CoClust: A Python Package for Co-clustering

François Role, Stanislas Morbieu, Mohamed Nadif

Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

CoClust: A Python Package for Co-clustering

François Role
LIPADE
Université Paris-Descartes
France

Stanislas Morbieu
LIPADE
Université Paris-Descartes
France

Mohamed Nadif
LIPADE
Université Paris-Descartes
France

Abstract

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

This paper presents the theory underlying several effective diagonal and non-diagonal co-clustering algorithms, and describes **CoClust**, a package which provides implementations for these algorithms. The quality of the results produced by the implemented algorithm is demonstrated through extensive tests performed on datasets of various size and balance. **CoClust** has been designed to complete and easily interface with popular Python Machine Learning libraries such as **scikit-learn**.

Keywords: Data Mining, Co-clustering, Python.

1. Introduction

In the era of data science, clustering various kinds of objects (documents, genes, customers) has become a key activity and many high quality packaged implementations are provided for this purpose by many popular packages such as **stats** (R Core Team 2013), **skmeans** (Hornik, Feinerer, Kober, and Buchta 2012), **kernlab** (Karatzoglou, Smola, Hornik, and Zeileis 2004), **NbClust** (Charrad, Ghazzali, Boiteau, and Niknafs 2014), **Cluto** (Karypis 2003), **scikit-learn** (Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg, Vanderplas, Passos, Cournapeau, Brucher, Perrot, and Duchesnay 2011), **SciPy** (`scipy.cluster` module) (Jones, Oliphant, and Peterson 2001–), **nltk** (`nltk.cluster` module) (Bird, Klein, and Loper 2009), **Weka** (Hall, Frank, Holmes, Pfahringer, Reutemann, and Witten 2009), etc. A natural extension of standard cluster analysis is co-clustering where objects and features are simultaneously grouped into meaningful blocks called co-clusters or biclusters, thus making large data sets easier to handle and interpret. In fact, since the

seminal work of Hartigan (1972), co-clustering has found applications in many areas such as bio-informatics (Cheng and Church 2000; Madeira and Oliveira 2004; Tanay, Sharan, and Shamir 2005; Cho and Dhillon 2008; Gupta and Aggarwal 2010; Hanczar and Nadif 2011, 2012), web mining (Xu, Zong, Dolog, and Zhang 2010; Charrad, Lechevallier, Ahmed, and Saporta 2009; George and Merugu 2005; Deodhar and Ghosh 2010) and text mining (Dhillon 2001; Dhillon, Mallela, and Modha 2003) and various co-clustering algorithms have been proposed over the years (recent surveys can be found in (Freitas, Ayadi, Elloumi, Oliveira, and Hao 2012; Eren, Deveci, Küçüktunç, and Çatalyürek 2013; Henriques, Antunes, and Madeira 2015)).

While quite a large number of implementations of co-clustering algorithms[1] have been developed for gene expression data, such as **biclust** (Kaiser and Leisch 2008), **bicat** (Barkow, Bleuler, Prelić, Zimmermann, and Zitzler 2006) and **bibench** (Eren *et al.* 2013), in contrast, not so many implementations are available for co-clustering co-occurrence matrices such, for example, as document-term matrices used in text mining applications. The **CoClust** package presented in this paper therefore provides implementations of algorithms designed to efficiently handle such matrices. Depending on the method used, algorithms for co-clustering co-occurrence matrices can broadly be divided into several categories:

- Spectral methods: Spectral co-clustering methods treat the input data matrix as a bipartite graph between documents and words, and approximate the normalized cut of this graph using a real relaxation. Currently **scikit-learn** supports two spectral co-clustering algorithms: (1) the well-known "Spectral Co-Clustering" (Dhillon 2001) and (2) the "SpectralBiclustering" (Kluger, Basri, Chang, and Gerstein 2003) which is also available in the **biclust** R package.

- Model-based methods: In probabilistic co-clustering methods, two model-based co-clustering methods are implemented in the **blockcluster** (Iovleff and Singh Bhatia 2015) and **blockmodels** (Leger 2016) R packages. The first relies on the lalent block models (LBM), especially Gaussian, Bernoulli and Poisson LBM. The derived algorithms are of type Expectation-Maximization; for details see for instance (Govaert and Nadif 2003, 2005, 2008; Nadif and Govaert 2010). The second relies on the Stochastic Block Model and the Latent Block Model without or with covariates. Both models have been extended to valued networks with optional covariates on the edges.

- Matrix factorization based methods: Matrix factorization based methods are also used in the clustering and co-clustering fields. However while packages exist for document clustering based on non-negative matrix factorization (e.g., the **NMF** R package (Gaujoux and Seoighe 2010) which includes different NMF methods) leading to clustering (see for instance (Ding, Li, Peng, and Park 2006; Ding and Li 2007)), there is unfortunately no package on Non Negative Matrix Trifactorization Factorization for co-clustering.

- Information theoretic based methods: Information theoretic based methods are used to co-cluster two-way contingency tables. In this approach, a joint probability distribution is first derived from the two-way contingency matrix. The loss function to minimize is then the loss in mutual information between this joint probability distribution and a distribution defined on a reduced contingency table obtained by collapsing the rows and

[1] Also known as biclustering

the columns according to the partitions yielded by the co-clustering program. Notable algorithms in this area include (Dhillon *et al.* 2003; Govaert and Nadif 2013).

- Modularity-based methods: The use of bipartite graph-modularity as a criterion to co-cluster matrices has been pioneered by (Labiod and Nadif 2011) and since further investigated in (Ailem, Role, and Nadif 2015, 2016). This method allows to co-cluster binary or contingency matrices by maximizing an adapted version of the modularity measure traditionally used for networks.

Another dimension for characterizing co-clustering algorithms is to distinguish between general partitional co-clustering algorithms and those that seek to discover a diagonal structure (which is displayed to the user in the form of a block diagonal matrix). In the former case, the requested number of row clusters can be different from the requested number of column clusters while in the latter the two numbers obviously have to be the same in order to produce a diagonal structure. Diagonal algorithms assign each row and each column to exactly one co-cluster. For text mining applications, the attractiveness of this approach lies in its simplicity: each set of documents is automatically labelled by a set of terms. However, there may be occasions where one may want to associate a set of documents with several sets of terms. In this case, general, non-diagonal co-clustering methods may be more suitable.

Spectral methods and methods based on matrix factorization are fast and lend themselves to simple implementations. However, as we noted in a recent comparative study whose preliminary results can be found in (Ailem *et al.* 2015), they are clearly outperformed by the other cited methods when co-clustering document-term matrices in terms of accuracy. Also, as indicated above, co-clustering spectral methods are already available in **scikit-learn**. Probabilistic co-clustering methods deliver better accuracy, but several open-source implementations already exist in the form of the above-mentioned **blockcluster** and **blockmodels** R. [2] The **CoClust** package presented in this paper therefore provides **scikit-learn**-compatible implementations of two modularity-based methods and one information theoretic based methods. It is freely available at `https://pypi.python.org/pypi/coclust`.

The first two algorithms, CoclustMod (Ailem *et al.* 2015, 2016) and CoclustSpecMod (Labiod and Nadif 2011), are recent representatives of the family of block-diagonal co-clustering algorithms. These algorithms have several advantages. First, being block-diagonal algorithms, they directly produce interpretable descriptions of the resulting document clusters since each document cluster is directly associated with one term cluster. Second, the experimental results described in (Ailem *et al.* 2015) have shown that these algorithms can adapt to various kinds of matrices (whether they are binary, contingency, weighted or unweighted matrices). In addition to this flexibility, they outperform popular, older block-diagonal algorithms such as the above-mentioned well-known "Spectral Co-Clustering" algorithm.

The third algorithm (CoclustInfo) is based on an information-theoretic approach. Information-theoretic co-clustering has become very popular in the text mining community since the above-mentioned work by (Dhillon *et al.* 2003). Its main benefits are speed of convergence and scalability. CoclustInfo is an implementation of the CROINFO algorithm described in (Govaert and Nadif 2013, 2016).

[2] It should however be noted as a word of warning that **blockcluster** is known to be a bit flawed and does not scale well, which could motivate us to integrate alternative implementations of probabilistic co-clustering algorithms into the **CoClust** package, as indicated in the conclusion.

Last but not least, the three algorithms have the advantage of simplicity since they can be implemented using simple alternated iterative procedures where one of the partitions is fixed (say, the column partition) and a better partition of the other set (say, the row partition) is searched.

The outline of the paper is as follows. We first review the theory underlying these algorithms (Section 2) before presenting the interface and API of the module that implements them (Section 3). We conclude with a benchmark that demonstrates the effectiveness of the software compared to other algorithms.

2. Theory

As said in the introduction, co-clustering algorithms simultaneously cluster rows and columns into partitions and a pair consisting of a row cluster and a column cluster determines a co-cluster, which is a submatrix of the original data matrix. Diagonal algorithms assign each row and each column to exactly one co-cluster and so the rows and columns can be rearranged so that the co-clusters form a kind of diagonal (see for example Figure 1). Non-diagonal algorithms, on the other hand, do not have this restriction and rearranging the rows and columns according to the found partition may result into structures such as the one shown in Figure 2.

The **CoClust** package provides three co-clustering algorithms (two diagonal and one non-diagonal). The first two perform co-clustering by maximizing the modularity of bipartite graphs while the third one uses the information-theoretic notion of mutual information to define its criterion. Before describing these notions, we first give some general notations we will use throughout the paper.

2.1. General notations

We will consider the partition of the sets I of n objects and the set J of d attributes into g non overlapping clusters, where g may be greater or equal to 2. Let us define a $n \times g$ indicator matrix $\mathbf{z} = (z_{ik})$ and a $d \times g$ indicator matrix $\mathbf{w} = (w_{jk})$. The kth row cluster is defined by the set of rows i such that $z_{ik} = 1$. In the same manner, the kth column cluster is defined by the set of rows j such that $w_{jk} = 1$. Finally, X is the matrix used as input to all the methods described in this paper; X can be of any kind provided it is a matrix with non-negative entries (e.g., a graph adjacency matrix, or a document-term matrix, depending on the application domain).

2.2. Modularity-based, block-diagonal co-clustering

The first family of algorithms implemented in the **CoClust** package consists of two algorithms (CoClustMod and CoclustSpecMod) that seek an optimal block diagonal clustering, meaning that objects and features have the same number of clusters and that, after proper permutation of the rows and columns, the algorithm produces as result a block diagonal matrix (see Figure 1). In the context of document-term matrices, this co-clustering model has the advantage of directly producing interpretable descriptions of the resulting document clusters.

