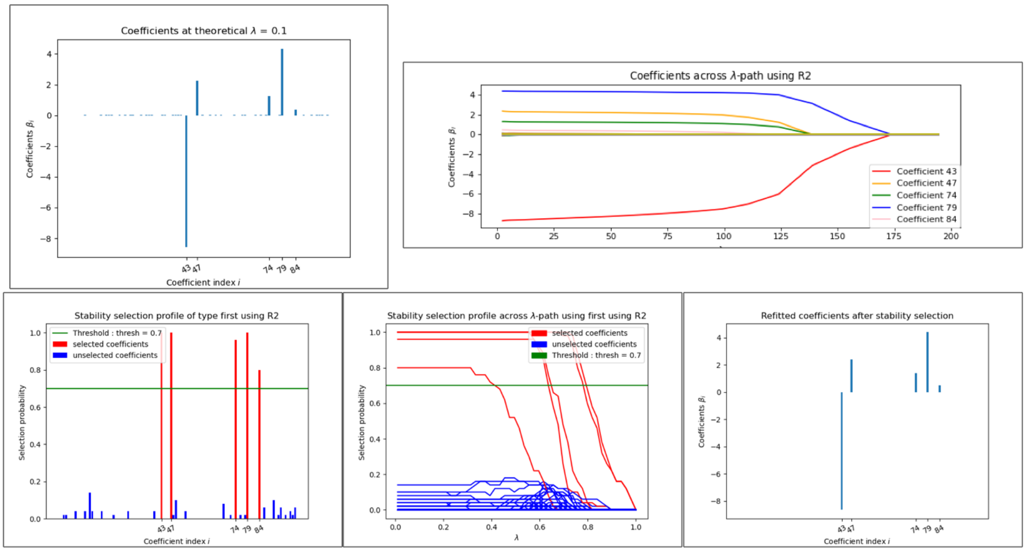

Figure 1: Visualizations after calling problem.solution

For this tuned example, the solutions at the fixed lambda and with stability selection recover the oracle solution. The solution vectors are stored in `problem.solution` and can be directly acccessed for each mode/model selection.

```
# Access to the estimated coefficient vector at a fixed lambda
problem.solution.LAMfixed.beta
```

Note that the run time for this $d = 100$-dimensional example for a single path computation is about 0.5 seconds on a standard Laptop.

Log-contrast regression on gut microbiome data

We next illustrate the application of `c-lasso` on the `COMBO` microbiome dataset (Combettes & Müller, 2020b; Lin et al., 2014; Shi et al., 2016), available in `c-lasso`'s data folder. We consider the computational approaches described in (Combettes & Müller, 2020b). The task is to predict the Body Mass Index (BMI) of $n = 96$ participants from $d = 45$ relative abundances of bacterial genera, absolute calorie and fat intake measurments. Below is the code snippet for this example, also included here.

```python
from classo import *

# Load microbiome and covariate data X
X0  = csv_to_mat('GeneraFilteredCounts.csv',begin=0).astype(float)
X_C = csv_to_mat('CaloriData.csv',begin=0).astype(float)
X_F = csv_to_mat('FatData.csv',begin=0).astype(float)


# Load BMI measurements y
y   = csv_to_mat('BMI.csv',begin=0).astype(float)[:,0]

# Load genus and covariate labels
labels  = csv_to_mat('GeneraPhylo.csv').astype(str)[:,-1]

# Normalize/transform data
y   = y - np.mean(y)
X_C = X_C - np.mean(X_C, axis=0)  #Covariate data (Calorie)
X_F = X_F - np.mean(X_F, axis=0)  #Covariate data (Fat)
X0 = clr(X0, 1 / 2).T

# Set up design matrix and zero-sum constraints for 45 genera
X = np.concatenate((X0, X_C, X_F, np.ones((len(X0), 1))), axis=1)
label = np.concatenate([labels,np.array(['Calorie','Fat','Bias'])])
C = np.ones((1,len(X[0])))
C[0,-1],C[0,-2],C[0,-3] = 0.,0.,0.

# Set up c-lassso problem
problem = classo_problem(X,y,C, label=label)

# Use formulation R3
problem.formulation.concomitant = True

# Use stability selection with theoretical lambda [Combettes & Müller, 2020b]
problem.model_selection.StabSel                       = True
problem.model_selection.StabSelparameters.method      = 'lam'
problem.solve()

# Use formulation R4
problem.formulation.huber = True
problem.formulation.concomitant = True

problem.solve()
```

