Coclust Documentation

Release 0.2.1

Francois Role, Stanislas Morbieu, Mohamed Nadif

Feb 10, 2019

Contents

Coclust provides both a Python package which implements several diagonal and non-diagonal co-clustering algorithms, and a ready to use script to perform co-clustering.

Co-clustering (also known as biclustering), is an important extension of cluster analysis since it allows to simultaneously groups objects and features in a matrix, resulting in both row and column clusters.

The *script* enables the user to process a dataset with co-clustering algorithms without writing Python code.

The Python package provides an *API* for Python developers. This API allows to use the algorithms in a pipeline with scikit-learn library for example.

coclust is distributed under the 3-Clause BSD license. It works with both Python 2.7 and Python 3.5.

CHAPTER 1

Installation

You can install **coclust** with all the dependencies with:

```
pip install "coclust[alldeps]"
```

It will install the following libraries:

- numpy
- scipy
- scikit-learn
- matplotlib

If you only want to use co-clustering algorithms and don't want to install visualization or evaluation dependencies, you can install it with:

```
pip install coclust
```

It will install the following required libraries:

- numpy
- scipy
- scikit-learn

1.1 Windows users

It is recommended to use a third party distribution to install the dependencies before installing coclust. For example, when using the Continuum distribution, go to the download site to get and double-click the graphical installer. Then, enter `pip install coclust` at the command line.

1.2 Linux users

It is recommended to install the dependencies with your package manager. For example, on Ubuntu or Debian:

```
sudo apt-get install python-numpy python-scipy python-sklearn python-matplotlib
sudo pip install coclust
```

1.2.1 Performance note

OpenBLAS provides a fast multi-threaded implementation, you can install it with:

```
sudo apt-get install libopenblas-base
```

If other implementations are installed on your system, you can select OpenBLAS with:

```
sudo update-alternatives --config libblas.so.3
```

1.3 Running the tests

In order to run the tests, you have to install nose, for example with:

```
pip install nose
```

You also have to get the datasets used for the tests:

```
git clone https://github.com/franrole/cclust_package.git
```

And then, run the tests:

```
cd cclust_package
nosetests --with-coverage --cover-inclusive --cover-package=coclust
```

CHAPTER 2

Examples

The datasets used here are available at:

https://github.com/franrole/cclust_package/tree/master/datasets

2.1 Basic usage

In the following example, the CSTR dataset is loaded from a Matlab matrix using the SciPy library. The data is stored in X and a co-clustering model using direct maximisation of the modularity is then fitted with 4 clusters. The modularity is printed and the predicted row labels and column labels are retrieved for further exploration or evaluation.

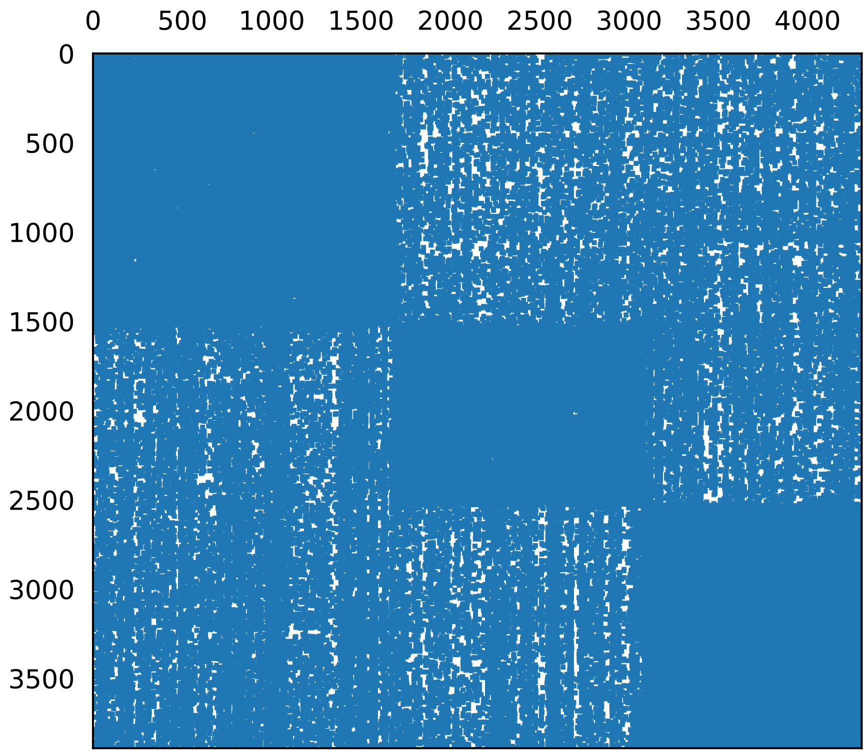
```
from scipy.io import loadmat
from coclust.coclustering import CoclustMod

file_name = "../datasets/cstr.mat"
matlab_dict = loadmat(file_name)
X = matlab_dict['fea']

model = CoclustMod(n_clusters=4)
model.fit(X)

print(model.modularity)
predicted_row_labels = model.row_labels_
predicted_column_labels = model.column_labels_
```

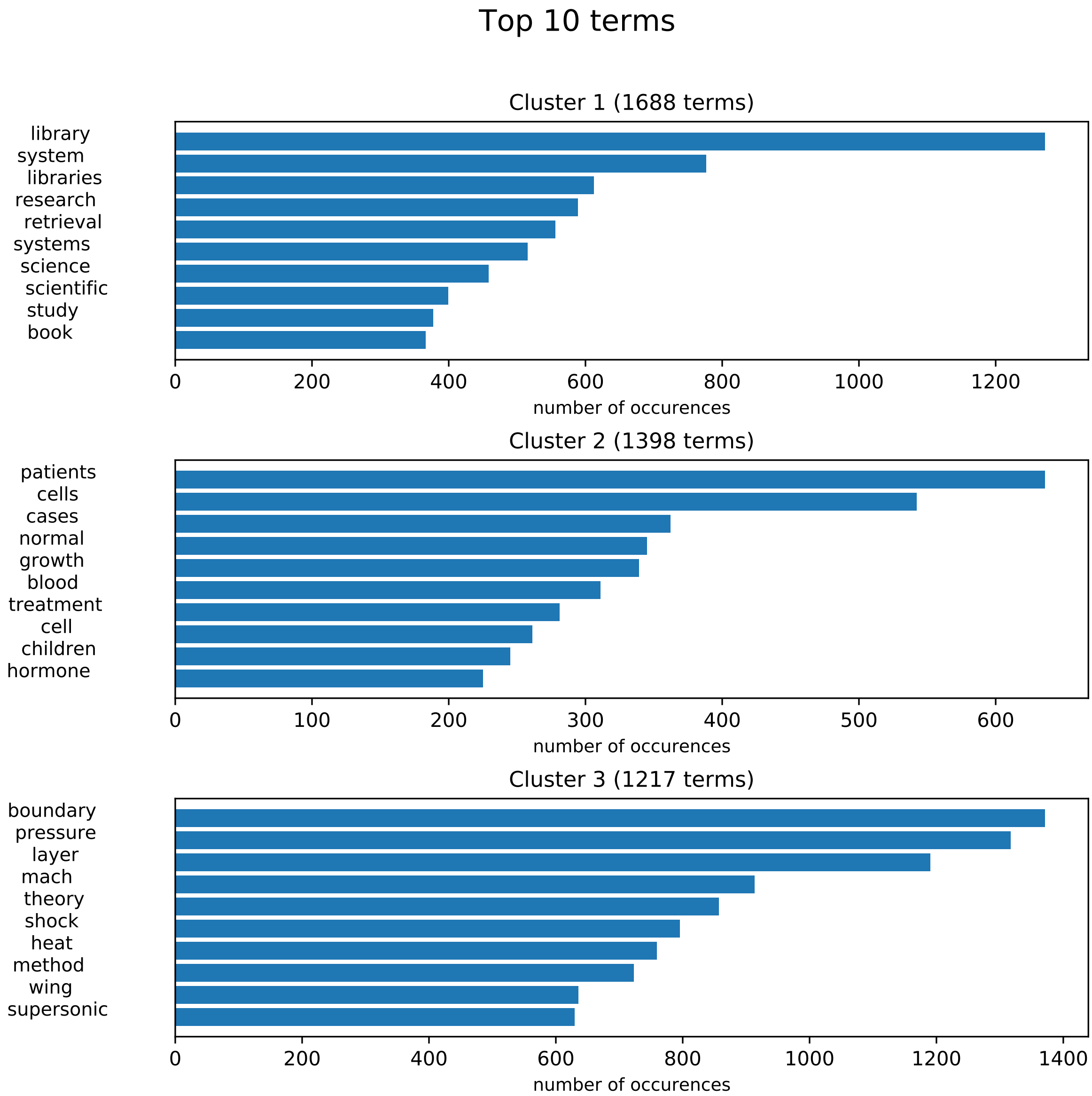
For example, the normalized mutual information score is computed using the scikit-learn library:

