An Introduction to `classo`

Younghoon Kim Navonil Deb Sumanta Basu

December 03, 2024

Contents

Introduction

'cxreg' is a package that fits complex-valued penalized regression models via exact coordinate descent method. Just like a well-known package 'glmnet' (2010), the regularization path is computed for the linear regression with lasso penalty, called classo, at a grid of values for the regularization parameter lambda. The package includes methods for prediction an plotting, and functions for cross-validation.

The authors of cxreg are Navonil Deb and Sumanta Basu, with contribution from Younghoon Kim, and the R package is maintained by Younghoon Kim.

This vignette describes basic usage of functions related to classo model in 'cxreg' package in R. There is an original description of the algorithm that should be useful:

- "Regularized estimation of sparse spectral precision matrices"

'classo' solves the following problem. For $Y \in \mathbb{Z}^n$ and $X \in \mathbb{Z}^{n \times p}$,

$$\min_{\beta} \frac{1}{2n} \|Y - X\beta\|_2^2 + \lambda \|\beta\|_1,$$

over a grid of values of λ covering the entire range of possible solutions. Here, since the absolute of complex-values means a complex modulus, $\beta \in \mathbb{Z}^p$ so that the ℓ_1 penalty can be viewed as

$$\|\beta\|_1 = \sum_{j=1}^{p} |\beta_j| = \sum_{j=1}^{p} \|\Re(\beta_j) + \Im(\beta_j)\|_2,$$

which is a group Lasso with p groups, each of size 2.

Installation

Like other R packages on Github, it can be downloaded by the command:

```
library(devtools)
```

```
## Warning: package 'devtools' was built under R version 4.3.3
```

```
devtools::install_github("yk748/cxreg")
```

Quick Start: classo

The purpose of this section is to give users a general sense of the package regarding classo. We will briefly go over the main functions, basic operations and outputs. 'cxreg' can be loaded using the `library` command:

```
library(cxreg)
```

We load a set of data created beforehand for illustration:

```
data(QuickStartExample)
x <- QuickStartExample$x
y <- QuickStartExample$y
```

Note that 'x' is already standardized, which makes the columns in X orthogonal. The advantage of the standardization is described in the original paper (2024).

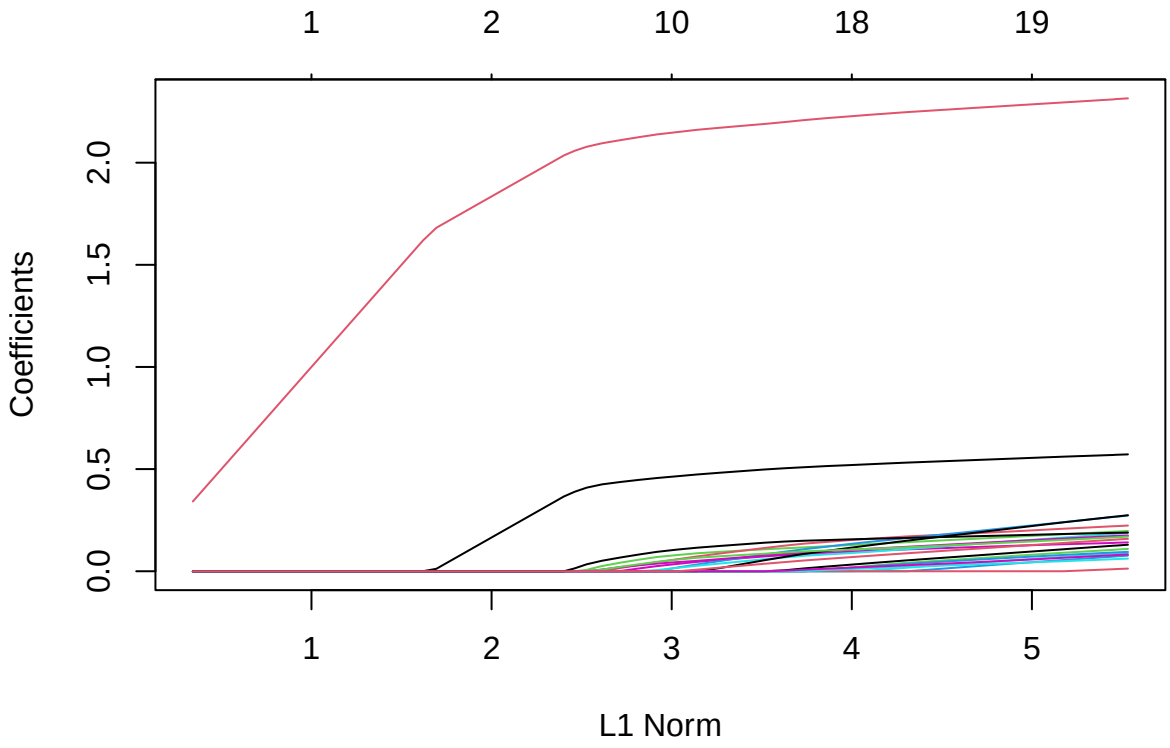
We fit the model using the most basic call to 'classo'.

```
fit <- classo(x, y)
```