A notable block-diagonal co-clustering algorithm is the bipartite spectral graph partitioning

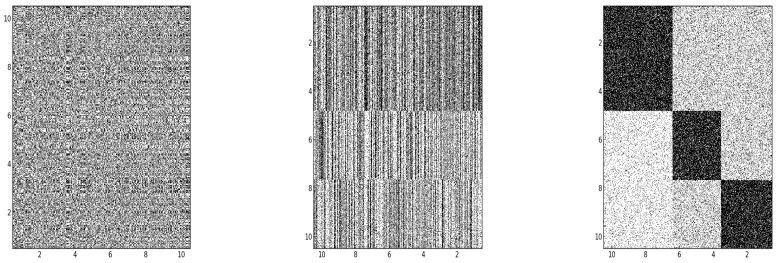

Figure 1: Left: Original data - Middle: data reorganized according to row clusters - Right: data reorganized according to row and column clusters.

algorithm described in (Dhillon 2001). Inspired by previous work on spectral graph clustering, this algorithm finds the optimal minimum cut partitions in a bipartite document-term graph by computing the second left and right singular vector of the normalized document-term matrix, thus using a real relaxation of the discrete optimization problem. The block diagonal algorithms implemented in **CoClust** follow a completely different approach: they try to maximize a measure of the concentration of edges within co-clusters compared with the random distribution of edges between all nodes regardless of the co-clusters. This criterion is an adaptation to the bipartite case of the standard "graph modularity". Before describing the two algorithms, it is therefore useful to review this notion of "bipartite graph modularity".

Bipartite Graph Modularity (BGM)

In this section we first review the standard graph modularity measure, and show how to adapt it so that it can be used in the co-clustering context.

Modularity is a quality criterion often used for detecting communities in graphs, which has received considerable attention in several disciplines since the seminal work by Newman and Girvan (2004). Intuitively, modularity compares the number of edges inside a cluster of nodes with the expected number if the edges in the graph were placed at random. [3]

Given the graph $G = (V, E)$, let X be a binary, symmetric adjacency matrix with (i, i') as entry; and $x_{ii'} = 1$ if there is an edge between the nodes i and i'. If there is no edge between nodes i and i', $x_{ii'}$ is equal to zero. Finding a partition of the set of nodes V into homogeneous subsets leads to the resolution of the following integer linear program: $\max_{\mathbf{c}} Q(X, \mathbf{c})$ where $Q(X, \mathbf{c})$ is the modularity measure:

$$Q(X, \mathbf{c}) = \frac{1}{2|E|} \sum_{i,i'=1}^{n} (x_{ii'} - \frac{x_{i.}x_{i'.}}{2|E|})c_{ii'} \tag{1}$$

In this formula, $c_{ii'} = \sum_{k=1}^{g} z_{ik}z_{i'k}$, meaning that $c_{ii'}$ is 1 when node i and i' are in the same group and 0 otherwise. In addition, $|E|$ is the number of edges and $x_{i.} = \sum_{i'} x_{ii'}$ is the degree of i.

Let now $\boldsymbol{\delta} = (\delta_{ii'})$ be the $(n \times n)$ data matrix defined by $\forall i, i' \quad \delta_{ii'} = \frac{x_{i.}x_{i'.}}{2|E|}$. Expression 1 then becomes $Q(X, \mathbf{c}) = \frac{1}{2|E|} Trace[(X - \boldsymbol{\delta})\mathbf{c}]$. In summary, we seek a binary matrix $\mathbf{c}$ which

[3] The standard null model used in the literature also assumes that the nodes keep the degree they have in the original network.

is defined as $\mathbf{z}\mathbf{z}^\top$ and models a partition in a relational space, thus having the properties of an equivalence relation.

$$\begin{cases} c_{ii} = 1, \ \forall i & \text{reflexivity} \\ c_{ii'} - c_{i'i} = 0, \forall (i, i') & \text{symmetry} \\ c_{ii'} + c_{i'i''} - c_{ii''} \leq 1, \forall (i, i', i'') & \text{transitivity} \\ x_{ii'} \in \{0, 1\}, \forall (i, i') & \text{binarity} \end{cases}$$

In a bipartite context, the basic idea is to model the simultaneous row and column partitions using a relation $\mathbf{c}$ defined on $I \times J$. Noting that $\mathbf{c} = \mathbf{z}\mathbf{w}^t$ and the general term can be expressed as follows: $c_{ij} = 1$ if object i is in the same block as attribute j and $c_{ij} = 0$ otherwise. Then $c_{ij} = \sum_{k=1}^{g} z_{ik} w_{jk}$. Now, given a rectangular matrix X defined on $I \times J$, modularity can be reformulated as follows in the co-clustering context:

$$Q(X, \mathbf{c}) = \frac{1}{x_{..}} \sum_{i=1}^{n} \sum_{j=1}^{d} \sum_{k=1}^{g} (x_{ij} - \frac{x_{i.} x_{.j}}{x_{..}}) z_{ik} w_{jk}, \tag{2}$$

where $x_{..} = \sum_{i,j} x_{ij} = |E|$ is the total weight of edges and $x_{i.} = \sum_j x_{ij}$ (the degree of i in the binary case and the sum of the weights in the contingency and continuous cases) and $x_{.j} = \sum_i x_{ij}$ (the degree of j in the binary case and the sum of the weights in the contingency and continuous cases). This modularity measure can also take the following form:

$$Q(X, \mathbf{c}) = \frac{1}{x_{..}} Trace[(X - \boldsymbol{\delta})^\top \mathbf{z}\mathbf{w}^\top] = Q(X, \mathbf{z}\mathbf{w}^\top). \tag{3}$$

As the objective function 3 is linear with respect to C and as the constraints that C must respect are linear equations, the problem can theoretically be solved using an integer linear programming solver. However, this problem is NP hard, and as a result, in practice, we use heuristics for dealing with large data sets.

CoclustMod: Co-clustering by alternated maximization of BGM

In this section we describe the theory underlying CoClustMod, one of the two block-diagonal algorithms provided by the **CoClust** package (Ailem *et al.* 2015, 2016).

Proposition 1: Let X be a $(n \times d)$ matrix with non-negative entries and $\mathbf{c}$ be a $(n \times d)$ matrix defining a block seriation, the modularity measure $Q(X, \mathbf{c})$ can be rewritten as:

1.
$$Q(X, \mathbf{c}) = \frac{1}{x_{..}} Trace[(X^{\mathbf{w}} - \boldsymbol{\delta}^{\mathbf{w}})^\top \mathbf{z}] = Q(X^{\mathbf{w}}, \mathbf{z})$$
where $X^{\mathbf{w}} = (x^{\mathbf{w}}_{ik})_{\substack{1 \leq i \leq n \\ 1 \leq k \leq g}}$ and $\boldsymbol{\delta}^{\mathbf{w}} = (\delta^{\mathbf{w}}_{ik})_{\substack{1 \leq i \leq n \\ 1 \leq k \leq g}}$ respectively defined by $x^{\mathbf{w}}_{ik} = \sum_{j=1}^{d} w_{jk} x_{ij}$ and $\delta_{kj} = \frac{x_{i.} x^{\mathbf{w}}_{.k}}{x_{..}}$ where $x^{\mathbf{w}}_{.k} = \sum_{j=1}^{d} w_{jk} x_{.j}$.
2.
$$Q(X, \mathbf{c}) = \frac{1}{x_{..}} Trace[(X^{\mathbf{z}} - \boldsymbol{\delta}^{\mathbf{z}})^\top \mathbf{w}] = Q(X^{\mathbf{z}}, \mathbf{w})$$
where $X^{\mathbf{z}} = (x^{\mathbf{z}}_{kj})_{\substack{1 \leq k \leq g \\ 1 \leq j \leq d}}$ and $\boldsymbol{\delta}^{\mathbf{z}} = (\delta^{\mathbf{z}}_{kj})_{\substack{1 \leq k \leq g \\ 1 \leq j \leq d}}$ respectively defined by $x^{\mathbf{z}}_{kj} = \sum_{i=1}^{n} z_{ik} x_{ij}$ and $\delta^{\mathbf{z}}_{kj} = \frac{x^{\mathbf{z}}_{k.} x_{.j}}{x_{..}}$ where $x^{\mathbf{z}}_{k.} = \sum_{i=1}^{n} z_{ik} x_{i.}$.

Proof for Proposition 1 can be found in (Ailem *et al.* 2015). This proposition is at the heart of the CoclustMod algorithm since it allows to maximize the modularity by alternatively maximizing $Q(X^{\mathbf{w}}, \mathbf{z})$ and $Q(X^{\mathbf{z}}, \mathbf{w})$. The optimal classification binary matrices $\mathbf{z}^*$ and $\mathbf{w}^*$ are respectively defined by $\mathbf{z}^* = \arg\max_{\mathbf{z}} Trace(X^{\mathbf{w}} - \boldsymbol{\delta}^{\mathbf{w}})^\top \mathbf{z}$ and $\mathbf{w}^* = \arg\max_{\mathbf{w}} Trace(X^{\mathbf{z}} - \boldsymbol{\delta}^{\mathbf{z}})^\top \mathbf{w}$. In fact, $Q(X^{\mathbf{w}}, \mathbf{z})$ and hence modularity can be maximized by assigning row i to the cluster k maximizing $(X^{\mathbf{w}} - \delta^{\mathbf{w}})_{ik}$ since we have:

$$\begin{aligned} Q(X^{\mathbf{w}}, \mathbf{z}) &= \frac{1}{x_{..}} Trace[(X^{\mathbf{w}} - \delta^{\mathbf{w}})^\top \mathbf{z}] \\ &= \frac{1}{x_{..}} \sum_{i,k} z_{ik} (X^{\mathbf{w}} - \delta^{\mathbf{w}})_{ik} \\ &= \frac{1}{x_{..}} \sum_{i=1}^{n} \left(\sum_{k=1}^{g} z_{ik} (X^{\mathbf{w}} - \delta^{\mathbf{w}})_{ik} \right) \end{aligned}$$

The same kind of argument applies for $Q(X^{\mathbf{z}}, \mathbf{w})$, which leads to the different steps presented in Algorithm 1.