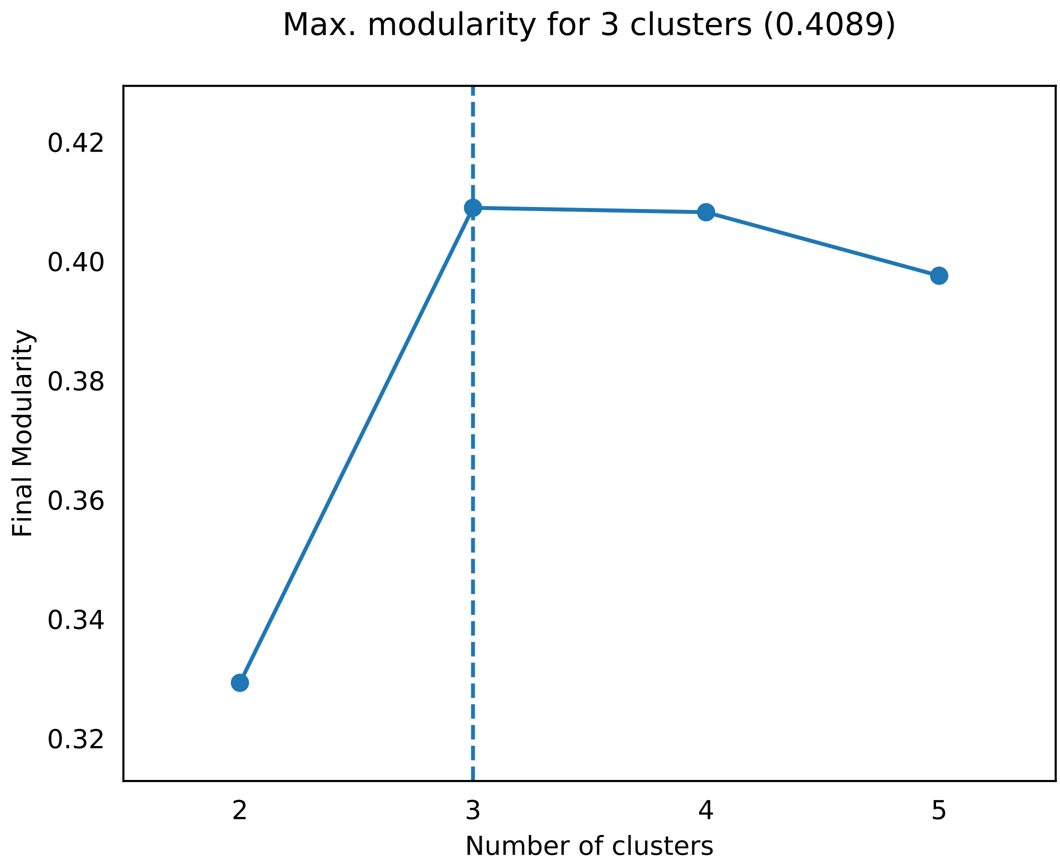
```
from sklearn.metrics.cluster import normalized_mutual_info_score as nmi

true_row_labels = matlab_dict['gnd'].flatten()

print(nmi(true_row_labels, predicted_row_labels))
```

2.2 Advanced usage overview

```
from coclust.io.data_loading import load_doc_term_data
from coclust.visualization import (plot_reorganized_matrix,
                                   plot_cluster_top_terms,
                                   plot_max_modularities)
from coclust.evaluation.internal import best_modularity_partition
from coclust.coclustering import CoclustMod

# read data
path = '../datasets/classic3_coclustFormat.mat'
doc_term_data = load_doc_term_data(path)
X = doc_term_data['doc_term_matrix']
labels = doc_term_data['term_labels']

# get the best co-clustering over a range of cluster numbers
clusters_range = range(2, 6)
model, modularities = best_modularity_partition(X, clusters_range, n_rand_init=1)

# plot the reorganized matrix
plot_reorganized_matrix(X, model)

# plot the top terms
n_terms = 10
plot_cluster_top_terms(X, labels, n_terms, model)

# plot the modularities over the range of cluster numbers
plot_max_modularities(modularities, range(2, 6))
```




Top 10 terms
Cluster 1 (1688 terms)
library
system
libraries
research
retrieval
systems
science
scientific
study
book
0
200
400
600
800
1000
1200
number of occurences
Cluster 2 (1398 terms)
patients
cells
cases
normal
growth
blood
treatment
cell
children
hormone
0
100
200
300
400
500
600
number of occurences
Cluster 3 (1217 terms)
boundary
pressure
layer
mach
theory
shock
heat
method
wing
supersonic
0
200
400
600
800
1000
1200
1400
number of occurences



2.3 scikit-learn pipeline

```
from coclust.coclustering import CoclustInfo

from sklearn.datasets import fetch_20newsgroups
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.metrics.cluster import normalized_mutual_info_score

categories = [
    'rec.motorcycles',
    'rec.sport.baseball',
    'comp.graphics',
    'sci.space',
    'talk.politics.mideast'
]

ng5 = fetch_20newsgroups(categories=categories, shuffle=True)

true_labels = ng5.target

pipeline = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('coclust', CoclustInfo()),
])

pipeline.set_params(coclust__n_clusters=5)
```

(continues on next page)

(continued from previous page)

```
pipeline.fit(ng5.data)

predicted_labels = pipeline.named_steps['coclust'].row_labels_

nmi = normalized_mutual_info_score(true_labels, predicted_labels)

print(nmi)
```

2.4 More examples

More examples are available as notebooks:

https://github.com/franrole/cclust_package/tree/master/demo

CHAPTER 3

Python API

3.1 Coclustering

The `coclust.coclustering` module gathers implementations of co-clustering algorithms.

3.1.1 Classes

Each of the following classes implements a co-clustering algorithm:

`coclust.coclustering.CoclustMod`([...])	Co-clustering by direct maximization of graph modularity.
`coclust.coclustering.CoclustSpecMod`([...])	Co-clustering by spectral approximation of the modularity matrix.
`coclust.coclustering.CoclustInfo`([...])	Information-Theoretic Co-clustering.

coclust.coclustering.CoclustMod

class coclust.coclustering.**CoclustMod**(*n_clusters=2*, *init=None*, *max_iter=20*, *n_init=1*, *tol=1e-09*, *random_state=None*)

Co-clustering by direct maximization of graph modularity.

Parameters

- **n_clusters** (*int, optional, default: 2*) – Number of co-clusters to form
- **init** (*numpy array or scipy sparse matrix, shape (n_features, n_clusters), optional, default: None*) – Initial column labels
- **max_iter** (*int, optional, default: 20*) – Maximum number of iterations
- **n_init** (*int, optional, default: 1*) – Number of time the algorithm will be run with different initializations. The final results will be the best output of *n_init* consecutive runs in terms of modularity.

- **random_state** (*integer or numpy.RandomState, optional*) – The generator used to initialize the centers. If an integer is given, it fixes the seed. Defaults to the global numpy random number generator.
- **tol** (*float, default: 1e-9*) – Relative tolerance with regards to modularity to declare convergence

row_labels_
array-like, shape (n_rows,) – Bicluster label of each row

column_labels_
array-like, shape (n_cols,) – Bicluster label of each column

modularity
float – Final value of the modularity

modularities
list – Record of all computed modularity values for all iterations

References

- Ailem M., Role F., Nadif M., Co-clustering Document-term Matrices by Direct Maximization of Graph Modularity. CIKM 2015: 1807-1810

fit (*X, y=None*)
Perform co-clustering by direct maximization of graph modularity.

Parameters X (*numpy array or scipy sparse matrix, shape=(n_samples, n_features)*) – Matrix to be analyzed

get_assignment_matrix (*kind, i*)
Returns the indices of 'best' i cols of an assignment matrix (row or column).