'fit' is an object of class 'classo' that contains all the relevant information of the fitted model for further use. Various methods are provided for the object such as 'plot', 'coef', and 'predict', just like 'glmnet' package. However, currently 'print' is not provided since the degree of freedom is not computed through the functions.

We can visualize the coefficients by executing the 'plot' method:

```
plot(fit)
```



Each curve corresponds to a variable in complex modulus (i.e., absolute scale). It shows the path of its coefficients in absolute scale against the ℓ-norm of the whole coefficient vector as λ varies. Users may also wish to annotate the curves: this can be done by setting 'label = TRUE' in the plot command.

We can obtain the model coefficients at one or more λ's within the range of the sequence:

```
coef(fit, s = 0.1)
```

```
##                              s1
## V1   4.010342e-01-2.970314e-01i
## V2   9.892119e-01+1.954953e+00i
## V3   1.034847e-01-3.124244e-02i
## V4   0.000000e+00+0.000000e+00i
## V5   0.000000e+00+0.000000e+00i
## V6   2.288699e-02-6.983771e-02i
## V7  -5.446104e-05-7.775609e-04i
## V8  -1.126125e-01+2.626958e-02i
## V9   0.000000e+00+0.000000e+00i
## V10  8.401064e-02-2.252062e-04i
## V11 -3.892410e-02+4.572019e-02i
## V12  7.754518e-02-1.017528e-02i
## V13 -5.600182e-02-5.461492e-03i
## V14  0.000000e+00+0.000000e+00i
## V15 -8.654906e-02+6.313472e-03i
## V16  0.000000e+00+0.000000e+00i
## V17  0.000000e+00+0.000000e+00i
```

```
## V18 -1.818175e-05-4.676537e-04i
## V19 -9.352612e-02-1.049293e-01i
## V20 -3.743944e-02-5.189871e-03i
```

Users can also make predictions at specific λ's with new input data: Note that the third line in the following chunk is about standardization, whose purpose is mentioned above.

```
set.seed(29)
nx <- array(rnorm(5*20), c(5,20)) + (1+1i) * array(rnorm(5*20), c(5,20))
for (j in 1:20) nx[,j] <- nx[,j] / sqrt(mean(Mod(nx[,j])^2))
predict(fit, newx = nx, s = c(0.1, 0.05))
```