Algorithm 1 CoclustMod

Input: binary or contingency data X, number of clusters g
Output: partition matrices $\mathbf{z}$ and $\mathbf{w}$
1. Initialization of $\mathbf{w}$
repeat
2. Compute $X^{\mathbf{w}}$
3. Compute $\mathbf{z}$ maximizing $Q(X^{\mathbf{w}}, \mathbf{z})$ by $z_{ik} = 1$ if $k = \arg\max_{1 \leq \ell \leq g} \left(x^{\mathbf{w}}_{i\ell} - \frac{x_{i.} x^{\mathbf{w}}_{.\ell}}{x_{..}} \right)$ and $z_{ik} = 0$ otherwise; $\forall i = 1, \ldots, n$
4. Compute $X^{\mathbf{z}}$
5. Compute $\mathbf{w}$ maximizing $Q(X^{\mathbf{z}}, \mathbf{w})$ by $w_{jk} = 1$ if $k = \arg\max_{1 \leq \ell \leq g} \left(x^{\mathbf{z}}_{\ell j} - \frac{x^{\mathbf{z}}_{\ell .} x_{.j}}{x_{..}} \right)$ and $w_{jk} = 0$; $\forall j = 1, \ldots, d$
6. Compute $Q(X, \mathbf{z}\mathbf{w}^\top)$
until *no change* of $Q(X, \mathbf{z}\mathbf{w}^\top)$

CoClustSpecMod: Co-clustering by spectral maximization of BGM

In this section we describe the theory underlying CoClustSpecMod, another block-diagonal algorithm provided by the **CoClust** package. In the same way as CoClustMod, CoClustSpecMod sees modularity-based co-clustering as a trace maximization problem, but with two important differences as described in (Labiod and Nadif 2011). First, it uses normalized versions of the $\mathbf{z}$ and $\mathbf{w}$ matrices, and second, it maximizes modularity using a spectral approach, which contrasts with the direct maximization performed by CoClustMod.

The use of a normalized modularity matrix is motivated by the desire to balance the row and column cluster sizes. The $\mathbf{z}$ matrix is therefore replaced by a $\widetilde{\mathbf{z}} = \mathbf{z}\mathbf{h}^{-\frac{1}{2}}$ where $\mathbf{h}$ is a diagonal matrix whose each diagonal element contains the number of elements in the kth row cluster. In the same way, the $\mathbf{w}$ matrix is replaced by a $\widetilde{\mathbf{w}} = \mathbf{w}\mathbf{f}^{-\frac{1}{2}}$ where $\mathbf{f}$ is a diagonal matrix whose each diagonal element contains the number of elements in the kth column cluster. The

modularity problem then amounts to the following trace maximization problem:

$$\max_{\tilde{\mathbf{z}}^\top \tilde{\mathbf{z}}=I_g, \tilde{\mathbf{w}}^\top \tilde{\mathbf{w}}=I_g} Trace[\tilde{\mathbf{z}}^\top (X-\boldsymbol{\delta})\tilde{\mathbf{w}}], \tag{4}$$

This maximization is performed using a spectral approach by performing the following steps:

1. Scale the modularity matrix;
2. Approximate the scaled matrix using SVD;
3. Use the matrices produced by the SVD decomposition to form a new matrix, then apply a clustering algorithm (e.g., k-means) to cluster the new matrix.

Step 1 is performed as follows. Let B be a bipartite modularity matrix. A scaled version $\widetilde{B}$ of this matrix is computed as: $\widetilde{B} = D_r^{\frac{-1}{2}} B D_c^{\frac{-1}{2}}$ where $D_r = diag(A\mathbb{1})$ and $D_c = diag(A^\top \mathbb{1})$. In step 2, $\widetilde{B}$ is approximated as $\sum_{k=1}^{g-1} \tilde{U}_k \lambda_k \tilde{V}_k^\top$. where $\tilde{U}$ and $\tilde{V}$ are derived from the singular vectors as follows:

$$\tilde{U}_k = \frac{D_r^{\frac{1}{2}} U_k}{||D_r^{\frac{1}{2}} U_k||} \text{ and } \tilde{V}_k = \frac{D_c^{\frac{1}{2}} V_k}{||D_c^{\frac{1}{2}} V_k||}$$

Finally, $\tilde{U}$ and $\tilde{V}$ are used to form a matrix $Q = \left(\tilde{U}, \tilde{V}\right)^\top$ which is given as input to a clustering algorithm such as k-means. The different steps of CoClustSpecMod are presented in Algorithm 2.

Algorithm 2 CoclustSpecMod

Input: data X, number of clusters g
Output: partition matrices R and C
1. Form the affinity matrix X
2. Define D_r and D_c to be the diagonal matrices $D_r = diag(X\mathbb{1})$ and $D_c = diag(X^\top \mathbb{1})$.
3. Find U,V the $(g-1)$ left-right largest eigenvectors of $\tilde{X} = D_r^{-\frac{1}{2}} X D_c^{-\frac{1}{2}}$.
4. Form matrices $\tilde{U}$, $\tilde{V}$ and $Q = \left(\tilde{U}, \tilde{V}\right)^\top$ from U, V.
5. Cluster the rows of Q into g clusters by using k-means
6. Assign object i to cluster R_k if and only if the corresponding row of the matrix Q was assigned to cluster R_k, and assign attribute j to cluster C_k if and only if the corresponding row of the matrix Q was assigned to cluster C_k.

2.3. Information-theoretic co-clustering

In this section we describe the notions underlying the third algorithm, CoclustInfo, provided by the **CoClust** package. In contrast to the previously described algorithms, CoclustInfo takes an information-theoretic approach and uses mutual information to define its criterion (Govaert and Nadif 2013)[4]. Another important difference is that this algorithm does not seek to discover a block-diagonal structure like the previously described algorithms. The requested number of row clusters can be different from the requested number of column clusters. A representative example of the kind of matrix obtained when using CoclustInfo is shown in Figure 2).

[4] Chapter 4.

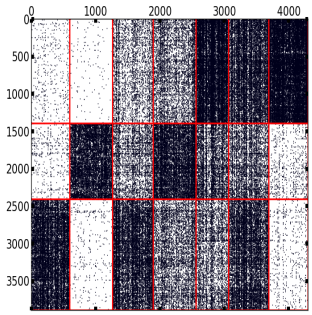

Figure 2: Typical matrix obtained when using CoclustInfo to co-cluster a dataset. This matrix is to be compared to the kind of block-diagonal matrix obtained when using one of the two previously described graph-based algorithms.

Initial contingency table and associated joint distribution

Let X be a $n \times d$ contingency table such as the example shown in Table 1 (left). This table can be associated with two categorical random variables, taking values in the sets $I = \{1, \ldots, i, \ldots, n\}$ and $J = \{1, \ldots, j, \ldots, d\}$ respectively. In summary, we have two categorical variables, one taking values in the set I of rows and the other in the set J of columns.

Let now $P_{IJ} = (p_{ij})$ denote the sample joint probability distribution associated with the two variables. It can be represented by a $n \times d$ matrix defined by $p_{ij} = \frac{p_{ij}}{N}$ with $N = x_{..}$. An example of the probability matrix corresponding to our sample contingency matrix is shown in Table 1 (right). As for other random variables, the association between the two

	1	2	3	4	5	
1	5	4	6	1	0	16
2	6	5	4	0	1	16
3	1	0	1	7	5	14
4	1	1	0	6	5	13
5	4	5	3	4	5	21
6	5	4	4	3	4	20
	22	19	18	21	20	100

	1	2	3	4	5	
1	0.05	0.04	0.06	0.01	0.00	0.16
2	0.06	0.05	0.04	0.00	0.01	0.16
3	0.01	0.00	0.01	0.07	0.05	0.14
4	0.01	0.01	0.00	0.06	0.05	0.13
5	0.04	0.05	0.03	0.04	0.05	0.21
6	0.05	0.04	0.04	0.03	0.04	0.20
	0.22	0.19	0.18	0.21	0.20	1.00

Table 1: Example of contingency table and associated joint distribution.

categorical variables I and J can be measured using mutual information. Intuitively, mutual information between two variables compares the observed frequencies in the data with the expected frequencies under the null hypothesis of no association. The mutual information between two variables I and J is expressed as $\mathcal{I}(P_{IJ}) = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{p_{i.}p_{.j}}$.

Aggregated contingency table and associated joint distribution

In this section, we describe the new contingency table and associated joint distributions that can be derived when simultaneously aggregating the rows and the columns of a contingency table X according to a couple of partitions of the sets I and J. In fact, if $\mathbf{z}$ and $\mathbf{w}$ are partitions in g clusters and m clusters of the set I of the rows and the set J of columns of X, then a new two-way contingency table $X^{\mathbf{zw}} = (x_{k\ell}^{\mathbf{zw}})$ can be associated with two categorical random variables taking values in sets $K = \{1, \ldots, g\}$ and $L = \{1, \ldots, m\}$ by merging the

rows and columns according to the partitions $\mathbf{z}$ and $\mathbf{w}$:

$$x_{k\ell}^{\mathbf{zw}} = \sum_{i,j} z_{ik} w_{j\ell} x_{ij} \qquad \forall k \in K \quad \text{and} \quad \forall \ell \in L.$$

The distribution that can be associated to $\mathbf{z}$ and $\mathbf{w}$ is the distribution $P_{KL}^{\mathbf{zw}} = (p_{k\ell}^{\mathbf{zw}})$ defined on $K \times L$ by $p_{k\ell}^{\mathbf{zw}} = \frac{x_{k\ell}^{\mathbf{zw}}}{N} = \sum_{i,j} z_{ik} w_{j\ell}\, p_{ij} \quad \forall (k,\ell) \in K \times L$. The following equation

$$\sum_{k,\ell} p_{k\ell}^{\mathbf{zw}} = \sum_{i,j,k,\ell} z_{ik} w_{j\ell} p_{ij} = \sum_{i,j} p_{ij} \sum_{k,\ell} z_{ik} w_{j\ell} = 1 \text{ since } \sum_{k,\ell} z_{ik} w_{j\ell} = 1$$

shows that $P_{KL}^{\mathbf{zw}}$ is a distribution. Moreover it can be noticed that, as

$$\sum_{\ell} p_{k\ell}^{\mathbf{zw}} = \sum_{i,j,\ell} z_{ik} w_{j\ell}\, p_{ij} = \sum_{i} (z_{ik} \sum_{j} (p_{jj} \sum_{\ell} w_{j\ell})) = \sum_{i} z_{ik} p_{i.} \text{ since } \sum_{\ell} w_{j\ell} = 1$$

the row margins of this new distribution do not depend on the partition $\mathbf{w}$ and will be denoted $p_{k.}^{\mathbf{z}}$. Similarly, the column margins $\sum_k p_{k\ell}^{\mathbf{zw}}$ are equal to $\sum_j w_{j\ell} p_{.j}$ and will be denoted $p_{.\ell}^{\mathbf{w}}$. For instance, the aggregation of the rows and columns of the data according the partitions $\mathbf{z} = (1,1,2,2,3,3)$ and $\mathbf{w} = (1,1,1,2,2)$ leads to the contingency table $X^{\mathbf{zw}}$ and the distribution $P_{KL}^{\mathbf{zw}}$ reported in Table 2.