Parameters kind (*string*) – Assignment matrix to be used: rows or cols

Returns Matrix containing the i 'best' columns of a row or column assignment matrix

Return type numpy array or scipy sparse matrix

get_indices (*i*)
Give the row and column indices of the i'th co-cluster.

Parameters i (*integer*) – Index of the co-cluster

Returns (row indices, column indices)

Return type (list, list)

get_params (*deep=True*)
Get parameters for this estimator.

Parameters deep (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns params – Parameter names mapped to their values.

Return type mapping of string to any

get_shape (*i*)
Give the shape of the i'th co-cluster.

Parameters i (*integer*) – Index of the co-cluster

Returns (number of rows, number of columns)

Return type (int, int)

get_submatrix (*m, i*)
Give the submatrix corresponding to co-cluster i.

Parameters

- **m** (*X : numpy array or scipy sparse matrix*) – Matrix from which the block has to be extracted
- **i** (*integer*) – index of the co-cluster

Returns Submatrix corresponding to co-cluster i

Return type numpy array or scipy sparse matrix

set_params(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Returns

Return type self

coclust.coclustering.CoclustSpecMod

class coclust.coclustering.**CoclustSpecMod**(*n_clusters=2*, *max_iter=20*, *tol=1e-09*, *n_init=1*, *random_state=None*)

Co-clustering by spectral approximation of the modularity matrix.

Parameters

- **n_clusters** (*int, optional, default: 2*) – Number of co-clusters to form
- **max_iter** (*int, optional, default: 20*) – Maximum number of iterations
- **n_init** (*int, optional, default: 10*) – Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of *n_init* consecutive runs in terms of inertia.
- **random_state** (*integer or numpy.RandomState, optional*) – The generator used to initialize the centers. If an integer is given, it fixes the seed. Defaults to the global numpy random number generator.
- **tol** (*float, default: 1e-9*) – Relative tolerance with regards to criterion to declare convergence

row_labels_

array-like, shape (n_rows,) – Bicluster label of each row

column_labels_

array-like, shape (n_cols,) – Bicluster label of each column

References

- Labiod L., Nadif M., ICONIP'11 Proceedings of the 18th international conference on Neural Information Processing - Volume Part II Pages 700-708

fit(*X*, *y=None*)

Perform co-clustering by spectral approximation of the modularity matrix

Parameters X (*numpy array or scipy sparse matrix, shape=(n_samples, n_features)*) – Matrix to be analyzed

get_indices(*i*)

Give the row and column indices of the i'th co-cluster.

Parameters **i** (*integer*) – Index of the co-cluster

Returns (row indices, column indices)

Return type (list, list)

get_params (*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

get_shape (*i*)
Give the shape of the i'th co-cluster.

Parameters **i** (*integer*) – Index of the co-cluster

Returns (number of rows, number of columns)

Return type (int, int)

get_submatrix (*m, i*)
Give the submatrix corresponding to co-cluster i.

Parameters

- **m** (*X : numpy array or scipy sparse matrix*) – Matrix from which the block has to be extracted
- **i** (*integer*) – index of the co-cluster

Returns Submatrix corresponding to co-cluster i

Return type numpy array or scipy sparse matrix

set_params (***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Returns

Return type self

coclust.coclustering.CoclustInfo

class coclust.coclustering.**CoclustInfo** (*n_row_clusters=2, n_col_clusters=2, init=None, max_iter=20, n_init=1, tol=1e-09, random_state=None*)

Information-Theoretic Co-clustering.

Parameters

- **n_row_clusters** (*int, optional, default: 2*) – Number of row clusters to form
- **n_col_clusters** (*int, optional, default: 2*) – Number of column clusters to form
- **init** (*numpy array or scipy sparse matrix, shape (n_features, n_clusters), optional, default: None*) – Initial column labels

- **max_iter** (*int, optional, default: 20*) – Maximum number of iterations
- **n_init** (*int, optional, default: 1*) – Number of time the algorithm will be run with different initializations. The final results will be the best output of *n_init* consecutive runs.
- **random_state** (*integer or numpy.RandomState, optional*) – The generator used to initialize the centers. If an integer is given, it fixes the seed. Defaults to the global numpy random number generator.
- **tol** (*float, default: 1e-9*) – Relative tolerance with regards to criterion to declare convergence

row_labels_
array-like, shape (n_rows,) – Bicluster label of each row

column_labels_
array-like, shape (n_cols,) – Bicluster label of each column

delta_kl_
array-like, shape (k,l) – Value $\frac{p_{kl}}{p_{k.} \times p_{.l}}$ for each row cluster k and column cluster l

fit (*X*, *y=None*)
Perform co-clustering.

Parameters X (*numpy array or scipy sparse matrix, shape=(n_samples, n_features)*) – Matrix to be analyzed

get_col_indices (*i*)
Give the column indices of the i'th co-cluster.

Parameters i (*integer*) – Index of the i'th column cluster

Returns list of column indices

Return type list

get_params (*deep=True*)
Get parameters for this estimator.

Parameters deep (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns params – Parameter names mapped to their values.

Return type mapping of string to any

get_row_indices (*i*)
Give the row indices of the i'th co-cluster.

Parameters i (*integer*) – Index of the i'th row cluster

Returns list of row indices

Return type list

get_shape (*i*, *j*)

Give the shape of block corresponding to the i'th row cluster and the j'th column cluster.

Parameters

- **i** (*integer*) – Index of the row cluster
- **j** (*integer*) – Index of the column cluster

Returns (number of rows, number of columns)

Return type (int, int)

get_submatrix (*m*, *i*, *j*)

Give the submatrix corresponding to row cluster i and column cluster j.

Parameters

- **m** (*X : numpy array or scipy sparse matrix*) – Matrix from which the block has to be extracted
- **i** (*integer*) – index of the row cluster
- **j** (*integer*) – index of the col cluster

Returns Submatrix corresponding to row cluster i and column cluster j

Return type numpy array or scipy sparse matrix

set_params (***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Returns

Return type self

3.1.2 User guide

coclust.coclustering.CoclustMod and *coclust.coclustering.CoclustSpecMod* are diagonal co-clustering algorithms whereas *coclust.coclustering.CoclustInfo* is a non-diagonal co-clustering algorithm.

3.2 Clustering

The *coclust.clustering* module provides clustering algorithms.

class coclust.clustering.**SphericalKmeans** (*n_clusters=2*, *init=None*, *max_iter=20*, *n_init=1*, *tol=1e-09*, *random_state=None*, *weighting=True*)

Spherical k-means clustering.

Parameters

- **n_clusters** (*int, optional, default: 2*) – Number of clusters to form
- **init** (*numpy array or scipy sparse matrix, shape (n_features, n_clusters), optional, default: None*) – Initial column labels
- **max_iter** (*int, optional, default: 20*) – Maximum number of iterations
- **n_init** (*int, optional, default: 1*) – Number of time the algorithm will be run with different initializations. The final results will be the best output of *n_init* consecutive runs.
- **random_state** (*integer or numpy.RandomState, optional*) – The generator used to initialize the centers. If an integer is given, it fixes the seed. Defaults to the global numpy random number generator.
- **tol** (*float, default: 1e-9*) – Relative tolerance with regards to criterion to declare convergence

- **weighting** (*boolean, default: True*) – Flag to activate or deactivate TF-IDF weighting

labels_
array-like, shape (n_rows,) – cluster label of each row

criterion
float – criterion obtained from the best run

criterions
list of floats – sequence of criterion values during the best run

fit (*X, y=None*)
Perform clustering.

Parameters X (*numpy array or scipy sparse matrix, shape=(n_samples, n_features)*) – Matrix to be analyzed

3.2.1 Spherical k-means

coclust.clustering.spherical_kmeans provides an implementation of the spherical k-means algorithm.

class coclust.clustering.spherical_kmeans.**SphericalKmeans** (*n_clusters=2, init=None, max_iter=20, n_init=1, tol=1e-09, random_state=None, weighting=True*)

Spherical k-means clustering.

