A tutorial for \textbf{blockcluster} R package

Version 1.01

Parmeet Singh Bhatia
INRIA-Lille, parmeet.bhatia@inria.fr

Contents

1 Introduction 1

2 Package details 2
  2.1 cocluster function 2
  2.2 cocluststrategy function 3
    \hspace{1em} 2.2.1 Understanding various input parameters 5
  2.3 Example using simulated Binary dataset 6

3 Examples with real datasets 7
  3.1 Image segmentation 7
  3.2 Document clustering 8

4 Remarks 9

Abstract

\textbf{blockcluster} is a newly developed R package for co-clustering of binary, contingency and continuous data. The core library is written in C++ and \textbf{blockcluster} API acts as a bridge between C++ core library and R statistical computing environment. The package is based on recently proposed [4], [2], [3] latent block models for simultaneous clustering of rows and columns. This tutorial is based on the package version 1.01.

1 Introduction

Cluster analysis is an important tool in a variety of scientific areas such as pattern recognition, information retrieval, micro-array, data mining, and so forth. Although many clustering procedures such as hierarchical clustering, k-means or self-organizing maps, aim to construct an optimal partition of objects or, sometimes, of variables, there are other methods, called block clustering methods, which consider simultaneously the two sets and organize the data into homogeneous blocks. Let $\mathbf{x}$ denotes a $n \times d$ data matrix defined by $\mathbf{x} = \{(x_{ij}); i \in I \text{ and } j \in J\}$, where $I$ is a set of $n$ objects (rows, observations, cases etc) and $J$ is a set of $d$ variables (columns, attributes etc). The basic idea of these methods consists in making permutations of objects and variables in order to draw a correspondence structure on $I \times J$. For illustration, consider Figure 1 where a binary data set defined on set of $n = 10$ individuals $I = A, B, C, D, E, F, G, H, I, J$ and set of $d = 7$ binary variables $J = 1, 2, 3, 4, 5, 6, 7$ is re-organized into a set of $3 \times 3$ clusters by permuting the rows and columns.
Owing to ever increasing importance of Co-clustering in variety of scientific areas, we have recently developed a R package for the same called blockcluster. The R package blockcluster allows to estimate the parameters of the co-clustering models [1] for binary, contingency and continuous data. This package is unique from the point of view of generative models it implements (latent block models), the used algorithms (BEM, BCEM) and, apart from that, special attention has been given to design the library for handling very huge data sets in reasonable time. The R package is already available on CRAN at http://cran.r-project.org/web/packages/blockcluster/index.html. In this tutorial, I will elaborate the usage of our R package blockcluster.

2 Package details

This package contains two main functions namely cocluster and cocluststrategy to perform co-clustering and to set various input parameters respectively. The package also contains two helper functions namely summary and plot to get the summary of estimated model parameters and to plot the results respectively. I will first give details of the two main functions. The helper functions are self-explanatory and I will use them in various examples for better understanding.

2.1 cocluster function

This is the main function of blockcluster package that performs Co-clustering for binary, contingency and continuous data. The prototype of the function is as follows:

cocluster(data, datatype, model, nbcocluster, strategy = cocluststrategy())

The various inputs of cocluster functions are as follows:

- **data**: Input data as matrix (or list containing data matrix, numeric vector for row effects and numeric vector column effects in case of contingency data with known row and column effects.)

- **datatype**: This is the type of data which can be ”binary”, ”contingency” or ”continuous”.

- **model**: This is the name of model. The various models that are available in package are given in Table [1]
• **nbcocluster**: Integer vector specifying the number of row and column clusters respectively.

• **strategy**: This input can use to control various input parameters. It can be created using the function `cocluststrategy` as explained in Section 2.2.

The only mandatory inputs to the function `cocluster` are `data`, `datatype` and `nbcocluster`. The default model for each data-type is the most general model with free row and column proportions and unequal dispersion/variance for each block. Furthermore we have default set of input parameters which works well in most cases which are explained in further details in Section 2.2.