	1	2	
1	30.0	2.00	32.0
2	4.00	23.0	27.0
3	25.0	16.0	41.0
	59.0	41.0	100

	1	2	
1	0.30	0.02	0.32
2	0.04	0.23	0.27
3	0.25	0.16	0.41
	0.59	0.41	1.00

Table 2: Aggregated contingency table $X^{\mathbf{zw}}$(left) and associated distribution $P_{KL}^{\mathbf{zw}}$ (right).

Table 2 gives the original distribution P_{IJ} and the distribution $P_{KL}^{\mathbf{zw}}$ obtained after aggregating the rows and columns. As can be seen in this example, the two distributions are similar. Using the mutual information applied on $P_{KL}^{\mathbf{zw}}$ distribution, we obtain the following measure:

$$\mathcal{I}(P_{KL}^{\mathbf{zw}}) = \sum_{k,\ell} p_{k\ell}^{\mathbf{zw}} \log \frac{p_{k\ell}^{\mathbf{zw}}}{p_{k.}^{\mathbf{z}} p_{.\ell}^{\mathbf{w}}}.$$

One can then express the loss in mutual information incurred when passing from the original probability matrix to the aggregated matrix as:

$$\mathcal{I}(P_{IJ}) - \mathcal{I}(P_{KL}^{\mathbf{zw}}) = \mathrm{KL}(P_{IJ} || P_{KL}^{\mathbf{zw}})$$

where $\mathrm{KL}(P_{IJ}||Q_{KL}^{\mathbf{zw}}) = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{p_{ij}^{\mathbf{zw}}}$ is the Kullback-Leibler distance between the two distributions P_{IJ} and $P_{KL}^{\mathbf{zw}}$. This difference in mutual information is exactly the criterion minimized by the CoclustInfo algorithm. It has been shown (Govaert and Nadif 2013) that this loss in mutual information can equivalently be expressed as:

$$\mathcal{I}(P_{IJ}) - \mathcal{I}(R_{IJ}^{\mathbf{zw}\boldsymbol{\gamma}}) = \mathrm{KL}(P_{IJ} || R_{IJ}^{\mathbf{zw}\boldsymbol{\gamma}})$$

where $R_{IJ}^{\mathbf{zw}\boldsymbol{\gamma}} = (r_{ij}^{\mathbf{zw}\boldsymbol{\gamma}})$ is a distribution depending on the partitions $\mathbf{z}$ and $\mathbf{w}$ and a parameter $\boldsymbol{\gamma}$. $R_{IJ}^{\mathbf{zw}\boldsymbol{\gamma}}$ can be defined by $r_{ij}^{\mathbf{zw}\boldsymbol{\gamma}} = p_{i.} p_{.j} \sum_{k,\ell} z_{ik} w_{j\ell} \gamma_{k\ell}$. The parameter $\boldsymbol{\gamma} = (\gamma_{k\ell})$ corresponds to

a matrix of size (g, m) where each $\gamma_{k\ell}$ plays the role of the centroid of the co-cluster $k\ell$ and such that $\gamma_{k\ell} > 0 \quad \forall k, \ell \quad$ and $\quad \sum_{k,\ell} p^{\mathbf{z}}_{k.} p^{\mathbf{w}}_{.\ell} \gamma_{k\ell} = 1$. It can be shown that $R^{\mathbf{zw}\boldsymbol{\gamma}}_{IJ}$ is a distribution, which in addition has the same column and row margins that the P_{IJ} distribution.

Before seeing how the $\widetilde{\mathrm{W}}_{\mathcal{I}}(\mathbf{z}, \mathbf{w}, \boldsymbol{\gamma}) = \mathcal{I}(P_{IJ}) - \mathcal{I}(R^{\mathbf{zw}\boldsymbol{\gamma}}_{IJ})$ criterion can be optimized in practice, it is worth noting here that it is a generalization of the criterion proposed for the well-known ITCC algorithm (Dhillon *et al.* 2003).[5]

The minimization of the criterion $\widetilde{\mathrm{W}}_{\mathcal{I}}(\mathbf{z}, \mathbf{w}, \boldsymbol{\gamma})$ can be obtained by alternating the three computations: $\mathbf{z} = \arg\min_{\mathbf{z}} \widetilde{\mathrm{W}}_{\mathcal{I}}(\mathbf{z}, \mathbf{w}, \boldsymbol{\gamma})$, $\mathbf{w} = \arg\min_{\mathbf{w}} \widetilde{\mathrm{W}}_{\mathcal{I}}(\mathbf{z}, \mathbf{w}, \boldsymbol{\gamma})$ and $\boldsymbol{\gamma} = \arg\min_{\boldsymbol{\gamma}} \widetilde{\mathrm{W}}_{\mathcal{I}}(\mathbf{z}, \mathbf{w}, \boldsymbol{\gamma})$. More precisely, it has been shown in (Govaert and Nadif 2013) that the minimization of $\widetilde{\mathrm{W}}_{\mathcal{I}}(\mathbf{z}, \mathbf{w}, \boldsymbol{\gamma})$ for fixed $\mathbf{w}$ and $\boldsymbol{\gamma}$ is obtained by assigning each row i to the cluster k maximizing $\sum_{\ell} p^{\mathbf{w}}_{i\ell} \log \gamma_{k\ell}$. Similarly, in the computation of $\mathbf{w}$, the minimization of $\widetilde{\mathrm{W}}_{\mathcal{I}}(\mathbf{z}, \mathbf{w}, \boldsymbol{\gamma})$ for fixed $\mathbf{z}$ and $\boldsymbol{\gamma}$ is obtained by assigning each column j to the cluster ℓ maximizing $\sum_{k} p^{\mathbf{z}}_{kj} \log \gamma_{k\ell}$. Finally, for the computations of $\boldsymbol{\gamma}$, the problem can be formulated as $\arg\max_{\boldsymbol{\gamma}} \sum_{k,\ell} p^{\mathbf{zw}}_{k\ell} \log \gamma_{k\ell}$ with $\sum_{k,\ell} p^{\mathbf{z}}_{k.} p^{\mathbf{w}}_{.\ell} \gamma_{k\ell} = 1$ which yields to $\gamma_{k\ell} = \frac{p^{\mathbf{zw}}_{k\ell}}{p^{\mathbf{z}}_{k.} p^{\mathbf{w}}_{.\ell}}$ for all k, ℓ. These different steps are summarized in the pseudo-code shown in Algorithm 3.

Algorithm 3 CoclustInfo

input: X, g, m
initialization: $\mathbf{z}$, $\mathbf{w}$, $\gamma_{k\ell} = \frac{p^{\mathbf{zw}}_{k\ell}}{p^{\mathbf{z}}_{k.} p^{\mathbf{w}}_{.\ell}}$
repeat
  repeat
    step 1. $z_{ik} = 1$ if $k = \arg\max_{1 \leq k' \leq g} \sum_{\ell} p^{\mathbf{w}}_{i\ell} \log \gamma_{k'\ell}$ and $z_{ik} = 0$ otherwise $\forall i$
    step 2. $\gamma_{k\ell} = \frac{p^{\mathbf{zw}}_{k\ell}}{p^{\mathbf{z}}_{k.} p^{\mathbf{w}}_{.\ell}}$
  until convergence
  repeat
    step 3. $w_{j\ell} = 1$ if $\ell = \arg\max_{1 \leq \ell' \leq m} \sum_{k} p^{\mathbf{z}}_{kj} \log \gamma_{k\ell'}$ and $w_{j\ell} = 0$ otherwise $\forall j$
    step 4. $\gamma_{k\ell} = \frac{p^{\mathbf{zw}}_{k\ell}}{p^{\mathbf{z}}_{k.} p^{\mathbf{w}}_{.\ell}}$
  until convergence
until convergence
return $\mathbf{z}$ and $\mathbf{w}$

3. Software

The **CoClust** package provides a set of convenience command-line tools enabling to launch a co-clustering task on a dataset by only providing the suitable parameters. In addition, for Python developers, it also exposes an API designed to provide a seamless integration with the **scikit-learn** library.

3.1. Scripts

The two scripts included in the **CoClust** package are:

[5] This generalization is possible thanks to the introduction of the $\boldsymbol{\gamma}$ parameter. See (Govaert and Nadif 2013) for more details on this point.

- `coclust` which runs a co-clustering algorithm;
- `coclust-nb` which provides recommendations to select the best number of co-clusters.

Running a co-clustering algorithm: the "coclust" script

The `coclust` script can be used to run an algorithm on a data matrix. It also provides parameters for the evaluation of the results.

The user has to select an algorithm which is given as a first argument to `coclust`. The choices are:

- `modularity`;
- `specmodularity`;
- `info`.

These choices correspond to the CoclustMod, CoclustSpecMod, and CoclustInfo algorithms respectively.

The other options that have to be given depend on the algorithm. Some of them are however common to all of them:

- those describing the input matrix;
- those used for the evaluation;
- some of the output and algorithm parameters.

The input matrix can be given as 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.

The names of the parameters are given in the sections corresponding to the algorithms.

CoclustMod Algorithm All the options available for the CoclustMod algorithm are summarized below:

```
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]
                   [-l TRUE_ROW_LABELS] [--visu]
                   INPUT_MATRIX
```

The meaning of the options is given below:

optional arguments:

`-h, --help`	show this help message and exit

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

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
`--output_fuzzy_column_labels`	file path for the predicted fuzzy column labels
`--convergence_plot`	file path for the convergence plot
`--reorganized_matrix`	file path for the reorganized matrix

algorithm parameters:

`-n=2, --n_coclusters=2`	number of co-clusters
`-m=15, --max_iter=15`	maximum number of iterations
`-e=1e-09, --epsilon=1e-09`	stop if the criterion (modularity) variation in an iteration is less than EPSILON
`-i, --init_row_labels`	file containing the initial row labels, if not set random initialization is performed
`--n_runs=1`	number of runs

evaluation parameters:

`-l, --true_row_labels`	file containing the true row labels
`--visu=False`	Plot modularity values and reorganized matrix (requires numpy/scipy and matplotlib).

CoclustSpecMod Algorithm The *CoclustSpecMod* algorithm takes almost the same options (a few less), a summary of which is given below:

```
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] [-l TRUE_ROW_LABELS] [--visu]
                       INPUT_MATRIX
```

The meaning of the options is the same as for the modularity algorithm.