Parameters

- **n_clusters** (*int, optional, default: 2*) – Number of clusters to form
- **init** (*numpy array or scipy sparse matrix, shape (n_features, n_clusters), optional, default: None*) – Initial column labels
- **max_iter** (*int, optional, default: 20*) – Maximum number of iterations
- **n_init** (*int, optional, default: 1*) – Number of time the algorithm will be run with different initializations. The final results will be the best output of *n_init* consecutive runs.
- **random_state** (*integer or numpy.RandomState, optional*) – The generator used to initialize the centers. If an integer is given, it fixes the seed. Defaults to the global numpy random number generator.
- **tol** (*float, default: 1e-9*) – Relative tolerance with regards to criterion to declare convergence
- **weighting** (*boolean, default: True*) – Flag to activate or deactivate TF-IDF weighting

labels_
array-like, shape (n_rows,) – cluster label of each row

criterion
float – criterion obtained from the best run

criterions
list of floats – sequence of criterion values during the best run

fit (*X*, *y=None*)
Perform clustering.

Parameters X (*numpy array or scipy sparse matrix, shape=(n_samples, n_features)*) – Matrix to be analyzed

3.3 Input and output

The *coclust.io* module provides functions to load data and check if it is correct to be given as input of a clustering or co-clustering algorithm.

3.3.1 Data loading

The *coclust.io.data_loading* module provides functions to load data from files of different types.

`coclust.io.data_loading.`**`load_doc_term_data`**(*data_filepath*, *term_labels_filepath=None*, *doc_labels_filepath=None*)
Load cooccurence data from a .[. . .]sv or a .mat file.

The expected formats are:

- `(data_filepath).[...]sv`: three [. . .] separated columns:

  1st line:

  – 1st column: number of documents

  – 2nd column: number of words

  Other lines:

  – 1st column: document index

  – 2nd column: word index

  – 3rd column: word counts

- `(data_filepath).mat`: matlab file with fields:

  – `'doc_term_matrix'`: `scipy.sparse.csr_matrix` of shape (#docs, #terms)

  – `'doc_labels'`: list of int (len = #docs)

  – `'term_labels'`: list of string (len = #terms)

  If the key `'doc_term_matrix'` is not found, data loading fails. If the key `'doc_labels'` or `'term_labels'` are missing, a warning message is displayed.

Term and doc labels can be separatly loaded from a one column .[x]svl.txt file:

- **(term_labels_filepath).[x]svl.txt:** one column, one term label per row. The row index is assumed to correspond to the term index in the (columns of the) co-occurrence data matrix.
- **(doc_labels_filepath).[x]svl.txt:** one column, one document label per row. The row index is assumed to correspond to the non zero value number read by row from the co-occurrence data matrix.

Parameters file_path (*string*) – Path to file that contains the cooccurence data

Returns

- `'doc_term_matrix'`: `scipy.sparse.csr_matrix` of shape (#docs, #terms)
- `'doc_labels'`: list of int (#docs)
- `'term_labels'`: list of string (#terms)

Return type a dictionnary

Raises `ValueError` – If the input file is not found or if its content is not correct.

Example

```
>>> dict = load_doc_term_data('../datasets/classic3.csv')
>>> dict['doc_term_matrix'].shape
(3891, 4303)
```

3.3.2 Input checking

The *coclust.io.input_checking* module provides functions to check input matrices.

coclust.io.input_checking.**check_array**(*a*, *pos=True*)

Check if an array contains numeric values with non empty rows nor columns.

Parameters

- **a** – The input array
- **pos** (*bool*) – If `True`, check if the values are positives

Raises

- `TypeError` – If the array is not a Numpy/SciPy array or matrix or if the values are not numeric.
- `ValueError` – If the array contains empty rows or columns or contains NaN values, or negative values (if `pos` is `True`).

coclust.io.input_checking.**check_numbers**(*matrix*, *n_clusters*)

Check if the given matrix has enough rows and columns for the given number of co-clusters.

Parameters

- **matrix** – The input matrix
- **n_clusters** (*int*) – Number of co-clusters

Raises `ValueError` – If the data matrix has not enough rows or columns.

coclust.io.input_checking.**check_numbers_clustering**(*matrix*, *n_clusters*)

Check if the given matrix has enough rows and columns for the given number of clusters.

Parameters

- **matrix** – The input matrix
- **n_clusters** (*int*) – Number of clusters

Raises `ValueError` – If the data matrix has not enough rows or columns.

coclust.io.input_checking.**check_numbers_non_diago**(*matrix*, *n_row_clusters*, *n_col_clusters*)

Check if the given matrix has enough rows and columns for the given number of row and column clusters.

Parameters

- **matrix** – The input matrix
- **n_row_clusters** (*int*) – Number of row clusters
- **n_col_clusters** (*int*) – Number of column clusters

Raises `ValueError` – If the data matrix has not enough rows or columns.

coclust.io.input_checking.**check_positive**(*X*)

Check if all values are positives.

Parameters **X** (*numpy array or scipy sparse matrix*) – Matrix to be analyzed

Raises `ValueError` – If the matrix contains negative values.

Returns X

Return type numpy array or scipy sparse matrix

3.3.3 Jupyter and IPython Notebook utilities

The *coclust.io.notebook* module provides functions to manage input and output in the evaluation notebook.

`coclust.io.notebook.`**`input_with_default_int`**(*prompt*, *prefill*)
Prompt an int.

Parameters

- **prompt** (*string*) – The message printed before the field.
- **prefill** (*int*) – The default value.

Returns The value entered by the user or the default value.

Return type int

`coclust.io.notebook.`**`input_with_default_str`**(*prompt*, *prefill*)
Prompt a string.

Parameters

- **prompt** (*string*) – The message printed before the field.
- **prefill** (*string*) – The default value.

Returns The value entered by the user or the default value.

Return type string

3.4 Evaluation

The *coclust.evaluation* module provides functions to evaluate the results of clustering or co-clustering algorithms.

3.4.1 Internal measures

The *coclust.evaluation.internal* module provides functions to evaluate clustering or co-clustering given internal criteria.

`coclust.evaluation.internal.`**`best_modularity_partition`**(*in_data*, *nbr_clusters_range*, *n_rand_init=1*)
Evaluate the best partition over a range of number of cluster using co-clustering by direct maximization of graph modularity.

Parameters

- **in_data** (*numpy array or scipy sparse matrix, shape=(n_samples, n_features)*) – Matrix to be analyzed
- **nbr_clusters_range** – Number of clusters to be evaluated
- **n_rand_init** – Number of time the algorithm will be run with different initializations

Returns

- **tmp_best_model** (*coclust.coclustering.CoclustMod*) – model with highest final modularity
- **tmp_max_modularities** (*list*) – final modularities for all evaluated partitions

3.4.2 External measures

The *coclust.evaluation.external* module provides functions to evaluate clustering or co-clustering results with external information such as the true labeling of the clusters.

coclust.evaluation.external.**accuracy**(*true_row_labels*, *predicted_row_labels*)
Get the best accuracy.

Parameters

- **true_row_labels** (*array-like*) – The true row labels, given as external information
- **predicted_row_labels** (*array-like*) – The row labels predicted by the model

Returns Best value of accuracy

Return type float

3.5 Visualization

The *coclust.visualization* module provides functions to visualize different measures or data.

3.5.1 General functions

coclust.visualization.plot_cluster_top_terms(...)	Plot the top terms for each cluster.
coclust.visualization.get_term_graph(X, ...)	Get a graph of terms.
coclust.visualization.plot_cluster_sizes(model)	Plot the sizes of the clusters.
coclust.visualization.plot_reorganized_matrix(X, ...)	Plot the reorganized matrix.
coclust.visualization.plot_confusion_matrix(cm)	Plot a confusion matrix.

coclust.visualization.plot_cluster_top_terms()