<table>
<thead>
<tr>
<th>Model</th>
<th>Datatype</th>
<th>Proportions</th>
<th>Dispersion/Variance</th>
<th>Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>pik_rhol_epsilonkl</td>
<td>binary</td>
<td>unequal</td>
<td>unequal</td>
<td>CEM</td>
</tr>
<tr>
<td>pik_rhol_2epsilon</td>
<td>binary</td>
<td>unequal</td>
<td>equal</td>
<td>CEM</td>
</tr>
<tr>
<td>pi_rhol_epsilonkl</td>
<td>binary</td>
<td>equal</td>
<td>unequal</td>
<td>CEM</td>
</tr>
<tr>
<td>pi_rhol_2epsilon</td>
<td>binary</td>
<td>equal</td>
<td>equal</td>
<td>CEM</td>
</tr>
<tr>
<td>pik_rhol_sigma2kl</td>
<td>continuous</td>
<td>unequal</td>
<td>unequal</td>
<td>CEM</td>
</tr>
<tr>
<td>pik_rhol_s2sigma</td>
<td>continuous</td>
<td>unequal</td>
<td>equal</td>
<td>CEM</td>
</tr>
<tr>
<td>pi_rhol_sigma2kl</td>
<td>continuous</td>
<td>equal</td>
<td>unequal</td>
<td>CEM</td>
</tr>
<tr>
<td>pi_rhol_s2sigma</td>
<td>continuous</td>
<td>equal</td>
<td>equal</td>
<td>CEM</td>
</tr>
<tr>
<td>pik_rhol_unknown</td>
<td>contingency</td>
<td>unequal</td>
<td>N.A</td>
<td>CEM</td>
</tr>
<tr>
<td>pi_rho_unknown</td>
<td>contingency</td>
<td>equal</td>
<td>N.A</td>
<td>CEM</td>
</tr>
<tr>
<td>pik_rho_known</td>
<td>contingency</td>
<td>unequal</td>
<td>N.A</td>
<td>Random</td>
</tr>
<tr>
<td>pi_rho_known</td>
<td>contingency</td>
<td>equal</td>
<td>N.A</td>
<td>Random</td>
</tr>
</tbody>
</table>

Table 1: Various models available in package `blockcluster`.

### 2.2 `cocluststrategy` function

In the package `blockcluster`, we have a function called `cocluststrategy` which can be used to set the values of various input parameters. In the following example, we call the function `cocluststrategy` without any arguments and then we called the overloaded function `summary` to see default values of various input parameters.

```r
R > defaultstrategy <- cocluststrategy()
R > summary(defaultstrategy)
```

******************************************************************
Algorithm: XEMStrategy
Initialization method(There is no default value):
Stopping Criteria: Parameter

Various Iterations

Number of global iterations while running initialization: 10
Number of iterations for internal E-step: 5
Number of EM iterations used during xem: 50
Number of EM iterations used during XEM: 500
Number of xem iterations: 5
Number of tries: 2

Various epsilons

**************
Tolerance value used while initialization: 0.01
Tolerance value for internal E-step: 0.01
Tolerance value used during xem: 1e-04
Tolerance value used during XEM: 1e-10

******************************************************************

One thing which is worth noting in the summary output (above) is that there is no default value for initialization method. It will be set automatically depending on the type of input model. To set these input parameters, we have to pass appropriate arguments to function `cocluststrategy` as shown in example below where I set `nbtry`, `nbxem` and `algo` parameters.

```
R > newstrategy <- cocluststrategy (nbtry =5 , nbxem =10 , algo = 'XCEMStrategy ')
```

The `newstrategy` object can then be passed to function `cocluster` to perform Co-clustering using the newly set input parameters. The various input arguments for the function `cocluststrategy` are as follows:

- **algo**: The valid values for this parameter are "XEMStrategy" (Default) and "XCEM-Strategy". This parameter sets the algorithm/strategy to run the model. The algorithms used are BEM (Block EM algorithm) for "XEMStrategy" and BCEM (Block classification EM algorithm) for "XCEMStrategy".

- **stopcriteria**: It specifies the stopping criteria. It can be based on either relative change in parameters value (preferred) or relative change in log-likelihood. Valid criterion values are "Parameter" and "Likelihood". Default criteria is "Parameter".

- **initmethod**: Method to initialize model parameters. The valid values are "CEMInit", "FuzzyCEMInit" and "RandomInit". For now only one kind of initialization exist for every model currently available in the package. Hence default value for initialization is set according to the model.

- **nbinititerations**: Number of Global iterations used in initialization step. Default value is 10.

- **initepsilon**: Tolerance value used inside initialization. Default value is 1e-2.

- **nbiterations_int**: Number of iterations for internal E step. Default value is 5.

- **epsilon_int**: Tolerance value for relative change in Parameter/likelihood for internal E-step. Default value is 1e-2.