CoclustInfo algorithm

```
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]
             [-l TRUE_ROW_LABELS] [--visu]
             INPUT_MATRIX
```

The *CoclustInfo* algorithm being non diagonal, different numbers of clusters for the rows and the columns can be specified using the following parameters:

`-K=2, --n_row_clusters=2`	number of row clusters
`-L=2, --n_col_clusters=2`	number of column clusters

Detecting the best number of co-clusters: the coclust-nb script

The `coclust-nb` script takes almost the same arguments as `coclust modularity`. A summary is given below:

```
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]
           [--visu]
           INPUT_MATRIX
```

The number of co-clusters being unknown, the `to` and `from` parameters serve to define the range within which the number has to be searched:

`--from=2`	minimum number of co-clusters
`--to=10`	maximum number of co-clusters

For example, for the CSTR dataset, the best number of co-clusters found by the following command is 4, the same as the real number of clusters:

```
> coclust-nb datasets/cstr.csv --seed=1 --n_runs=30 --max_iter=60 --visu
```

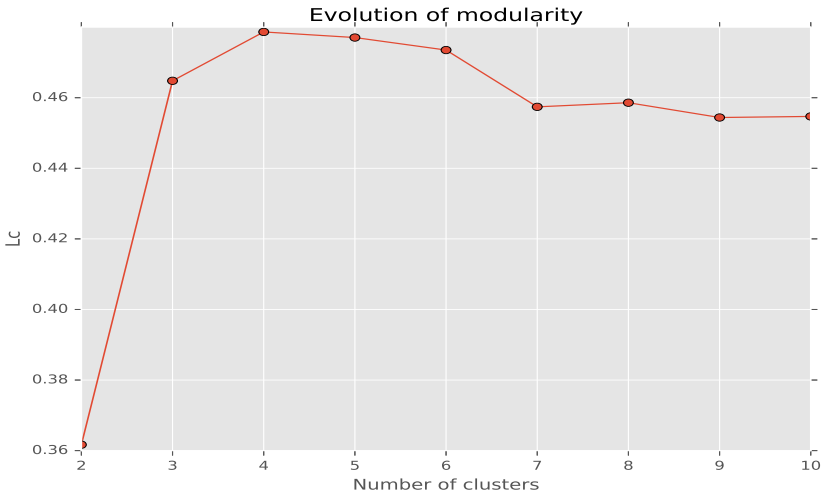

Figure 3: Using modularity to detect the best number of clusters.

In this example, the best modularity is 0.478638 and is attained for 4 co-clusters. The evolution of the modularity against the number of co-clusters is plotted in Figure 3.

3.2. Python API

Each algorithm is implemented in a specific class, all sharing the same methods as those used by **scikit-learn**, thus making it very easy to integrate with this library. The following sections contain usage examples for each of the three algorithms. The examples also demonstrate how to load matrices stored in different formats.

CoClustMod usage

The following example shows how to load the Classic3 dataset from a MATLAB file. The MATLAB file is loaded using a function provided by the **SciPy** library. A matrix is then extracted from the MATLAB dictionary and stored in variable `X`. A co-clustering model with 3 co-clusters is then created, and receives as input the `X` matrix. Then, after displaying the maximum modularity value as well as its evolution over the iterations, (Figure 4), several graphical representations of the obtained term clusters are produced via the `plot_cluster_top_terms` and `get_term_graph` functions (Figures 5 and 6). The `get_term_graph` function extracts the n most frequent terms in a given term cluster along with the k most similar (in terms of cosine similarity) neighbors of each of these most frequent terms.

```
>>> from scipy.io import loadmat
>>> from coclust.coclustering import CoclustMod
>>> from coclust.visualization import (plot_cluster_top_terms,
...                                    get_term_graph,
...                                    plot_convergence)
>>>
>>> file_name = "datasets/classic3.mat"
>>> matlab_dict = loadmat(file_name)
>>>
>>> X = matlab_dict['A']
>>>
>>> model = CoclustMod(n_clusters = 3, random_state = 0)
>>> model.fit(X)
>>>
>>> print("MODULARITY: %s" % model.modularity)
>>> plot_convergence(model.modularities, "Modularities")
>>>
>>> terms = [str(x[0][0]) for x in matlab_dict['ms']]
>>> plot_cluster_top_terms(X, terms, 10, model)
```

Output:

```
NMI: 0.914522323155
ARI: 0.946337155178
MODULARITY: 0.409347854401
```

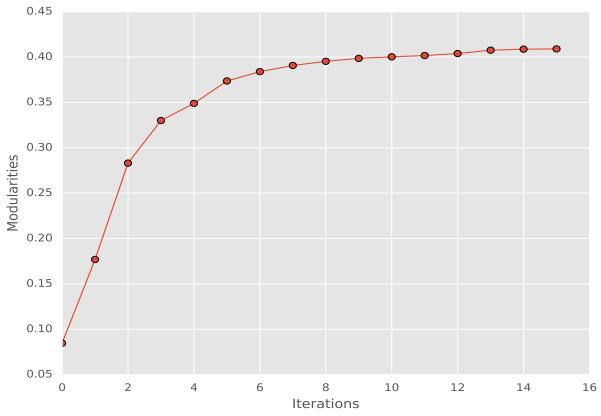

Figure 4: CoclustMod: evolution of modularity across iterations.

CoClustSpecMod usage

In this example, the CSTR dataset is imported as a CSV file. The first line of the file is the number of rows followed by the number of columns and the number of clusters the model is fitted with. The other lines are tuples of the form *(row number, column number, value)*. The

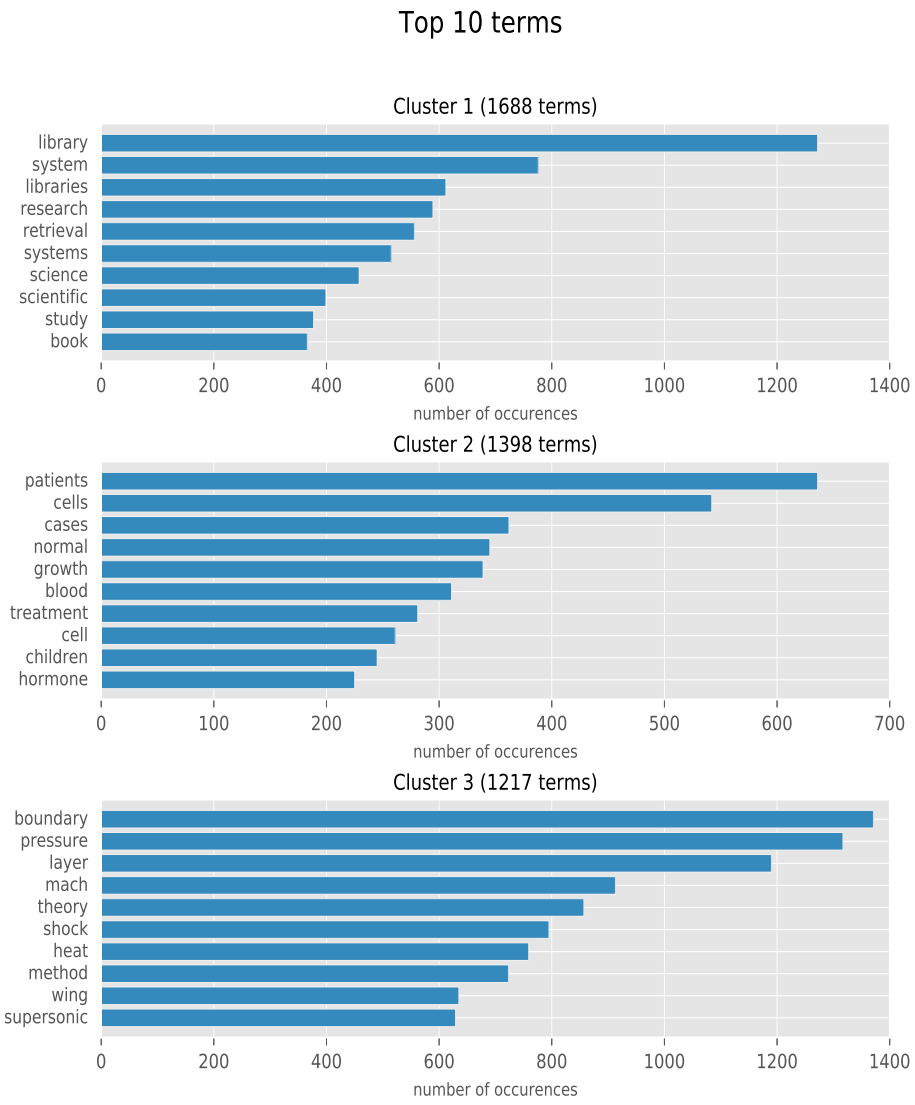

Figure 5: CoclustMod: displaying the top terms of each cluster using the `plot_cluster_top_terms` function.

spectral modularity based model is fitted, and the `plot_cluster_sizes` function (available in the `visualization` module) is then used to display the sizes of the document and term clusters (see Figure 7).

```
>>> import numpy as np
>>> from scipy.sparse import coo_matrix
>>> from coclust.coclustering import CoclustSpecMod
>>> from coclust.visualization import plot_cluster_sizes
>>>
>>> n_clust = 4
>>> file_name = "datasets/cstr.csv"
```

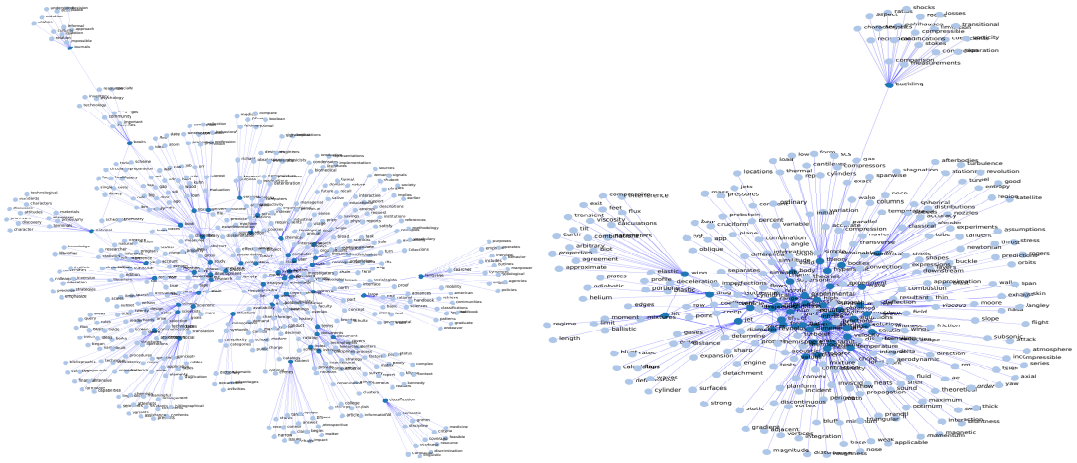

Figure 6: CoclustMod: graph representations of two term clusters. The `get_term_graph` convenience function is used to produce a representation allowing to visually detect that the cluster on the right is dense, and more thematically focused (aerodynamics) than the cluster on the left which is about more general notions (information, knowledge and science in general).

```
>>>
>>> a = np.loadtxt(file_name, delimiter = ',', skiprows = 1)
>>> X = (coo_matrix((a[:, 2], (a[:, 0].astype(int), a[:, 1].astype(int)))))
>>> X = X.tocsr()
>>>
>>> model = CoclustSpecMod(n_clusters = n_clust, random_state = 0)
>>> model.fit(X)
>>>
>>> plot_cluster_sizes(model)
```