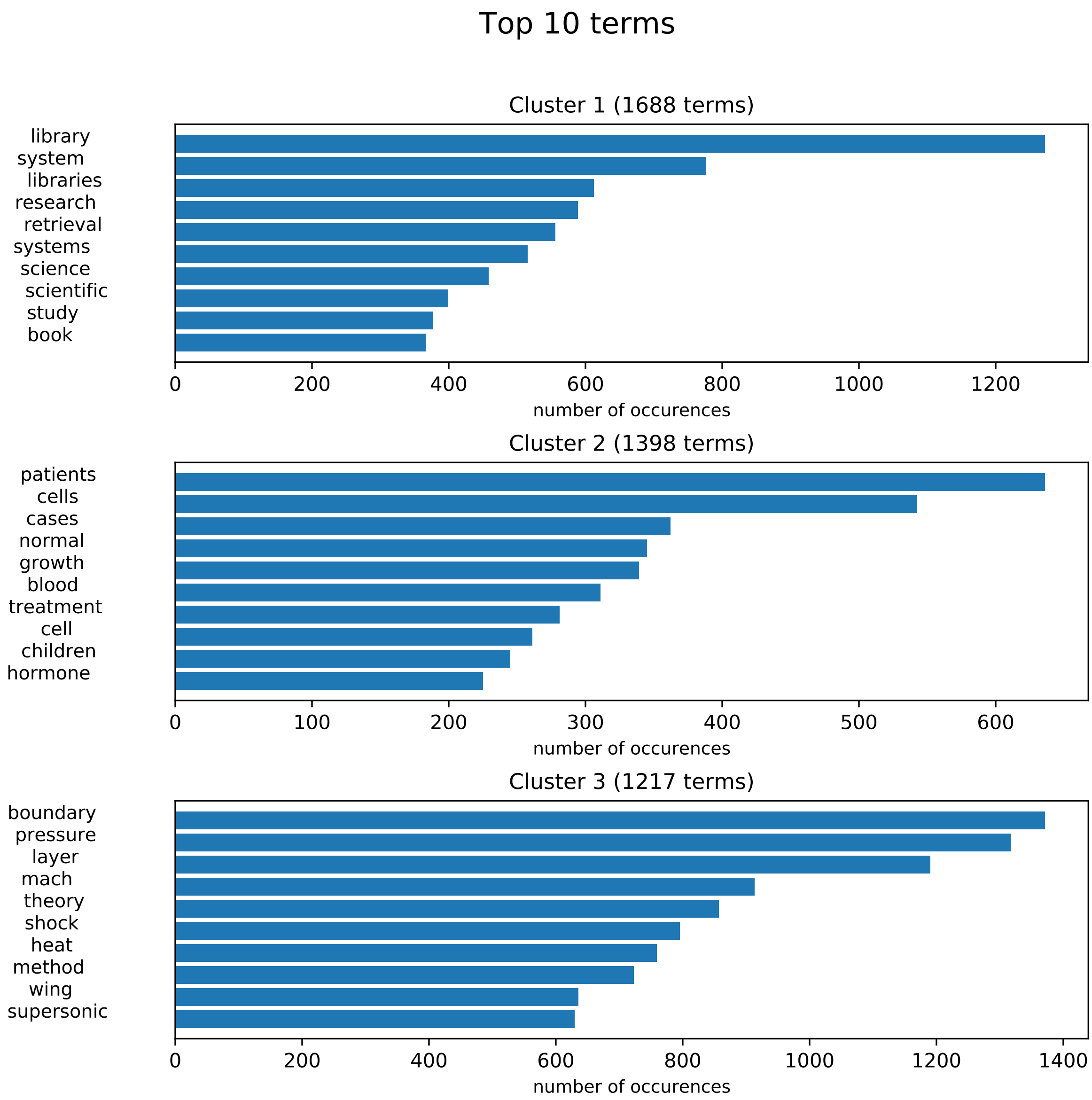
coclust.visualization.**plot_cluster_top_terms**(*in_data*, *all_terms*, *nb_top_terms*, *model*)
Plot the top terms for each cluster.

Parameters

- **in_data** (*numpy array or scipy sparse matrix, shape=(n_samples, n_features)*) –
- **all_terms** (*list of string*) – list of all terms from the original data set
- **nb_top_terms** (*int*) – number of top terms to be displayed per cluster
- **model** (coclust.coclustering.BaseDiagonalCoclust) – a co-clustering model

Example

```
>>> plot_cluster_top_terms(in_data, all_terms, nb_top_terms, model)
```



coclust.visualization.get_term_graph()

coclust.visualization.**get_term_graph**(*X*, *model*, *terms*, *n_cluster*, *n_top_terms=10*, *n_neighbors=2*, *stopwords=[]*)

Get a graph of terms.

Parameters

- **X** – input matrix
- **model** (`coclust.coclustering.BaseDiagonalCoclust`) – a co-clustering model
- **terms** (*list of string*) – list of terms
- **n_cluster** (*int*) – Id of the cluster

- **n_top_terms** (*int, optional, default: 10*) – Number of terms
- **n_neighbors** (*int, optional, default: 2*) – Number of neighbors
- **stopwords** (*list of string, optional, default: []*) – Words to remove

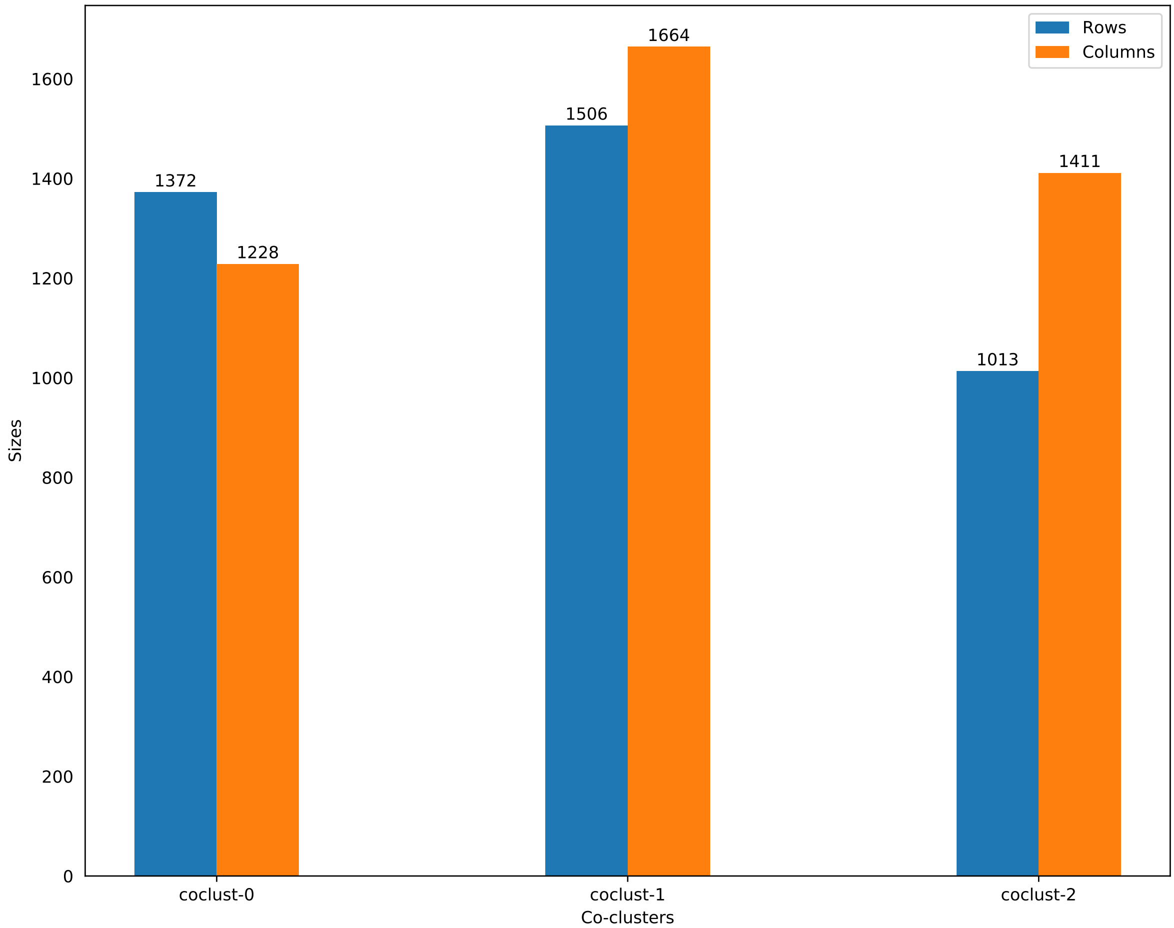
coclust.visualization.plot_cluster_sizes()

coclust.visualization.**plot_cluster_sizes**(*model*)
Plot the sizes of the clusters.