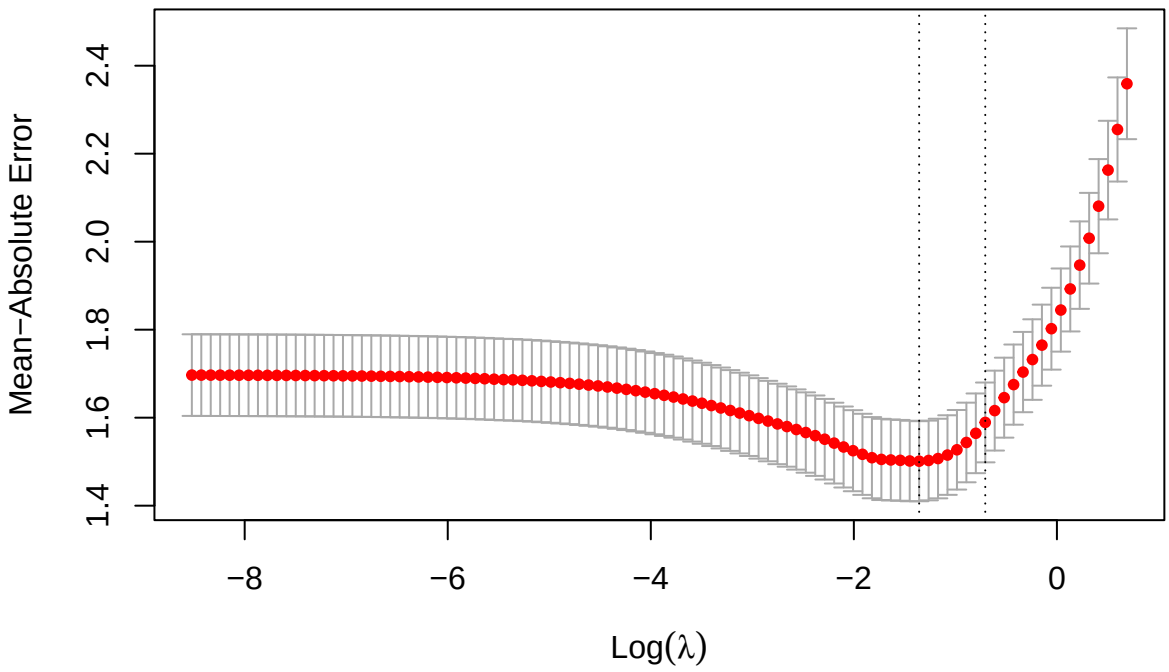
```
##                         s1                  s2
## [1,] -0.325501+3.646340i -0.396446+3.704557i
## [2,] -0.575236+1.797156i -0.588198+2.036628i
## [3,] -1.177101-1.152848i -1.279456-1.289822i
## [4,]  0.324236-1.769115i  0.434046-1.844779i
## [5,]  0.062806+2.286424i  0.049002+2.183437i
```

The function 'classo' returns a sequence of models for the users to choose from. In many cases, users may prefer the software to select one of them. Cross-validation is perhaps the simplest and most widely used method for that task. 'cv.classo' is the main function to do cross-validation here, along with various supporting methods such as plotting and prediction.

```
cvfit <- cv.classo(x, y)
```

'cv.classo} returns a 'cv.classo' object, a list with all the ingredients of the cross-validated fit.

```
plot(cvfit)
```



This plots the cross-validation curve (red dotted line) along with upper and lower standard deviation curves along the λ sequence (error bars). Two special values along the λ sequence are indicated by the vertical dotted lines. 'lambda.min' is the value of λ that gives minimum mean cross-validated error, while 'lambda.lse' is the value of λ that gives the most regularized model such that the cross-validated error is within one standard error of the minimum.

We can use the following code to get the value of 'lambda.min' and the model coefficients at that value of λ:

```
cvfit$lambda.min
```

```
## [1] 0.2574575
```

```
coef(cvfit, s = "lambda.min")
```

```
##                          s1
## V1   0.30900192-0.23668039i
## V2   0.94977201+1.82606394i
## V3   0.00000000+0.00000000i
## V4   0.00000000+0.00000000i
## V5   0.00000000+0.00000000i
## V6   0.00000000+0.00000000i
## V7   0.00000000+0.00000000i
## V8   0.00000000+0.00000000i
## V9   0.00000000+0.00000000i
## V10  0.00000000+0.00000000i
## V11  0.00000000+0.00000000i
```

```
## V12  0.00000000+0.00000000i
## V13  0.00000000+0.00000000i
## V14  0.00000000+0.00000000i
## V15  0.00000000+0.00000000i
## V16  0.00000000+0.00000000i
## V17  0.00000000+0.00000000i
## V18  0.00000000+0.00000000i
## V19 -0.01006017-0.01008821i
## V20  0.00000000+0.00000000i
```

To get the corresponding values at 'lambda.1se', simply replace 'lambda.min' with 'lambda.1se} above, or omit the 's' argument, since 'lambda.1se' is the default.

Note that unlike 'glmnet' package, the coefficients are not represented in sparse matrix format, rather they are in the dense format. This is because of the traits of complex values in the function.

Predictions can be made based on the fitted 'cv.glmnet' object as well. The code below gives predictions for the new input matrix 'newx' at 'lambda.min':

```
predict(cvfit, newx = x[1:5,], s = "lambda.min")
```

This concludes the basic usage of `classo`.

Deb, Navonil, Amy Kuceyeski, and Sumanta Basu. 2024. "Regularized Estimation of Sparse Spectral Precision Matrices." *arXiv Preprint arXiv:2401.11128*.

Friedman, Jerome, Trevor Hastie, and Rob Tibshirani. 2010. "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software* 33 (1): 1.