CoClustInfo usage

In this example, the Classic3 dataset is imported from a MATLAB file. A model is created and fitted. A graph showing the evolution of the criterion is displayed along with the last γ_{kl} matrix obtained at the end of the execution. This matrix allows to visually spot the most cohesive co-clusters produced by the algorithm (see Figure 8).

```
>>> import scipy.io as io
>>> from sklearn.metrics import (adjusted_rand_score as ari,
...                              normalized_mutual_info_score as nmi)
>>> from coclust.coclustering import CoclustInfo
>>> from coclust.evaluation.external import accuracy
>>> from coclust.visualization import plot_delta_kl, plot_convergence
>>>
>>> print("1) Loading data")
>>> file_name = "datasets/classic3.mat"
>>> matlab_dict = io.loadmat(file_name)
```

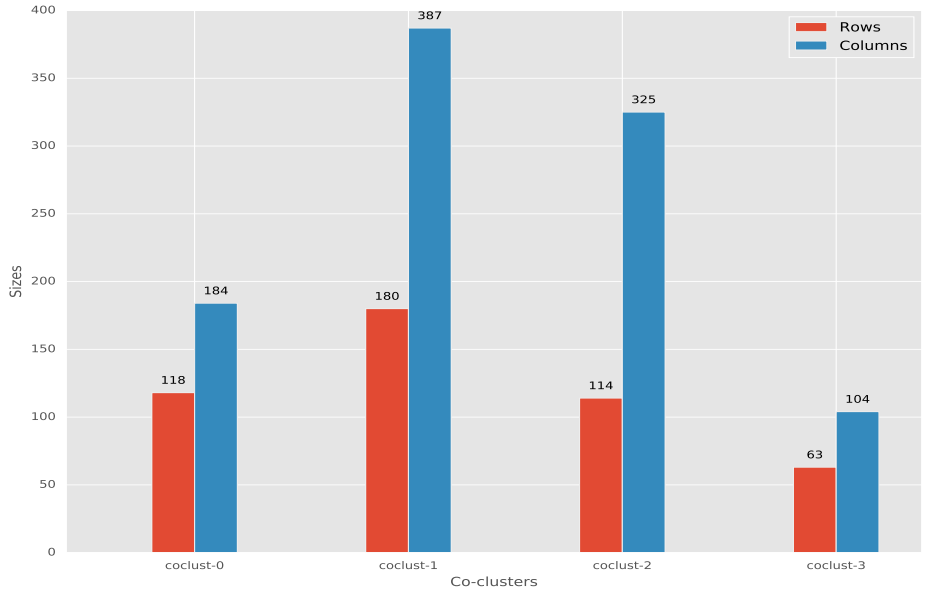

Figure 7: CoclustSpecMod: plotting the sizes of the obtained co-clusters using the `plot_cluster_sizes` utility function.

```
>>> X = matlab_dict['A']
>>>
>>> nb_clusters = 3
>>> model = CoclustInfo(n_row_clusters = nb_clusters,
...                     n_col_clusters = nb_clusters,
...                     n_init = 4, random_state = 0)
>>> model.fit(X)
>>>
>>> print("CRITERION: %s" % model.criterion)
>>> true_row_labels = matlab_dict['labels'].flatten()
>>> predicted_row_labels = model.row_labels_
>>> nmi_ = nmi(true_row_labels, predicted_row_labels)
>>> ari_ = ari(true_row_labels, predicted_row_labels)
>>> print("NMI: {}\nARI: {}".format(nmi_, ari_))
>>> accuracy = accuracy(true_row_labels, predicted_row_labels)
>>> print("ACCURACY: %s" % accuracy)
>>>
>>> plot_convergence(model.criterions, 'P_KL MI')
>>> plot_delta_kl(model)
```

Combined usage

The following example shows how easy it is to run several algorithms on the same dataset and then plot the resulting reorganized matrices in order to have a first visual grasp of what can be expected from the different algorithms. A plot of three different reorganized matrices for the **CSTR** dataset is shown in Figure 9.

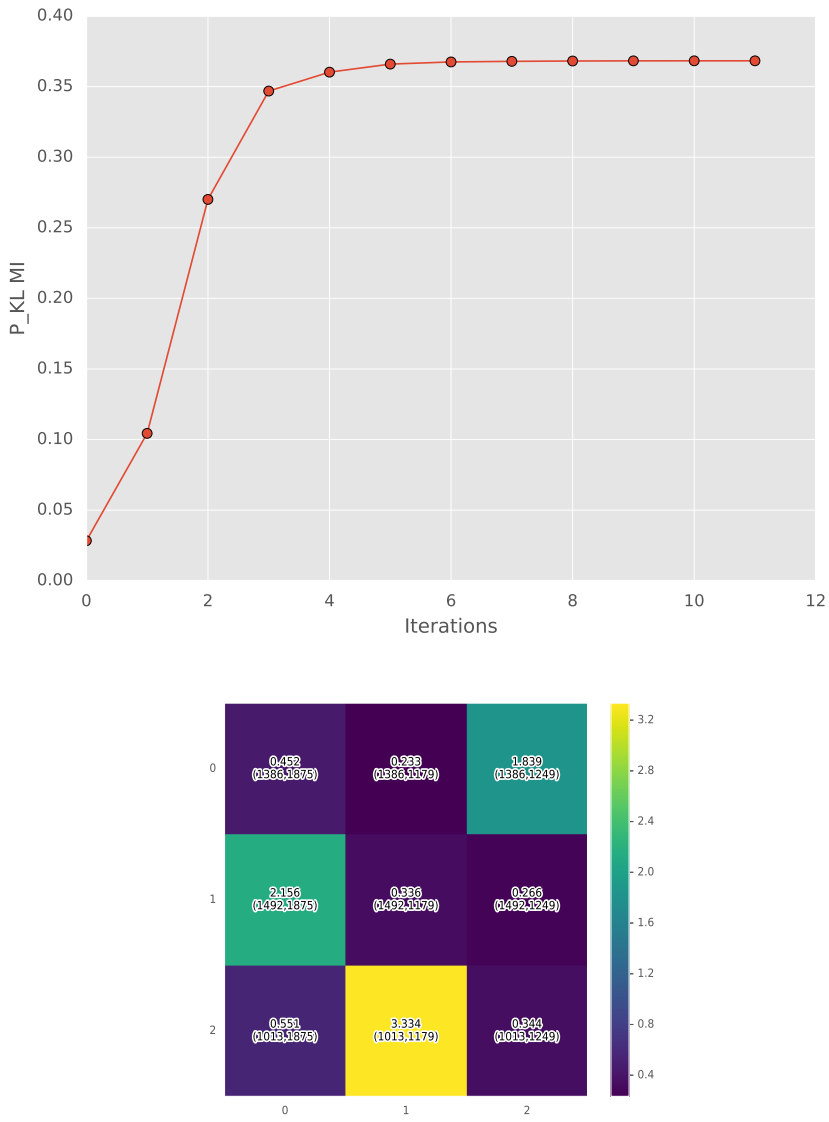

Figure 8: CoclustInfo: evolution of the objective function across iterations (top), and heatmap showing the final γ_{kl} values obtained for each row cluster k and each column cluster l. This may help to spot the interesting pairs of row ad column clusters (bottom).

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np, scipy.sparse as sp, scipy.io as io
>>> from sklearn.metrics import confusion_matrix
>>>
>>> from coclust.coclustering import (CoclustMod, CoclustSpecMod, CoclustInfo)
>>> from coclust.visualization import plot_reorganized_matrix
>>>
>>> file_name = "datasets/cstr.mat"
```

```
>>> matlab_dict = io.loadmat(file_name)
>>> X = matlab_dict['fea']
>>>
>>> model_1 = CoclustMod(n_clusters = 4, n_init = 4)
>>> model_1.fit(X)
>>>
>>> model_2 = CoclustSpecMod(n_clusters = 4, n_init = 4)
>>> model_2.fit(X)
>>>
>>> model_3 = CoclustInfo(n_row_clusters = 3, n_col_clusters = 4, n_init = 4)
>>> model_3.fit(X)
>>>
>>> plot_reorganized_matrix(X, model_1)
>>>
>>> plot_reorganized_matrix(X, model_2)
>>>
>>> plot_reorganized_matrix(X, model_3)
```

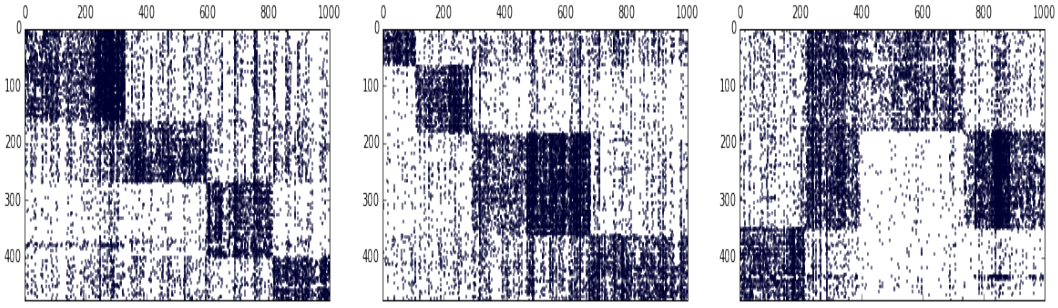

Figure 9: Using the `plot_reorganized_matrix` utility function to plot three reorganized matrices for the CSTR dataset.

3.3. Example of integration with scikit-learn

In the following example, the **scikit-learn** library is used to import the corpus of documents NG20 (see Section 4.1), select only five categories, and create a document-term matrix. This example shows how easy it is to include an algorithm of the **CoClust** package in a **scikit-learn** `Pipeline`.