Parameters model (coclust.coclustering.BaseDiagonalCoclust) – a co-clustering model

Example

```
>>> plot_cluster_sizes(model)
```



coclust.visualization.plot_reorganized_matrix()

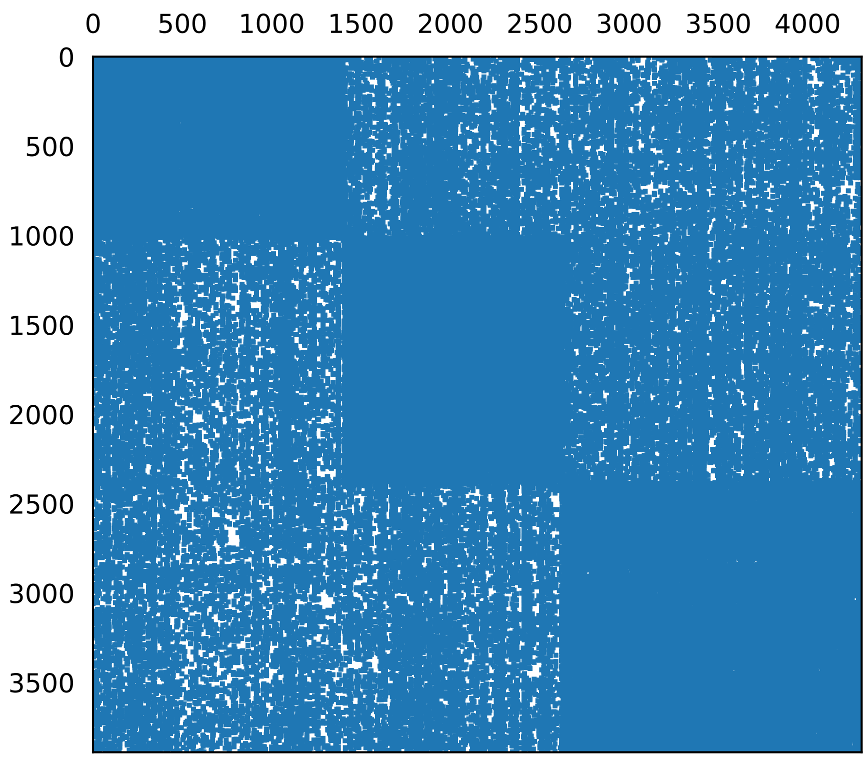
coclust.visualization.**plot_reorganized_matrix**(*X, model, precision=0.8, markersize=0.9*)
Plot the reorganized matrix.

Parameters

- **X** (*matrix*) – Data matrix
- **model** – Fitted co-clustering model
- **precision** (*float, optional*) – values greater than *precision* will be plotted
- **markersize** (*float*) – marker size

Example

```
>>> plot_reorganized_matrix(X, model)
```



coclust.visualization.plot_confusion_matrix()

coclust.visualization.**plot_confusion_matrix**(*cm*, *colormap=<matplotlib.colors.ListedColormap object>*, *labels='012'*)

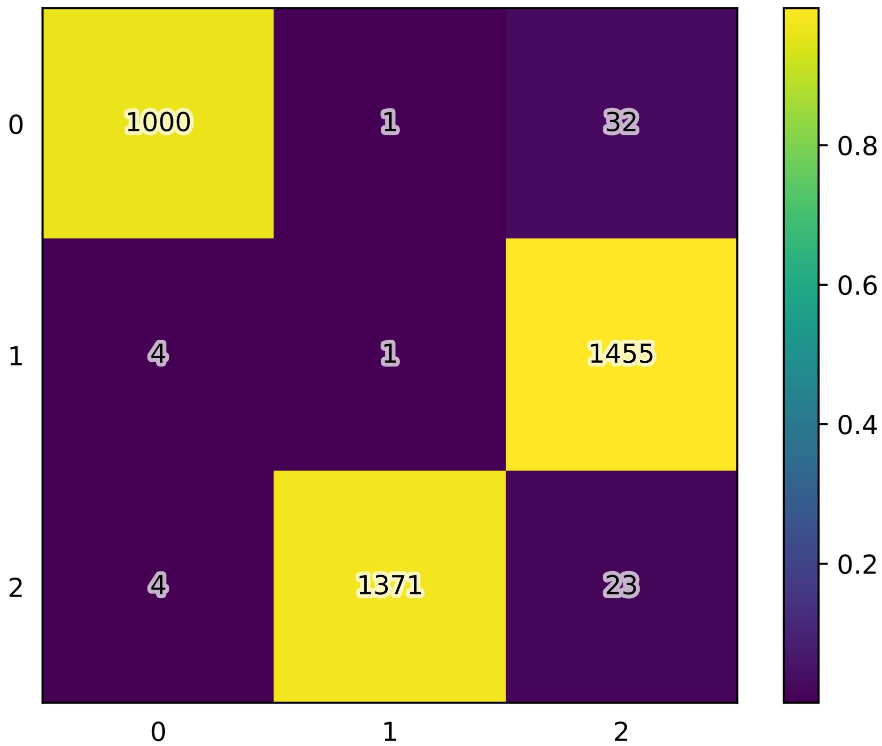
Plot a confusion matrix.

Parameters

- **cm** (*array*) – Confusion matrix
- **colormap** (matplotlib.colors.Colormap) – Color map
- **labels** – Labels

Example

```
>>> plot_confusion_matrix(cm)
```



coclust.visualization.plot_convergence(…)	Plot the convergence of a given criteria.

coclust.visualization.plot_convergence()

coclust.visualization.**plot_convergence**(*criteria*, *criterion_name*, *marker='o'*)

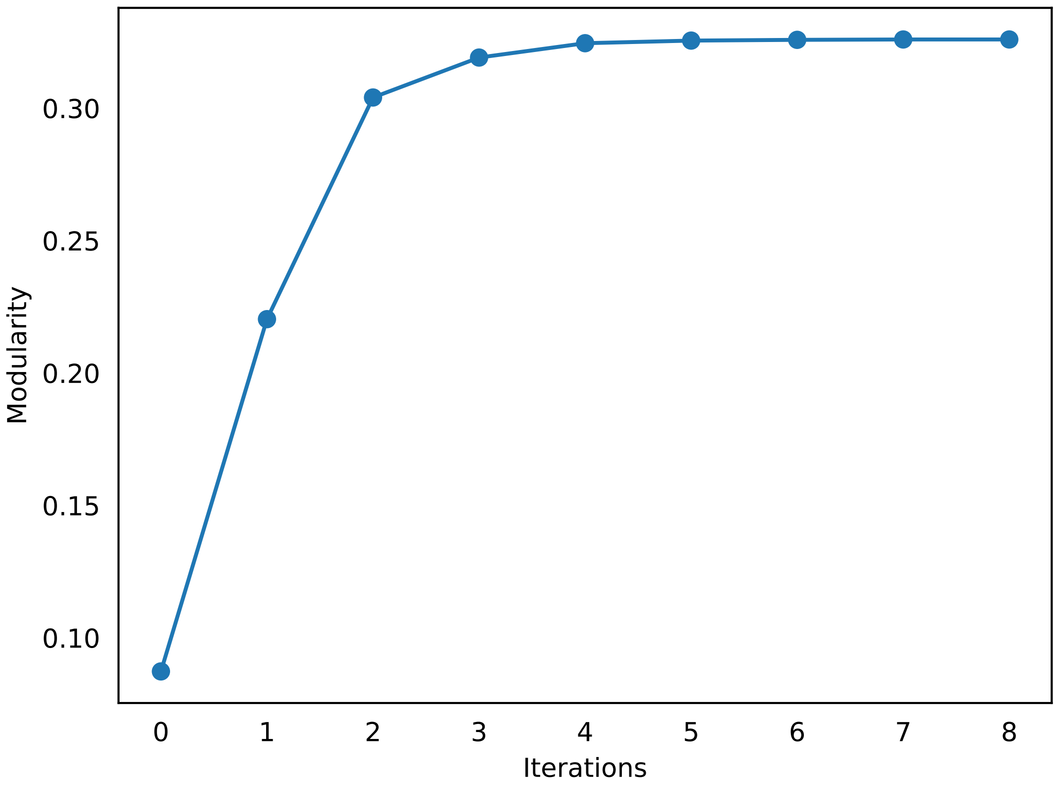
Plot the convergence of a given criteria.