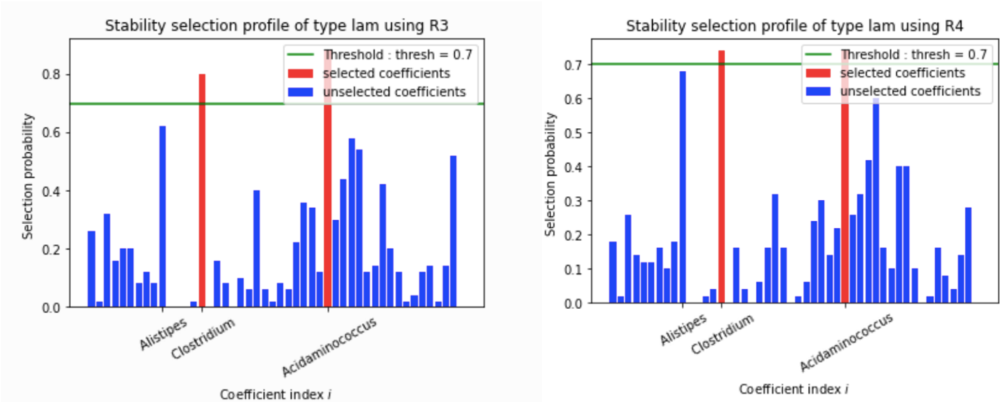

Figure 2: Stability selection profiles of problems R3/R4 on the COMBO data

Stability selection profiles of formulation *R3* (left) and *R4*(right) on the COMBO dataset, reproducing Figure 5a in (Combettes & Müller, 2020b).

Calling `c-lasso` in R

The `c-lasso` package can also be conveniently integrated into `R` using the `R` package `reticulate`. We refer to `reticulate`'s manual for technical details about connecting `python` environments and `R`. A successful interfacing is available in the `R` package `trac` (Bien, Yan, Simpson, & Müller, 2020).

The code snippet below shows how `c-lasso` is called in `R` to perform regression at a fixed λ $\lambda = 0.1\lambda_{\max}$. In `R`, X and C should be of `matrix` type, and y of `array` type.

```
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 coefficent vector
beta <- as.matrix(map_dfc(problem$solution$LAMfixed$beta, as.numeric))
```

Acknowledgements

This work of LS was conducted at and financially supported by the Center for Computational Mathematics (CCM), Flatiron Institute, New York, and the Institute of Computational Biology, Helmholtz Zentrum München. We thank Dr. Leslie Greengard (CCM and Courant Institute, NYU) for facilitating the initial contact between LS and CLM.

References

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

Bien, J., Yan, X., Simpson, L., & Müller, C. L. (2020). Tree-Aggregated Predictive Modeling of Microbiome Data. *bioRxiv*. doi:10.1101/2020.09.01.277632

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. (2020a). 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. (2020b). 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. L., & Pesquet, J.-C. (2012). 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

Huber, P. (1981). *Robust statistics*. John Wiley & Sons Inc. ISBN: 0-471-41805-6

James, G. M., Paulson, C., & Rusmevichientong, P. (2020). Penalized and constrained optimization: An application to high-dimensional website advertising. *Journal of the American Statistical Association*, *115*(529), 107–122. doi:10.1080/01621459.2019.1609970

Jeon, J.-J., Kim, Y., Won, S., & Choi, H. (2020). Primal path algorithm for compositional data analysis. *Computational Statistics & Data Analysis*, *148*, 106958. doi:https://doi.org/10.1016/j.csda.2020.106958

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. doi:10.1093/biomet/asu031

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. L. (2019). Robust Regression with Compositional Covariates. Retrieved from http://arxiv.org/abs/1909.04990

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

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