```
>>> from coclust.coclustering import CoclustMod
>>> 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', CoclustMod()),
... ])
>>>
>>> pipeline.set_params(coclust__n_clusters = 5)
>>> pipeline.fit(ng5.data)
>>>
>>> predicted_labels = pipeline.named_steps['coclust'].row_labels_
>>>
>>> nmi = normalized_mutual_info_score(true_labels, predicted_labels)
>>>
>>> print(nmi)
```

For a **scikit-learn** Pipeline, the `set_params` method takes a variable number of arguments, each of the form `component__parameter`. In `pipeline.set_params(coclust__n_clusters =5)`, the component corresponding to the modularity co-clustering algorithm is named `coclust` and the parameter of the class CoClustMod we want to set to 5 is `n_clusters`.

4. Experiments

4.1. Description of datasets

To assess the performance of the three implemented algorithms, we tested them on 8 datasets of different size, sparsity and balance [6] (see Table 3). The characteristics of each dataset are reported in Table 3.

- The **CSTR** dataset was previously used in (Li 2005) and includes the abstracts of technical reports published in the Department of Computer Science of Rochester University. These abstracts were divided into 4 research fields: Natural Language Processing (NLP), Robotics/Vision, Systems, and Theory.

- **CLASSIC3** and **CLASSIC4**[7] consist respectively of 3 different document collections: CISI, CRANFIELD, and MEDLINE and 4 different document collections: CACM, CISI, CRANFIELD, and MEDLINE.

- **SPORTS** is a dataset from the **CLUTO** toolkit (Karypis 2003), and is the same as that used in (Zhong and Ghosh 2005). This dataset contains documents about 7 different sports including baseball, basketball, bicycling, boxing, football, golfing and hockey.

- **REVIEWS** is also a standard dataset used by the **CLUTO** toolkit.

- **WEBACE** (Ding and Li 2007) contains news articles partitioned across 20 different topics obtained from the WEBACE project (Han, Boley, Gini, Gross, Hastings, Karypis, Kumar, Mobasher, and Moore 1998).

- **RCV1** (Cai and He 2012) is a subset of a newswire stories corpus made available by Reuters containing 4 categories: C15, ECAT, GCAT, and MCAT.

- Finally, **NG20** is the 20 Newsgroups data set. [8]

Datasets	Characteristics				
	#Documents	#Words	#Clusters	Sparsity (%)	Balance
Cstr	475	1000	4	96.60	0.399
Weback	2340	1000	20	91.83	0.169
Classic3	3891	4303	3	98.0	0.71
Classic4	7095	5896	4	99.41	0.323
Ng20	19949	43586	20	99.99	0.991
Rcv1	9625	29992	4	99.75	0.766
Sports	8580	14870	7	99.99	0.036
Reviews	4069	18483	5	99.99	0.098

Table 3: Description of datasets.

In addition to the three algorithms included in the package (denoted as CoclustMod, CoclustSpecMod and CoclustInfo in the experiments), we also included in the comparison

[6] The balance is the ratio of the number of documents in the smallest class to the number of documents in the largest class.

[7] http://www.dataminingresearch.com/index.php/2010/09/classic3-classic4-datasets

[8] http://qwone.com/~jason/20Newsgroups/

the implementations of the two co-clustering algorithms available in **scikit-learn**, denoted as SpectralBi and SpectralCo in our experiments. [9]

4.2. Setup

The experiments were performed on a standard workstation (CPU : Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz; Memory : 8192 MB DDR3 @ 1600 MHz).[10] The reported results were obtained by running each algorithm 100 times with random initialization and averaging over the best 50 executions .[11] For SpectralBi and SpectralCo the default parameter values were used, except of course for the numbers of clusters which were set as the same values as for the other algorithms. The document-term contingency matrices were used in their original form without any pre-processing or weighting.

To evaluate the performance of the algorithms, we compared the results they generated with the true classes, by computing clustering accuracy, Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI), the number of requested clusters corresponding to the values in the fourth column of Table 3.

Clustering accuracy, denoted *Acc*, measures the extent to which each cluster contains data points from the corresponding class and is defined by $\text{Acc} = \frac{1}{n} \max_{\substack{1\leq k\leq g \\ 1\leq \ell\leq g}} [\sum_{\mathcal{C}_k,\mathcal{L}_\ell} T(\mathcal{C}_k, \mathcal{L}_\ell)]$, where $\mathcal{C}_k$ is the kth cluster in the final results, $\mathcal{L}_\ell$ is the true ℓth class and $T(\mathcal{C}_k, \mathcal{L}_\ell)$ is the number of entities that belong to class ℓ and are assigned to cluster k. The greater the clustering accuracy, the better the clustering performance. NMI (Strehl and Ghosh 2003) is estimated by:

$$\text{NMI} = \frac{\sum_{k,\ell} \frac{N_{k,\ell}}{n} \log \frac{nN_{k,\ell}}{N_k \hat{N}_\ell}}{\sqrt{(\sum_k \frac{N_k}{n} \log \frac{N_k}{n})(\sum_\ell \frac{\hat{N}_\ell}{n} \log \frac{\hat{N}_\ell}{n})}},$$

where N_k denotes the number of objects contained in the cluster $\mathcal{C}_k (1 \leq k \leq g)$, $\hat{N}_\ell$ is the number of objects belonging to the class $\mathcal{L}_\ell (1 \leq \ell \leq g)$, and $N_{k,\ell}$ denotes the number of objects that are in the intersection between cluster $\mathcal{C}_k$ and class $\mathcal{L}_\ell$. The larger the NMI, the better the quality of clustering.[12] For ARI, we used the implementation provided by **scikit-learn**.

4.3. Results and discussion

In particular, the results presented in tables 5, 6 and 7. clearly show that the three **CoClust** implementations outperform the spectral implementations available in **scikit-learn** in terms of NMI and clustering accuracy.[13] More precisely, CoclustInfo and CoclustMod perform better than their spectral competitors (including CoclustSpecMod). Also, CoclustMod provides an easy way of estimating the appropriate number of clusters (a feature implemented

[9] SpectralBi and SpectralCo are implementations of the algorithms described in (Kluger *et al.* 2003) and (Dhillon 2001).

[10] Except for the co-clustering of **NG20** using CoclustMod, which required more memory.

[11] By "best", we mean the solution optimizing the criterion used by the given algorithm (for example, maximizing the modularity for CoclustSpecMod).

[12] The datasets and benchmark code can be found in the benchmark directory of the package.

[13] In these tables the number of row and column clusters requested is specified within parentheses after the dataset name.

by the `coclust-nb` script described in Section 3.1). There, is however a marked difference in execution time between CoclustInfo and CoclustMod. A drawback of CoclustMod is that it has to handle a non sparse, modularity matrix, which is both time and memory consuming. As a consequence, in terms of execution time, CoclustMod is the slowest of the five compared algorithms. In contrast, the implementations available in **scikit-learn** are very fast but, as already said, have significantly lower accuracy and NMI scores.

Dataset	CoclustInfo	CoclustMod	CoclustSpecMod	SpectralBiclustering	SpectralCoclustering
Classic3	1.211±0.254	3.188±0.987	7.146±0.015	0.178±0.008	0.060±0.010
Cstr	0.290±0.057	0.349±0.053	0.146±0.009	1.813±0.019	0.024±0.003
Webace	1.395±0.269	1.317±0.232	1.047±0.020	1.796±0.008	0.150±0.018
Classic4	3.187±1.170	6.946±1.304	28.162±0.231	0.294±0.015	0.080±0.017
Reviews	6.222±1.780	15.713±3.737	127.104±1.123	0.616±0.036	0.257±0.037
Sports	6.865±1.482	23.658±4.984	169.982±0.676	0.751±0.046	0.303±0.041
Rcv1	8.876±2.866	42.338±6.912	118.218±1.618	0.760±0.033	0.224±0.028
Ng20	32.580±6.569	587.200±102.738	420.517±1.944	3.676±0.232	4.238±0.877

Table 4: Compared execution times.

Dataset	CoclustInfo	CoclustMod	CoclustSpecMod	SpectralBiclustering	SpectralCoclustering
Classic3	0.935±0.001	0.918±0.003	0.914±0.000	0.336±0.060	0.227±0.038
Cstr	0.653±0.027	0.591±0.032	0.717±0.000	0.136±0.031	0.308±0.039
Webace	0.614±0.009	0.595±0.013	0.568±0.010	0.316±0.024	0.406±0.015
Classic4	0.640±0.046	0.712±0.027	0.508±0.020	0.224±0.036	0.070±0.013
Reviews	0.592±0.025	0.530±0.032	0.341±0.019	0.230±0.031	0.028±0.008
Sports	0.568±0.038	0.547±0.026	0.544±0.010	0.202±0.038	0.058±0.011
Rcv1	0.489±0.023	0.469±0.034	0.012±0.003	0.124±0.026	0.009±0.001
Ng20	0.562±0.008	0.508±0.012	0.474±0.012	0.059±0.011	0.027±0.002

Table 5: NMI values.

Dataset	CoclustInfo	CoclustMod	CoclustSpecMod	SpectralBiclustering	SpectralCoclustering
Classic3	0.987±0.000	0.983±0.001	0.979±0.000	0.646±0.071	0.665±0.032
Cstr	0.713±0.058	0.714±0.052	0.817±0.000	0.399±0.023	0.609±0.036
Webace	0.514±0.017	0.583±0.023	0.501±0.020	0.290±0.024	0.346±0.019
Webace_ori	0.539±0.039	0.567±0.019	0.478±0.018	0.308±0.014	0.372±0.017
Classic4	0.781±0.082	0.888±0.018	0.596±0.011	0.582±0.026	0.427±0.014
Reviews	0.718±0.024	0.686±0.035	0.477±0.006	0.481±0.037	0.329±0.011
Sports	0.579±0.052	0.674±0.027	0.638±0.028	0.437±0.036	0.403±0.006
Rcv1	0.707±0.026	0.710±0.042	0.301±0.000	0.434±0.028	0.294±0.001
Ng20	0.484±0.020	0.394±0.021	0.283±0.020	0.088±0.005	0.084±0.002

Table 6: Accuracy values.

Dataset	CoclustInfo	CoclustMod	CoclustSpecMod	SpectralBiclustering	SpectralCoclustering
Classic3	0.962±0.001	0.948±0.002	0.941±0.000	0.305±0.070	0.265±0.046
Cstr	0.594±0.046	0.546±0.044	0.718±0.000	0.035±0.025	0.318±0.055
Webace	0.435±0.029	0.550±0.031	0.334±0.037	0.145±0.026	0.235±0.021
Classic4	0.548±0.102	0.703±0.040	0.299±0.055	0.233±0.033	-0.007±0.015
Reviews	0.621±0.042	0.529±0.053	0.184±0.022	0.142±0.036	-0.003±0.003