Parameters

- **criteria** (*array-like*) – Criteria values
- **criterion_name** (*str*) – Name of the criteria
- **marker** – Marker

Example

```
>>> plot_convergence(modularities, "Modularity")
```



3.5.2 Model specific functions

coclust.visualization.plot_max_modularities(...)	Plot all max modularities obtained after a series of evaluations.
coclust.visualization.plot_intermediate_modularities(model)	Plot all intermediate modularities for a model.
coclust.visualization.plot_delta_kl(model[, ...])	Plot the delta values of the Information-Theoretic Co-clustering.

coclust.visualization.plot_max_modularities()

coclust.visualization.**plot_max_modularities**(*max_modularities*, *range_n_clusters*)

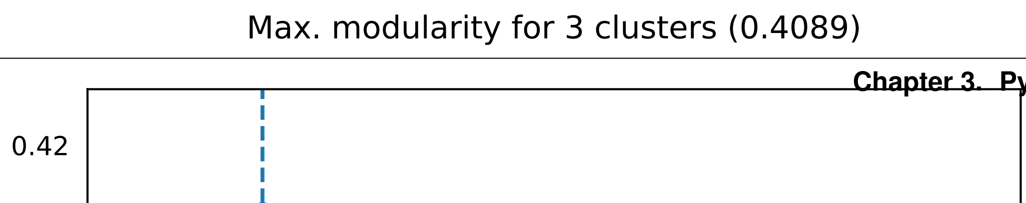
Plot all max modularities obtained after a series of evaluations. The best partition is indicated in the graph and main title.

Parameters

- **max_modularities** (*list of float*) – Final modularities for all evaluated partitions
- **range_n_clusters** (*list*) – Number of clusters for which the algorithm is to be executed

Example

```
>>> plot_max_modularities(all_max_modularities, range_n_clusters)
```



coclust.visualization.plot_intermediate_modularities()

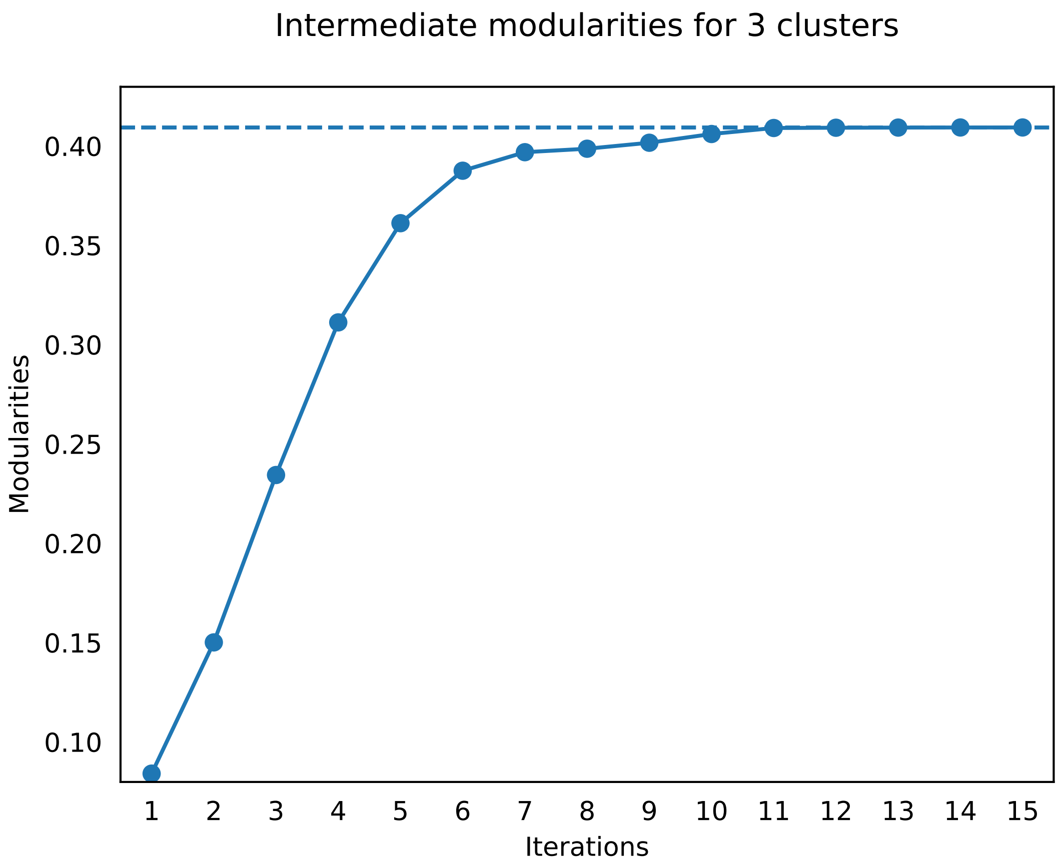
`coclust.visualization.`**`plot_intermediate_modularities`**(*model*)

Plot all intermediate modularities for a model.

Parameters **`model`** (*`coclust.coclustering.CoclustMod`*) – Fitted model

Example

```
>>> plot_intermediate_modularities(model)
```



coclust.visualization.plot_delta_kl()

`coclust.visualization.`**`plot_delta_kl`**(*model, colormap=<matplotlib.colors.ListedColormap object>, labels='012'*)

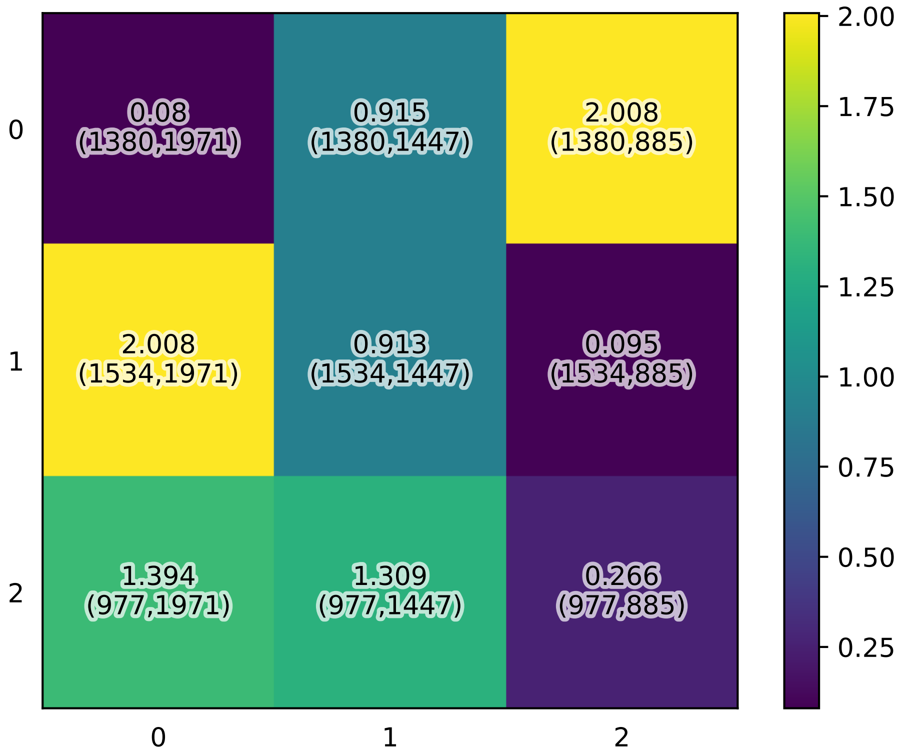
Plot the delta values of the Information-Theoretic Co-clustering.

Parameters

- **`model`** (`coclust.coclustering.CoClustInfo`) – The fitted co-clustering model
- **`colormap`** (`matplotlib.colors.Colormap`) – Color map
- **`labels`** – Labels

Example

```
>>> plot_delta_kl(model)
```



3.6 Initialization

The *coclust.initialization* module provides functions to initialize clustering or co-clustering algorithms.

`coclust.initialization.`**`random_init`**(*n_clusters*, *n_cols*, *random_state=None*)

Create a random column cluster assignment matrix.

Each row contains 1 in the column corresponding to the cluster where the processed data matrix column belongs, 0 elsewhere.

Parameters

- **n_clusters** (*int*) – Number of clusters
- **n_cols** (*int*) – Number of columns of the data matrix (i.e. number of rows of the matrix returned by this function)
- **random_state** (int or `numpy.RandomState`, optional) – The generator used to initialize the cluster labels. Defaults to the global numpy random number generator.

Returns Matrix of shape (`n_cols`, `n_clusters`)

Return type matrix

coclust.initialization.**random_init_clustering**(*n_clusters*, *n_rows*, *random_state=None*)

Create a random row cluster assignment matrix.

Each row contains 1 in the column corresponding to the cluster where the processed data matrix row belongs, 0 elsewhere.

Parameters

- **n_clusters** (*int*) – Number of clusters
- **n_rows** (*int*) – Number of rows of the data matrix (i.e. also the number of rows of the matrix returned by this function)
- **random_state** (int or numpy.RandomState, optional) – The generator used to initialize the cluster labels. Defaults to the global numpy random number generator.

