classo

Leo

Feb 25, 2020

CONTENTS

CHAPTER

ONE

FUNCTIONS OF THE PACKAGE

1.1 Random data

CLasso.**random_data** (*n*, *d*, *d_nonzero*, *k*, *sigma*, *zerosum=False*, *seed=False*, *classification=False*)

Generation of random matrices as data such that y = X.sol + sigma. noise

The data X is generated as a normal matrix The vector sol is generated randomly with a random support of size d_nonzero, and componants are projected random intergers between -10 and 10 on the kernel of C restricted to the support The vector y is then generated with X.dot(sol)+ sigma*noise , with noise a normal vector

Parameters

- **n** (*int*) – Number of sample, dimension of y
- **d** (*int*) – Number of variables, dimension of sol
- **d_nonzero** (*int*) – Number of non null componant of sol
- **k** (*int*) – Number of constraints, number of rows of C
- **sigma** (*float*) – size of standard error
- **zerosum** (*bool, optional*) – If True, then C is the all-one matrix with 1 row, independently of k
- **seed** (*bool or int, optional*) – Seed for random values, for an equal seed, the result will be the same. If set to False: pseudo-random vectors
- **classification** (*bool, optional*) – if True, then it returns sign(y) instead of y

Returns tuple of three ndarray that corresponds to the data : (X,C,y) ndarray : array corresponding to sol which is the real solution of the problem y = Xbeta + noise s.t. beta sparse and Cbeta = 0

Return type tuple

1.2 File converters

CLasso.**csv_to_mat** (*file*, *begin=1*, *header=None*)

Function to read a csv file and to create an ndarray with this

Parameters

- **file** (*str*) – Name of csv file
- **begin** (*int, optional*) – First colomn where it should read the matrix

- **header** (*None or int, optional*) – Same parameter as in the function `pandas.read_csv()`

Returns matrix of the csv file

Return type ndarray

`CLasso.`**`mat_to_np`**(*file*)

Function to read a mat file and to create an ndarray with this

Parameters **file** (*str*) – Name of mat file

Returns matrix of the mat file

Return type ndarray

1.3 Matrices normalization

`CLasso.`**`rescale`**(*matrices*)

Function that rescale the matrix and returns its scale

Substract the mean of y, then divides by its norm. Also divide each colomn of X by its norm. This will change the solution, not only by scaling it, because then the L1 norm will affect every component equally (and not only the variables with big size)

Parameters **matrices** (*tuple*) – tuple of three ndarray matrices corresponding to (X,C,y)

Returns tuple of the three corresponding matrices after normalization tuple : tuple of the three information one need to recover the initial data : lX (list of initial colomn-norms of X), ly (initial norm of y), my (initial mean of y)

Return type tuple

`CLasso.`**`clr`**(*array*, *coef=0.5*)

Centered-Log-Ratio transformation

Set all negaitve or null entry to a constant coef. Then compute the log of each component. Then substract the mean of each colomn on each colomn.

Parameters

- **array** (*ndarray*) – matrix nxd
- **coef** (*float, optional*) – Value to replace the zero values

Returns clr transformed matrix nxd

Return type ndarray

1.4 Theoretical lambda

`CLasso.`**`theoretical_lam`**(*n*, *d*)

Theoretical lambda as a function of the dimension of the problem

This function returns (with $\phi = erf$) :

$4/\sqrt{n}\phi^{-1}(1-2x)$ such that $x = 4/d(\phi^{-1}(1-2x)4 + \phi^{-1}(1-2x)^2)$

Which is the same (thanks to formula : $norm^{-1}(1-t) = \sqrt{2}\phi^{-1}(1-2t)$) as :

$\sqrt{2/n} * norm^{-1}(1-k/p)$ such that $k = norm^{-1}(1-k/p)^4 + 2norm^{-1}(1-k/p)^2$

Parameters

- **n** (*int*) – number of sample
- **d** (*int*) – number of variables

Returns theoretical lambda

Return type float

1.5 Numpy example

`numpy.zeros` (*shape*, *dtype=float*, *order='C'*)

Return a new array of given shape and type, filled with zeros.