Sports	0.459±0.058	0.516±0.029	0.390±0.012	0.118±0.038	0.034±0.006
Rcv1	0.492±0.027	0.484±0.043	-0.000±0.000	0.096±0.024	-0.001±0.000
Ng20	0.377±0.011	0.285±0.019	0.196±0.019	0.009±0.003	0.006±0.001

Table 7: ARI values.

5. Conclusion

Co-clustering is an important technique in the era of so-called "big data" since it allows to compress large, high dimensional matrices. However, few tools were available so far for the Python community, and the **CoClust** package, therefore, aims at filling this gap. By presenting and contrasting the theory and implementation of two distinct families of co-clustering algorithms (block-diagonal and non diagonal algorithms). The paper also provides the reader with a representative survey of methods available in the co-clustering field.

Experimental results show that the three implemented algorithms adapt well to datasets of various balance and sparsity and can be used with good co-clustering performance in many settings. In particular, they clearly outperform the available Python implementations of co-clustering algorithm in terms of result quality.

In the future we plan to include model-based co-clustering algorithms. We will more specifically focus on algorithms based on the Poisson latent-block model (Govaert and Nadif 2016), but with extensions to these models to better take into account data sparsity. Adding post-processing tools for facilitating the interpretation of the produced co-clusters is another path for future work.

References

Ailem M, Role F, Nadif M (2015). "Co-clustering Document-term Matrices by Direct Maximization of Graph Modularity." In *CIKM 2015*, pp. 1807–1810.

Ailem M, Role F, Nadif M (2016). "Graph modularity maximization as an effective method for co-clustering text data." *Knowledge-Based Systems*, **109**, 160–173.

Barkow S, Bleuler S, Prelić A, Zimmermann P, Zitzler E (2006). "BicAT: A Biclustering Analysis Toolbox." *Bioinformatics*, **22**(10), 1282–1283. ISSN 1367-4803.

Bird S, Klein E, Loper E (2009). *Natural Language Processing with Python*. O'Reilly Media.

Cai D, He X (2012). "Manifold Adaptive Experimental Design for Text Categorization." *Knowledge and Data Engineering, IEEE Transactions on*, **24**(4), 707–719.

Charrad M, Ghazzali N, Boiteau V, Niknafs A (2014). "NbClust: An R Package for Determining the Relevant Number of Clusters in a Data Set." *Journal of Statistical Software*, **61**(1), 1–36. ISSN 1548-7660.

Charrad M, Lechevallier Y, Ahmed MB, Saporta G (2009). "Block Clustering for Web Pages Categorization." In *Intelligent Data Engineering and Automated Learning-IDEAL 2009*, pp. 260–267. Springer-Verlag.

Cheng Y, Church GM (2000). "Biclustering of Expression Data." In *ISMB2000, 8th International Conference on Intelligent Systems for Molecular Biology*, volume 8, pp. 93–103.

Cho H, Dhillon I (2008). "Coclustering of Human Cancer Microarrays Using Minimum Sum-squared Residue Coclustering." *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, **5**(3), 385–400.

Deodhar M, Ghosh J (2010). "SCOAL: A Framework for Simultaneous Co-clustering and Learning from Complex Data." *ACM Transactions on Knowledge Discovery from Data (TKDD)*, **4**(3), 11.

Dhillon I (2001). "Co-clustering Documents and Words Using Bipartite Spectral Graph Partitioning." In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'01, pp. 269–274.

Dhillon IS, Mallela S, Modha DS (2003). "Information-theoretic Co-clustering." In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 89–98. ACM.

Ding C, Li T (2007). "Adaptive Dimension Reduction Using Discriminant Analysis and k-means Clustering." In *Proceedings of the 24th international conference on Machine learning*, pp. 521–528. ACM.

Ding C, Li T, Peng W, Park H (2006). "Orthogonal Non-negative Matrix Tri-factorization for Clustering." *ACM SIGKDD*, pp. 126–135.

Eren K, Deveci M, Küçüktunç O, Çatalyürek ÜV (2013). "A Comparative Analysis of Biclustering Algorithms for Gene Expression Data." *Briefings in bioinformatics*, **14**(3), 279–292.

Freitas A, Ayadi W, Elloumi M, Oliveira L, Hao JK (2012). "Survey on Biclustering of Gene Expression Data." *Biological Knowl. Disc. Handbook*, pp. 591–608.

Gaujoux R, Seoighe C (2010). "A Flexible R Package for Nonnegative Matrix Factorization." *BMC bioinformatics*, **11**(1), 1.

George T, Merugu S (2005). "A Scalable Collaborative Filtering Framework Based on Co-Clustering." In *ICDM '05*, pp. 625–628. IEEE Computer Society. ISBN 0-7695-2278-5.

Govaert G, Nadif M (2003). "Clustering with Block Mixture Models." *Pattern Recognition*, **36**(2), 463–473.

Govaert G, Nadif M (2005). "An EM Algorithm for the Block Mixture Model." *IEEE Trans. Pattern Anal. Mach. Intell.*, **27**(4), 643–647.

Govaert G, Nadif M (2008). "Block Clustering with Bernoulli Mixture Models: Comparison of Different Approaches." *Computational Statistics and Data Analysis*, **52**, 3233–3245.

Govaert G, Nadif M (2013). *Co-Clustering*. John Wiley & Sons.

Govaert G, Nadif M (2016). "Mutual information, phi-squared and model-based co-clustering for contingency tables." *Advances in Data Analysis and Classification*, pp. 1–34.

Gupta N, Aggarwal S (2010). "MIB: Using Mutual Information for Biclustering Gene Expression Data." *Pattern Recognition*, **43**(8), 2692–2697.

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009). "The WEKA Data Mining Software: An Update." *SIGKDD Explor. Newsl.*, **11**(1), 10–18. ISSN 1931-0145.

Han EH, Boley D, Gini ML, Gross R, Hastings K, Karypis G, Kumar V, Mobasher B, Moore J (1998). "WebACE: A Web Agent for Document Categorization and Exploration." In *Agents*, pp. 408–415.

Hanczar B, Nadif M (2011). "Using the Bagging Approach for Biclustering of Gene Expression Data." *Neurocomputing*, **74**(10), 1595–1605.

Hanczar B, Nadif M (2012). "Ensemble Methods for Biclustering Tasks." *Pattern Recognition*, **45**(11), 3938–3949.

Hartigan JA (1972). "Direct Clustering of a Data Matrix." *Journal of the American Statistical Association*, pp. 123–129.

Henriques R, Antunes C, Madeira SC (2015). "A Structured View on Pattern Mining-based Biclustering." *Pattern Recognition*, **48**(12), 3941–3958.

Hornik K, Feinerer I, Kober M, Buchta C (2012). "Spherical k-Means Clustering." *Journal of Statistical Software*, **50**(10), 1–22.

Iovleff S, Singh Bhatia P (2015). *Blockcluster: Coclustering Package for Binary, Categorical, Contingency and Continuous Data-Sets*. R package version 4.0.2, URL https://CRAN.R-project.org/package=blockcluster.

Jones E, Oliphant T, Peterson P (2001–). "SciPy: Open Source Scientific Tools for Python." [Online; accessed 2016-06-08], URL http://www.scipy.org/.

Kaiser S, Leisch F (2008). "A Toolbox for Bicluster Analysis in R."

Karatzoglou A, Smola A, Hornik K, Zeileis A (2004). "Kernlab - An S4 Package for Kernel Methods in R." *Journal of Statistical Software*, **11**(1), 1–20.

Karypis G (2003). "CLUTO: A Clustering Toolkit." *Technical Report 02-017*, Department of Computer Science, University of Minnesota. URL http://glaros.dtc.umn.edu/gkhome/fetch/sw/cluto/manual.pdf.

Kluger Y, Basri R, Chang JT, Gerstein M (2003). "Spectral Biclustering of Microarray Cancer Data: Co-clustering Genes and Conditions." *Genome Research*, **13**, 703–716.

Labiod L, Nadif M (2011). "Co-clustering for Binary and Categorical Data with Maximum Modularity." In *11th IEEE International Conference on Data Mining, ICDM 2011, Vancouver, BC, Canada, December 11-14, 2011*, pp. 1140–1145.

Leger J (2016). "Blockmodels: A R-package for Estimating in Latent Block Model and Stochastic Block Model, with Various Probability Functions, with or without Covariates."

Li T (2005). "A General Model for Clustering Binary Data." In *KDD '05*, pp. 188–197.

Madeira SC, Oliveira AL (2004). "Biclustering Algorithms for Biological Data Analysis: A Survey." *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, **1**(1), 24–45.

Nadif M, Govaert G (2010). "Model-based co-clustering for continuous data." In *Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on*, pp. 175–180. IEEE.

Newman MEJ, Girvan M (2004). "Finding and Evaluating Community Structure in Networks." *Phys. Rev. E*.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011). "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research*, **12**, 2825–2830.

R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Strehl A, Ghosh J (2003). "Cluster Ensembles — A Knowledge Reuse Framework for Combining Multiple Partitions." *The Journal of Machine Learning Research*, **3**, 583–617.

Tanay A, Sharan R, Shamir R (2005). "Biclustering Algorithms: A Survey." *Handbook of computational molecular biology*, **9**(1-20), 122–124.

Xu G, Zong Y, Dolog P, Zhang Y (2010). "Co-clustering Analysis of Weblogs Using Bipartite Spectral Projection Approach." In *Knowledge-Based and Intelligent Information and Engineering Systems*, pp. 398–407. Springer-Verlag.

Zhong S, Ghosh J (2005). "Generative Model-based Document Clustering: A Comparative Study." *Knowledge and Information Systems*, **8**(3), 374–384.

Affiliation:

François Role, Stanislas Morbieu, Mohamed Nadif
LIPADE
Université Paris-Descartes
45 rue des Saints Pères
Paris 75006, France

E-mail: <francois.role@parisdescartes.fr>,
<stanislas.morbieu@etu.parisdescartes.fr>,
<mohamed.nadif@parisdescartes.fr>
URL: http://lipade.mi.parisdescartes.fr/?page_id=242

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd