LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR STATISTIK

Sebastian Kaiser and Friedrich Leisch

A Toolbox for Bicluster Analysis in R

Technical Report Number 028, 2008
Department of Statistics
University of Munich

http://www.stat.uni-muenchen.de

A Toolbox for Bicluster Analysis in R

Sebastian Kaiser and Friedrich Leisch

Department of Statistics, Ludwig-Maximilians-Universität München,
Ludwigstrasse 33, 80539 München, Germany,
firstname.lastname@stat.uni-muenchen.de

Abstract. Over the last decade, bicluster methods have become more and more popular in different fields of two way data analysis and a wide variety of algorithms and analysis methods have been published. In this paper we introduce the R package `biclust`, which contains a collection of bicluster algorithms, preprocessing methods for two way data, and validation and visualization techniques for bicluster results. For the first time, such a package is provided on a platform like R, where data analysts can easily add new bicluster algorithms and adapt them to their special needs.

Keywords: Biclustering, Two-Way-Clustering, Software, R

This is a pre-print of an article which has been accepted for the *Compstat 2008-Proceedings in Computational Statistics*.

1 Introduction

Biclustering is an important new technique in two way data analysis. After Cheng and Church (2000) followed the initial bicluster idea of Hartigan (1972) and started to calculate bicluster on microarray data, a wide range of different articles were published dealing with different kinds of algorithms and methods to preprocess and analyze the results of such methods. Comparisons of several bicluster algorithms can be found, e.g., in Madeira and Oliveira (2004) or Prelic et al. (2006).

Consider a two-way data set of form

$$\begin{array}{c|cccccc} & c_1 & \dots & c_i & \dots & c_m \\ \hline r_1 & a_{11} & \dots & a_{i1} & \dots & a_{m1} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ r_j & a_{1j} & \dots & a_{ij} & \dots & a_{mj} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ r_n & a_{1n} & \dots & a_{in} & \dots & a_{mn} \end{array}$$

with rows r_i and columns c_j and entries a_{ij}. The goal of biclustering is to find subgroups of rows and columns which are as similar as possible to each other and as different as possible to the rest.

As noted above, the recent boom in biclustering has originated in the analysis of genetic data, where rows r_i correspond to genes and columns c_j to conditions, and a_{ij} is the expression level of gene r_i under condition c_j. The task is to find groups of genes which are co-regulated under some conditions. However, two way data appear also in other research fields. E.g., in marketing biclusters can be used to group consumers into market segments which have several preferences in common. Traditional market segmentation methods like k-means cluster analysis or mixture models use the same set of variables for all clusters, while bicluster methods are able to select different sets of variables for different segments (Goveart and Nadif (2003)).

This article is organized as follows: In Section 2 we give an introduction to the structure of R package `biclust` including a brief description of the theory and algorithms of the five bicluster methods that have been implemented yet. We also focus on preprocessing of the data, validation and visualization methods, and show the advantage of storing the resulting bicluster output in consistent classes for all methods. In Section 3, we demonstrate usage of the package using the popular yeast data (Barkow et al., 2006). Finally, we give a short summary and point out future plans for extending `biclust`.

2 R package `biclust`

Most bicluster methods have been developed for a particular data analysis problem, some authors provide standalone software for their algorithms, while others only describe the algorithms in papers. Barkow et al. (2006) provide a first toolbox with several algorithms within a single graphical user interface. While the GUI has the advantage that it is easy to use, it has also the disadvantage that it is again monolithic software, which cannot be changed easily by users, and results cannot be directly used as input for other statistical methods.

We have therefore started to implement a comprehensive bicluster toolbox in R (R Development Core Team, 2007). It provides a growing list of bicluster methods, together with pre-processing and visualization techniques, using S4 classes and methods (Chambers, 1998). The software is open source and freely available from R-Forge at `http://R-Forge.R-project.org`.

One of the main design principles of the package is to provide the results as an entity of `Biclust-Class`, an S4-class containing all information needed for postprocessing of results. It consists of the four slots `Parameters`, `RowxNumber`, `NumberxCol` and `Number`. Slot `Parameters` contains parameters and algorithm used, `Number` the number of biclusters found. The `RowxNumber` and `NumberxCol` slots represents the biclusters that have been found. They are both logical matrices of dimension (rows of data $\times$ number of biclus-

ters found) with a TRUE-value in `RowxNumber[i,j]` if row i is in bicluster j. `NumberxCol` is the same for the columns, but due to computational reasons, here the rows of the matrix represent the number of biclusters and the columns represent the columns of the data. So by simply calling

```
data[ Biclust@RowxNumber[,a] * Biclust@NumberxCol[a,] ]
```

the values of the bicluster `a` can be extracted.

Objects of class `Biclust-class` are created using a uniform interface for all bicluster methods by calls of form `biclust(x,method=BiclustMethod,...)`. This generic function takes as inputs the preprocessed data matrix `x`, a bicluster algorithm represented as a `Biclustmethod-Class` and additional arguments (`...`) for the latter.

In the following we give a brief description of the five algorithms already implemented in the package, subsection headings correspond to the name of the respective `Biclustmethod-Class`. The naming scheme is `BCxxx` where `xxx` is an abbreviation for the name of the algorithm. Some methods have been chosen because open source code from the original authors is available, others have been newly implemented to make the overall toolbox as comprehensive as possible. Of course, there is always room for improvement, and more methods will be added to the package in the future. See also van Mechelen and Schepers (2006) for a discussion on main directions of bicluster calculation. Algorithms are described in alphabetic order and, if not stated otherwise, functions were implemented in interpreted S code.

2.1 `BCBimax()`

The Bimax algorithm presented by Prelic et al. (2006) finds subgroups in a binary matrix where all entries are one. The algorithm iterates the following two steps:

1. Rearrange the rows and columns to concentrate ones in the upper right of the matrix.
2. Divide the matrix into two submatrices.

Whenever in one of the submatrices only ones are found, this submatrix is returned. In order to get satisfying results the method has to be restarted several times with different starting points.

Although the algorithm was originally designed to deliver ideas for bicluster validation, it can also be used as a bicluster method itself. In our implementation we used the original and fast C Code of Prelic et al. (2006).

2.2 `BCCC()`

The CC method implements the algorithm by Cheng and Church (2000). Starting from an adjusted matrix, where normalization or simple standard-

ization preprocessing is suggested, they define a score

$$H(I,J) = \frac{1}{\|I\|\|J\|} \sum_{i \in I, j \in J} (a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2, \tag{1}$$

where a_{iJ} is the mean of row i,a_{Ij} is the mean of column j and a_{IJ} is the overall mean. They call a subgroup a bicluster if the score is below a level α and above a δ-fraction of the whole data. The algorithm itself has three major steps:

1. Deleting rows and columns with a score larger than *alpha* times the matrix score.
2. Deleting rows and columns with largest scores.
3. Adding Rows or Columns until *alpha* level is reached.

These steps are repeated until a maximum number of biclusters is reached or no bicluster is found. The result are constant bicluster where all a_{ij} are nearly on the same level. Choosing an appropriate preprocessing methods is essential for good solutions.

2.3 `BCPlaid()`

This algorithm is an improvement of the plaid model of Lazzeroni and Owen (2002) by Turner et al. (2005). The original algorithm was fitting layers k to the model

$$Y_{ij} = (\mu_0 + \alpha_{i0} + \beta_{j0}) + \sum_{k=1}^{K} (\mu_k + \alpha_{ik} + \beta_{jk}) \rho_{ik} \kappa_{jk} + \varepsilon_{ij} \tag{2}$$

using ordinary least squares (OLS), where μ, α, β represent mean, row and column effects and ρ and κ identify if a row or column is member of the layer, respectively. After the computation of the residuals of the obtained data, the calculation has the following steps:

1. Update all parameters one after another S times.
2. Calculate the sum of squares of the layer (LSS) using the resulting parameters.
3. Compare Result with random permutation and return bicluster if LSS is higher.

The algorithm terminates when no new layer (bicluster) is found. In the new faster algorithm of Turner et al. (2005), OLS is replaced with a binary least square algorithm. In our implementation we used the original code from Turner et al. (2005).

2.4 `BCSpectral()`

The bicluster algorithm described by Kluger et al. (2003) includes several preprocessing steps, like normalization, independent scaling, bistochastization and log interactions. The goal is to find a checkerboard structure of the data matrix and in order to identify it, the following steps are performed:

1. Reorder the data matrix and choose a normalization method.
2. Compute a singular value singular value decomposition to get eigenvalues and eigenvectors.
3. Depending on the chosen normalization methods, construct biclusters beginning from the largest or second largest eigenvalue.

The quantity of bicluster depends on the number and value of the eigenvalues. The biclusters found have higher or lower values than the rows and columns around them and are arranged in a checkerboard structure.

2.5 `BCXmotifs()`

The Xmotifs algorithm of Murali and Kasif (2003) searches for rows with constant values over a set of columns. For gene expression data, they call the biclusters "conserved genes expression motifs", short "Xmotifs". Again, finding a good preprocessing method is crucial, because the main aspect of their algorithm is to define a gene state where a gene (row) is called conserved, if it has the same state in all samples (columns). One way to deal with gene states is to simply discretize the data (for example with function `discretize()`). Once the data matrix represents the states, the algorithm works by choosing a random number of columns n times and performs the following steps:

1. Choose a subset from these columns and collect all rows with equal state in this subset.
2. Collect all columns where these rows have the same state.
3. Return the bicluster if it has the most rows from all found and is also larger than a *alpha* fraction of the data.

To collect more than 1 bicluster the calculation can be reran without the rows and columns already found or just return the smaller combinations found.

This algorithm finds submatrices where all rows have the same value structure over the columns. So here it is possible to find groups with a large variance in their values in the row direction.