Parameters

- **shape** (*int or tuple of ints*) – Shape of the new array, e.g., `(2, 3)` or `2`.
- **dtype** (*data-type, optional*) – The desired data-type for the array, e.g., *numpy.int8*. Default is *numpy.float64*.
- **order** (*{'C', 'F'}, optional, default: 'C'*) – Whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory.

Returns **out** – Array of zeros with the given shape, dtype, and order.

Return type ndarray

See also:

zeros_like() Return an array of zeros with shape and type of input.

empty() Return a new uninitialized array.

ones() Return a new array setting values to one.

full() Return a new array of given shape filled with value.

Examples

```
>>> np.zeros(5)
array([ 0.,  0.,  0.,  0.,  0.])
```

```
>>> np.zeros((5,), dtype=int)
array([0, 0, 0, 0, 0])
```

```
>>> np.zeros((2, 1))
array([[ 0.],
       [ 0.]])
```

```
>>> s = (2,2)
>>> np.zeros(s)
array([[ 0.,  0.],
       [ 0.,  0.]])
```

```
>>> np.zeros((2,), dtype=[('x', 'i4'), ('y', 'i4')]) # custom dtype
array([(0, 0), (0, 0)],
      dtype=[('x', '<i4'), ('y', '<i4')])
```

CHAPTER

TWO

DESCRIPTION OF THE CLASS CLASSO_PROBLEM

2.1 CLasso

2.2 Class classo_problem

class CLasso.**classo_problem**(*X*, *y*, *C='zero-sum'*, *labels=False*)

Class that contains all the information about the problem

Parameters

- **X** (*ndarray*) – Matrix representing the data of the problem
- **y** (*ndarray*) – Vector representing the output of the problem
- **C** (*str or ndarray, optional*) – Matrix of constraints to the problem. If it is 'zero-sum' then the corresponding attribute will be all-one matrix. Default value to 'zero-sum'
- **rescale** (*bool, optional*) – if True, then the function *rescale()* will be applied to data when solving the problem. Default value is 'False'

label

list of the labels of each variable. If set to False then there is no label

Type list or bool

data

object containing the data of the problem.

Type *classo_data*

formulation

object containing the info about the formulation of the minimization problem we solve.

Type *classo_formulation*

model_selection

object giving the parameters we need to do variable selection.

Type *classo_model_selection*

solution

object giving caracteristics of the solution of the model_selection that is asked. Before using the method solve() , its componant are empty/null.

Type *classo_solution*

`classo_problem.solve()`
Method that solve every model required in the attributes of the problem and update the attribute `problem.solution` with the characteristics of the solution.

2.3 Class classo_data

class `CLasso.classo_data`(*X*, *y*, *C*, *rescale=False*)
Class containing the data of the problem

Parameters

- **X** (*ndarray*) – Matrix representing the data of the problem
- **y** (*ndarray*) – Vector representing the output of the problem
- **C** (*str or array, optional*) – Matrix of constraints to the problem. If it is 'zero-sum' then the corresponding attribute will be all-one matrix.
- **rescale** (*bool, optional*) – if True, then the function *rescale()* will be applied to data when solving the problem

X
Matrix representing the data of the problem

Type ndarray

y
Vector representing the output of the problem

Type ndarray

C
Matrix of constraints to the problem. If it is 'zero-sum' then the corresponding attribute will be all-one matrix.

Type str or array, optional

rescale
if True, then the function *rescale()* will be applied to data when solving the problem

Type bool, optional

2.4 Class classo_formulation

class `CLasso.solver.classo_formulation`
Class containing the data of the problem

huber
True if the formulation of the problem should be robust Default value = False

Type bool

concomitant
True if the formulation of the problem should be with an M-estimation of sigma. Default value = True

Type bool

classification
True if the formulation of the problem should be classification (if yes, then it will not be concomitant) Default value = False

Type bool

rho
Value of rho for robust problem. Default value = 1.345

Type float

rho_classification
value of rho for huberized hinge loss function for classification (this parameter has to be negative). Default value = -1.