Returns Matrix of shape (n_rows, n_clusters)

Return type matrix

CHAPTER 4

Scripts

The input matrix can be a Matlab file or a text file. For the Matlab file, the key corresponding to the matrix must be given. For the text file, each line should describe an entry of a matrix with three columns: the row index, the column index and the value. The separator is given by a script parameter.

4.1 Perform co-clustering: the *coclust* script

The *coclust* script can be used to run a particular co-clustering algorithm on a data matrix. The user has to select an algorithm which is given as a first argument to *coclust*. The choices are:

- modularity
- specmodularity
- info

The following command line shows how to run the *CoclustMod* algorithm three times on a matrix contained in a Matlab file whose matrix key is the string 'fea'. The computed row labels are to be stored in a file called cstr-rows.txt:

```
coclust modularity  -k fea --n_coclusters 4 --output_row_labels cstr-rows.txt  --n_
↪runs 3 cstr.mat
```

To have a list of all possible parameters for a given algorithm use the -h option as in the following example:

```
coclust modularity -h
```

```
usage: coclust [-h] {modularity,specmodularity,info} ...
```

4.1.1 Positional Arguments

subparser_name Possible choices: modularity, specmodularity, info

choose the algorithm to use

4.1.2 Sub-commands:

modularity

use the modularity based algorithm

```
coclust modularity [-h] [-k MATLAB_MATRIX_KEY | -sep CSV_SEP]
                   [--output_row_labels OUTPUT_ROW_LABELS]
                   [--output_column_labels OUTPUT_COLUMN_LABELS]
                   [--output_fuzzy_row_labels OUTPUT_FUZZY_ROW_LABELS]
                   [--output_fuzzy_column_labels OUTPUT_FUZZY_COLUMN_LABELS]
                   [--convergence_plot CONVERGENCE_PLOT]
                   [--reorganized_matrix REORGANIZED_MATRIX] [-n N_COCLUSTERS]
                   [-m MAX_ITER] [-e EPSILON]
                   [-i INIT_ROW_LABELS | --n_runs N_RUNS] [--seed SEED]
                   [-l TRUE_ROW_LABELS] [--visu]
                   INPUT_MATRIX
```

input

INPUT_MATRIX matrix file path

-k, --matlab_matrix_key if not set, csv input is considered

-sep, --csv_sep if not set, "," is considered; use "t" for tab-separated values

Default: ","

output

--output_row_labels file path for the predicted row labels

--output_column_labels file path for the predicted column labels

--output_fuzzy_row_labels file path for the predicted fuzzy row labels

Default: 2

--output_fuzzy_column_labels file path for the predicted fuzzy column labels

Default: 2

--convergence_plot file path for the convergence plot

--reorganized_matrix file path for the reorganized matrix

algorithm parameters

-n, --n_coclusters number of co-clusters

Default: 2

-m, --max_iter maximum number of iterations

Default: 15

-e, --epsilon stop if the criterion (modularity) variation in an iteration is less than EPSILON

Default: 1e-09

-i, --init_row_labels file containing the initial row labels, if not set random initialization is performed

--n_runs number of runs

Default: 1

--seed set the random state, useful for reproductible results

evaluation parameters

-l, --true_row_labels file containing the true row labels

--visu Plot modularity values and reorganized matrix (requires Numpy, SciPy and matplotlib).

Default: False

specmodularity

use the spectral modularity based algorithm

```
coclust specmodularity [-h] [-k MATLAB_MATRIX_KEY | -sep CSV_SEP]
                       [--output_row_labels OUTPUT_ROW_LABELS]
                       [--output_column_labels OUTPUT_COLUMN_LABELS]
                       [--reorganized_matrix REORGANIZED_MATRIX]
                       [-n N_COCLUSTERS] [-m MAX_ITER] [-e EPSILON]
                       [--n_runs N_RUNS] [--seed SEED] [-l TRUE_ROW_LABELS]
                       [--visu]
                       INPUT_MATRIX
```

input

INPUT_MATRIX matrix file path

-k, --matlab_matrix_key if not set, csv input is considered

-sep, --csv_sep if not set, "," is considered; use "t" for tab-separated values

Default: ","

output

--output_row_labels file path for the predicted row labels

--output_column_labels file path for the predicted column labels

--reorganized_matrix file path for the reorganized matrix

algorithm parameters

-n, --n_coclusters number of co-clusters

Default: 2

-m, --max_iter maximum number of iterations

Default: 15

-e, --epsilon stop if the criterion (modularity) variation in an iteration is less than EPSILON

Default: 1e-09

--n_runs number of runs

Default: 1

--seed set the random state, useful for reproductible results

evaluation parameters

-l, --true_row_labels file containing the true row labels

--visu Plot modularity values and reorganized matrix (requires Numpy, SciPy and matplotlib).

Default: False

info

Undocumented

```
coclust info [-h] [-k MATLAB_MATRIX_KEY | -sep CSV_SEP]
             [--output_row_labels OUTPUT_ROW_LABELS]
             [--output_column_labels OUTPUT_COLUMN_LABELS]
             [--reorganized_matrix REORGANIZED_MATRIX] [-K N_ROW_CLUSTERS]
             [-L N_COL_CLUSTERS] [-m MAX_ITER] [-e EPSILON]
             [-i INIT_ROW_LABELS | --n_runs N_RUNS] [--seed SEED]
             [-l TRUE_ROW_LABELS] [--visu]
             INPUT_MATRIX
```

input

INPUT_MATRIX matrix file path

-k, --matlab_matrix_key if not set, csv input is considered

-sep, --csv_sep if not set, "," is considered; use "t" for tab-separated values

Default: ","

output

--output_row_labels file path for the predicted row labels

--output_column_labels file path for the predicted column labels

--reorganized_matrix file path for the reorganized matrix

algorithm parameters

-K, --n_row_clusters number of row clusters

Default: 2

-L, --n_col_clusters number of column clusters

Default: 2

-m, --max_iter maximum number of iterations

Default: 15

-e, --epsilon stop if the criterion (modularity) variation in an iteration is less than EPSILON

Default: 1e-09

-i, --init_row_labels file containing the initial row labels, if not set random initialization is performed

--n_runs number of runs

Default: 1

--seed set the random state, useful for reproductible results

evaluation parameters

-l, --true_row_labels file containing the true row labels

--visu Plot modularity values and reorganized matrix (requires Numpy, SciPy and matplotlib).

Default: False

4.2 Detect the best number of co-clusters: the *coclust-nb* script

coclust-nb detects the number of co-clusters giving the best modularity score. It therefore relies on the CoclustMod algorithm. This is a simple yet often effective way to determine the appropriate number of co-clusters. A sample usage sample is given below:

```
coclust-nb cstr.csv --seed=1 --n_runs=20 --max_iter=60  --from 2 --to 6
```

```
usage: coclust-nb [-h] [-k MATLAB_MATRIX_KEY | -sep CSV_SEP]
                  [--output_row_labels OUTPUT_ROW_LABELS]
                  [--output_column_labels OUTPUT_COLUMN_LABELS]
                  [--reorganized_matrix REORGANIZED_MATRIX] [--from FROM]
                  [--to TO] [-m MAX_ITER] [-e EPSILON] [--n_runs N_RUNS]
                  [--seed SEED] [--visu]
                  INPUT_MATRIX
```

4.2.1 input

INPUT_MATRIX matrix file path

-k, --matlab_matrix_key if not set, csv input is considered

-sep, --csv_sep if not set, "," is considered; use "t" for tab-separated values

Default: ","

4.2.2 output

--output_row_labels file path for the predicted row labels

--output_column_labels file path for the predicted column labels

--reorganized_matrix file path for the reorganized matrix

4.2.3 algorithm parameters

--from minimum number of co-clusters

Default: 2

--to maximum number of co-clusters

Default: 10

-m, --max_iter maximum number of iterations

Default: 15

-e, --epsilon stop if the criterion (modularity) variation in an iteration is less than EPSILON

Default: 1e-09

--n_runs number of runs

Default: 1

--seed set the random state, useful for reproductible results

4.2.4 evaluation parameters

--visu Plot modularity values and reorganized matrix (requires Numpy, SciPy and matplotlib).

Default: False

LIPADE

Laboratoire d'Informatique PAris DEscartes

Python Module Index

c

Index

A

B

C

D

F

G

I

L

M

P

R

S