- **nbtry**: Number of tries (XEM steps). Default value is 2.

- **nbxem**: Number of xem steps. Default value is 5.

- **nbiterationsxem**: Number of EM iterations used during xem step. Default value is 50.

- **nbiterationsXEM**: Number of EM iterations used during XEM step. Deafult value is 500.

- **epsilonxem**: Tolerance value used during xem step. Default value is 1e-4.

- **epsilonXEM**: Tolerance value used during XEM step. Default value is 1e-10.

To understand many of the above input parameters, we need to have some basic idea about the algorithms and the way they are run inside package `blockcluster`, which is why I have a separate dedicated section [2.2.1](#) for the same.
2.2.1 Understanding various input parameters

You might be wondering why there are so many types of iterations and tolerances inside the package. Well to get some basic understanding about various input parameters, it is important to know a bit about the algorithms. I am not going to provide here full fledged theory of these algorithms but shall give enough details to make you understand the meaning of all the input parameters. If you go through the papers of latent block models, you will see that the algorithms are called Block EM (BEM) algorithm and Block CEM (BCEM) algorithm. From now on I will explain everything using BEM but it is applicable in same way to BCEM algorithm. The BEM algorithm can be stated as follows in laymen language.

1. Run EM algorithm on rows.
2. Run EM algorithm on columns.
3. Iterate between above two steps until convergence.

We use the following strategy to run the above algorithm.

1. Run the BEM Algorithm for 'nbxem' number of times (with high tolerance and low number of iterations) and keep the best model parameters (based on likelihood) among these runs. We can this step as 'xem' step.
2. Starting with the best model parameters, run the algorithm again but this time with a low value of epsilon (low tolerance) and a high number of iterations. We can this step as 'XEM' step.
3. Repeat above two steps for 'nbtry' number of times and keep the best model estimation.

With this background, I will explain various input parameters.

- **nbxem, nbtry:** As explained above these numbers represents the number of time we run 'xem' step and 'xem'+'XEM' step respectively. The tuning of the values of 'nbxem' and 'nbtry' need to be done intuitively, and could have a substantial effect on final results. A good way to set these values is to run co-clustering few number of times and check if final log-likelihood is stable. If not, one may need to increase these values. In practice, it is better to increment 'nbxem' as it could lead to better (stable) results without compromising too much the running time.

- **nbiterationsxem, nbiterationsXEM:** These are number of iterations for BEM algorithm i.e the number of times we run EM on rows and EM on columns. As the name suggests, they are respectively for 'xem' and 'XEM' steps.

- **nbiterations_int:** This is the number of iterations for EM algorithm on rows/columns.

- **epsilonxem, epsilonXEM:** These are tolerance values for BEM algorithm during 'xem' and 'XEM' step respectively.

- **epsilon_int:** This is the tolerance value for EM algorithm on rows/columns.

- **initepsilon, nbinititerations:** These are the tolerance value and number of iterations respectively used during initialization of model parameters.
2.3 Example using simulated Binary dataset

I have simulated binary data-set with parameters given in Table 2. The class mean and dispersion are respectively represented by $a$ and $\epsilon$, whereas $\pi$ and $\rho$ represents row and column proportions respectively. The data consist of 1000 rows (samples) and 100 columns (variables) with two clusters on rows and three clusters on columns. The following R commands shows how to load the library, process the data and visualize/summarize results using `blockcluster`.

\[
\begin{array}{cccc}
 a, \epsilon & 0, 0.1 & 0, 0.3 & 1, 0.1 \\
 & 1, 0.3 & 1, 0.2 & 0, 0.1 \\
\end{array}
\]

[\pi] = [0.6, 0.4, \rho] = [0.3, 0.3, 0.4]

Table 2: Parameters for simulation of binary data.

![Figure 2: Original and co-clustered binary data (a), and distributions for each block along with various mixture densities (b).](image)

Figure 2: Original and co-clustered binary data (a), and distributions for each block along with various mixture densities (b).

```r
R > library("blockcluster")
R > data("binarydata")
R > out <- cocluster(binarydata, datatype = "binary", nbcocluster = c(2, 3))
R > summary(out)
```

**Model Family : Bernoulli Latent block model**
**Model Name : pik_rhol_epsilonkl**

Model Parameters..

Class Mean:

\[
\begin{array}{ccc}
 & [1,] & [2,] & [3,] \\
[1,] & 0 & 0 & 1 \\
[2,] & 0 & 1 & 0 \\
\end{array}
\]

Class Dispersion:

\[
\begin{array}{ccc}
 & [1,] & [2,] & [3,] \\
[1,] & 0.09798013 & 0.3022391 & 0.1011803 \\
\end{array}
\]
The following R command is used to plot the original and co-clustered data (Figure 2(a)) with aspect ratio set to false (it is true by default). When `asp` is set to false, R graphics will optimize the output figure for the display, hence the original aspect ratio may not be conserved.

\[ R > \text{plot(out, asp = 0)} \]

To Plot various block distributions (Figure 2(b)), the following R command is used with `type` argument of overloaded `plot` function set to 'distribution' (type is 'cocluster' by default which plots the original and Co-clustered data as shown in (Figure 2(a))).

\[ R > \text{plot(out, type = 'distribution')} \]

### 3 Examples with real datasets

To arouse your curiosity, I will demonstrate the applicability of package on real data. In the following sections, I gave two examples: one for Image segmentation and other for document (co-)clustering.

#### 3.1 Image segmentation

Automatic image segmentation is an important technique and have numerous applications especially in fields of medical imaging. Here I present an interesting application of co-clustering (as pre-processing step) for segmenting object(s) in image. I assume that the object pixels follows Gaussian distribution. Hence I run the `blockcluster` package with Gaussian Family model `pik_rhol_sigma2kl` on image shown in Figure 3. It can be clearly seen that the image got nicely segmented into snake and insect in two different blocks.

![Figure 3: Original and co-clustered (segmented) image.](image-url)
3.2 Document clustering

Document clustering is yet another data mining technique where co-clustering seems to be very useful. Here we run our package on one of the datasets being used in [1] which is publicly available at [ftp://ftp.cs.cornell.edu/pub/smart](ftp://ftp.cs.cornell.edu/pub/smart). We mix Medline (1033 medical abstracts) and Cranfield (1398 aeronautical abstracts) making a total of 2431 documents. Furthermore, we used all the words (excluding stop words) as features making a total of 9275 unique words. The data matrix consist of words on the rows and documents on the columns with each entry giving the term frequency, that is the number of occurrences of corresponding word in corresponding document. I assume that the term frequency follows Poisson distribution. Hence we can apply the model \( \texttt{pik\_rhol\_unknown} \) available in our package for contingency (Poisson Family) datasets with unknown row and column effects. Table 3 shows the confusion matrix and compare our results with classical bipartite spectral graph partitioning algorithm of [1], where we have obtained 100 percent correct classification. Figure 4 depicts the \(2 \times 2\) checkerboard pattern in the data matrix, hence confirming the more frequent occurrence of particular set of words in one document and vice-versa. Please note that the data matrix images are extremely sparse (data points almost invisible) and have been processed using simple image processing tools for visualization purpose only.

![Data Matrix Images](image)

<table>
<thead>
<tr>
<th></th>
<th>Medline</th>
<th>Cranfield</th>
</tr>
</thead>
<tbody>
<tr>
<td>Medline</td>
<td>1026</td>
<td>0</td>
</tr>
<tr>
<td>Cranfield</td>
<td>7</td>
<td>1400</td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th></th>
<th>Medline</th>
<th>Cranfield</th>
</tr>
</thead>
<tbody>
<tr>
<td>Medline</td>
<td>1033</td>
<td>0</td>
</tr>
<tr>
<td>Cranfield</td>
<td>0</td>
<td>1398</td>
</tr>
</tbody>
</table>

(b)

Table 3: Confusion Matrix: Results reported in [1] (a), and Results using blockcluster (b). The difference in number of Cranfield documents is because we made use of the already available data extracted from the documents and there are two less documents data in the same.

![Data Matrix Images](image)

Figure 4: Original data matrix with words on rows and documents on columns (a), and checkerboard pattern in words by documents matrix obtained after performing co-clustering (b).
4 Remarks

In this tutorial, I have given a brief introduction about the blockcluster R package. I have demonstrated the use of package using Binary data-set but the package can be used in similar fashion for other types of data. Please note that this tutorial is based on version 1.01. For future release, we have already included new functionalities into the package which I shall explain in next tutorial once we release the new package on CRAN. In the meantime, if you have any questions or suggestions, do not hesitate to contact me personally or putting it on public forum at https://gforge.inria.fr/forum/forum.php?forum_id=11190&group_id=3679.

References