Type float

e
value of e in concomitant formulation. If 'n/2' then it becomes n/2 during the method solve(), same for 'n'. Default value : 'n' if huber formulation ; 'n/2' else

Type float or string

2.5 Class classo_model_selection

class CLasso.solver.**classo_model_selection**
Class containing the data of the problem

PATH
True if path should be computed. Default Value = False

Type bool

PATHparameters
object parameters to compute the lasso-path.

Type *PATHparameters*

CV
True if Cross Validation should be computed. Default Value = False

Type bool

CVparameters
object parameters to compute the cross-validation.

Type *CVparameters*

StabSel
True if Stability Selection should be computed. Default Value = True

Type boolean

StabSelparameters
object parameters to compute the stability selection.

Type *StabSelparameters*

LAMfixed
True if solution for a fixed lambda should be computed. Default Value = False

Type boolean

LAMfixedparameters
object parameters to compute the lasso for a fixed lambda

Type LAMparameters

2.6 Classes used in classo_model_selection

class `CLasso.solver.`**`PATHparameters`**
object parameters to compute the lasso-path.

`numerical_method`
name of the numerical method that is used, it can be : 'Path-Alg' (path algorithm) , 'P-PDS' (Projected primal-dual splitting method) , 'PF-PDS' (Projection-free primal-dual splitting method) or 'DR' (Douglas-Rachford-type splitting method) Default value : 'choose', which means that the function `choose_numerical_method()` will choose it accordingly to the formulation

Type str

`n_active`
if it is an integer, then the algo stop computing the path when n_active variables are actives. then the solution does not change from this point. Dafault value : False

Type int or bool

`lambdas`
list of lambdas for computinf lasso-path for cross validation on lambda. Default value : np.array([10**(-delta * float(i) / nlam) for i in range(0,nlam)]) with delta=2. and nlam = 40

Type numpy.ndarray

`plot_sigma`

class `CLasso.solver.`**`CVparameters`**
object parameters to compute the cross-validation.

`seed`
Seed for random values, for an equal seed, the result will be the same. If set to False/None: pseudo-random vectors Default value : None

Type bool or int, optional

`numerical_method`
name of the numerical method that is used, can be : 'Path-Alg' (path algorithm) , 'P-PDS' (Projected primal-dual splitting method) , 'PF-PDS' (Projection-free primal-dual splitting method) or 'DR' (Douglas-Rachford-type splitting method) Default value : 'choose', which means that the function `choose_numerical_method()` will choose it accordingly to the formulation

Type str

`lambdas`
list of lambdas for computinf lasso-path for cross validation on lambda. Default value : np.linspace(1., 1e-3, 500)

Type numpy.ndarray

`oneSE`
if set to True, the selected lambda if computed with method 'one-standard-error' Default value : True

Type bool

`Nsubsets`
number of subset in the cross validation method Dafault value : 5

Type int

class `CLasso.solver.`**`StabSelparameters`**
object parameters to compute the stability selection.

seed
Seed for random values, for an equal seed, the result will be the same. If set to False/None: pseudo-random vectors Default value : None

Type bool or int, optional

numerical_method
name of the numerical method that is used, can be : 'Path-Alg' (path algorithm) , 'P-PDS' (Projected primal-dual splitting method) , 'PF-PDS' (Projection-free primal-dual splitting method) or 'DR' (Douglas-Rachford-type splitting method) Default value : 'choose', which means that the function `choose_numerical_method()` will choose it accordingly to the formulation

Type str

lam
(only used if *method* = 'lam') lam for which the lasso should be computed. Default value : 'theoretical' which mean it will be equal to *theoretical_lam* once it is computed

Type float or str

true_lam
(only used if *method* = 'lam') True if the lambda given is the real lambda, False if it lambda/lambdamax which is between 0 and 1. If True and lam = 'theoretical' , then it will takes the value n*theoretical_lam. Default value : True

Type bool

theoretical_lam
(only used if *method* = 'lam') Theoretical lam. Default value : 0.0 (once it is not computed yet, it is computed thanks to the function *theoretical_lam()* used in `classo_problem.solve()`)

Type float

method
'first', 'lam' or 'max' depending on the type of stability selection we do. Default value : 'first'

Type str

B
number of subsample considered. Default value : 50

Type int

q
number of selected variable per subsample. Default value : 10

Type int

percent_nS
size of subsample relatively to the total amount of sample Default value : 0.5

Type float

lamin
lamin when computing the lasso-path for method 'max' Default value : 1e-2

Type float

hd
if set to True, then the 'max' will stop when it reaches n-k actives variables Default value : False

Type bool

threshold
threhold for stability selection Default value : 0.7

Type float

threshold_label
threshold to know when the label should be plot on the graph. Default value : 0.4

Type float

class `CLasso.solver.`**LAMfixedparameters**
object parameters to compute the lasso for a fixed lambda

numerical_method
name of the numerical method that is used, can be : 'Path-Alg' (path algorithm) , 'P-PDS' (Projected primal-dual splitting method) , 'PF-PDS' (Projection-free primal-dual splitting method) or 'DR' (Douglas-Rachford-type splitting method) Default value : 'choose', which means that the function `choose_numerical_method()` will choose it accordingly to the formulation

Type str

lam
lam for which the lasso should be computed. Default value : 'theoretical' which mean it will be equal to *`theoretical_lam`* once it is computed

Type float or str

true_lam
True if the lambda given is the real lambda, False if it lambda/lambdamax which is between 0 and 1. If True and lam = 'theoretical' , then it will takes the value n*theoretical_lam. Default value : True

Type bool

theoretical_lam
Theoretical lam Default value : 0.0 (once it is not computed yet, it is computed thanks to the function *`theoretical_lam()`* used in `classo_problem.solve()`)

Type float

2.7 Class classo_solution

class `CLasso.solver.`**classo_solution**

Class giving characteristics of the solution of the model_selection that is asked. Before using the method solve() , its componant are empty/null.

PATH
Solution components of the model PATH

Type *solution_PATH*

CV
Solution components of the model CV

Type *solution_CV*

StabelSel
Solution components of the model StabSel

Type *solution_StabSel*

LAMfixed
Solution components of the model LAMfixed

Type *solution_LAMfixed*

2.8 Classes used in classo_solution

class CLasso.solver.**solution_PATH**(*matrices*, *param*, *formulation*)

Class giving characteristics of the lasso-path computed, which also contains a method _repr_ that plot the graphic of this lasso-path

BETAS

SIGMAS

LAMBDAS

method

save

formulation

time

class CLasso.solver.**solution_CV**(*matrices*, *param*, *formulation*)

Class giving characteristics of the cross validation computed, which also contains a method _repr_() that plot the selected parameters and the solution of the not-sparse problem on the selected variables set It also contains a method gaphic(self, mse_max=1.,save=False) that computes the curve of validation error as a function of lambda

beta

sigma

xGraph

yGraph

standard_error

index_min

index_1SE

selected_param

refit

formulation

time

class CLasso.solver.**solution_StabSel**(*matrices*, *param*, *formulation*)

Class giving characteristics of the stability selection computed, which also contains a method _repr_() that plot the selected parameters, the solution of the not-sparse problem on the selected variables set, the stability plot with the evolution of it with lambda if the used method is 'first'

distribution

lambdas_path

selected_param

to_label

refit

formulation

time

class CLasso.solver.**solution_LAMfixed**(*matrices*, *param*, *formulation*)

Class giving characteristics of the lasso computed which also contains a method _repr_() that plot this solution.

beta

sigma

lambdamax

selected_param

refit

formulation

time

2.9 Example

CHAPTER

THREE

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

C