2.6 Other functions of package `biclust`

In addition to the cluster algorithms described above, our package provides several methods for data pre-processing, some of which have already been

described above. Other functions provide utilities for cluster validation, like an adaptation of the well known Jaccard index for comparison of two cluster results in `jaccardind()`. Function `constantVariance()` implemements the variation index of Madeira and Oliveira (2004).

Another important focus of the package is visualization of bicluster results. As the results are stored in consistent classes, it is easy to implement visualization techniques which work for results of different algorithms. Parallel coordinates (function `parallelCoordinates()`) can be used to visualize similarity of rows over columns within a bicluster. Heatmaps (`drawHeatmap()`) highlight the difference between the bicluster and the surrounding rows and columns. The bubbleplot (`Bubbleplot()`) of Santamaria et al. (2007) represents biclusters in two dimensions, its position in the graph is a two dimensional representation of the row and column combination in the cluster, the size of the bubble corresponds to the size of the bicluster. Hence, a bicluster containing another bicluster is drawn as a big bubble around a smaller one.

3 Example with yeast data

After introducing the main functions of the package we now want to show how the package works. As a standard example we ran all the algorithms on the BicatYeast data from Barkow et al. (2006). To do so the data has to be preprocessed and committed to the biclust function together with the chosen algorithm (here Xmotifs) and parameters:

```
> data(BiclustYeast)
> x<-discretize(Bicatyeast)
> res<-biclust(x, method=BCXmotifs(), alpha=0.05, number=50)
```

To visualize the result you can simply call any visualization function on the result, for example:

```
> parallelCoordinates( x=BicatYeast, result=res, bicluster=4)
```

The output of this code can be seen in Figure 1.

Table 1 shows the pairwise Jaccard indices of all bicluster algorithms. The Jaccard index is a measure of similarity between two cluster results, zero means no concordance, one means that the results are identical. It can be seen that all algorithms find very different sets of biclusters. This can be partly explained by different pre-processing steps which were necessary such that the data conform to the respective assumptions of the algorithms. Another important aspect is that we selected the first algorithms to implement to get a collection of algorithms which differ from each other as much as possible. It is now very easy for practitioners to try various bicluster methods in R and choose the one which works best for given data set.

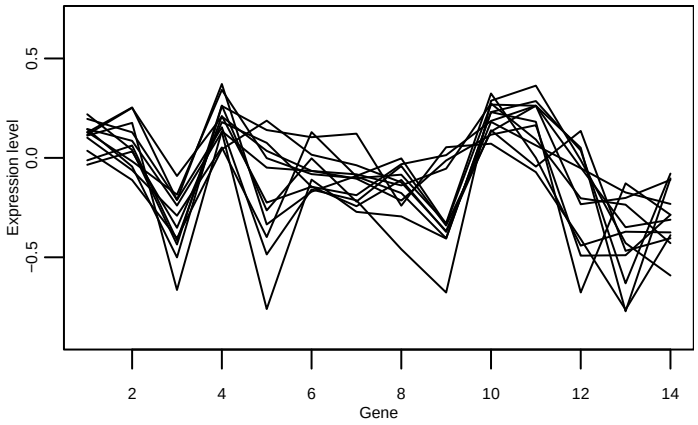

Fig. 1. Example for parallel coordinates plot: Expression levels of conditions across their genes in the 4th bicluster in the result of the Xmotifs algorithm.

	BCPlaid	BCXmotifs	BCCC	BCSpect.	BCBimax
BCPlaid	1.0000	0.0007	0.0116	0.0000	0.0000
BCXmotifs	0.0007	1.0000	0.1789	0.0935	0.0000
BCCC	0.0116	0.1789	1.0000	0.0898	0.0036
BCSpectral	0.0000	0.0935	0.0898	1.0000	0.0000
BCBimax	0.0000	0.0000	0.0036	0.0000	1.0000

Table 1. Bicluster results similarity measure with an adaptation of Jaccard index

4 Summary and future work

In this article, we gave a short introduction to R package `biclust` with special emphasis on the algorithms that have already been implemented. We explained some of the design principles and object structures of the packages, and demonstrated usage of the software on a real word data set. All methods implemented share common infrastructure for data pre-processing, storing results and visualization. It is now very easy to add new bicluster methods, because the modular design allows for re-use of existing building blocks.

As a next step, we will do benchmark experiments comparing all the algorithms. As demonstrated in the example above, bicluster results do strongly depend on the clustering algorithm. It will be interesting to investigate which algorithm works best for which type of data, and how sensitive the algorithms are with respect to their parameters and data pre-processing steps.

5 Acknowledgments

Package `biclust` is joint work with Rodrigo Santamaria.

References

BARKOW, S., BLEULER, S., PRELIC, A., ZIMMERMANN, P., and ZITZLER, E. (2006): Bicat: a biclustering analysis toolbox. *Bioinformatics, 22,1282–1283.*

CHAMBERS, J. M. (1998): *Programming with data: A guide to the S Language.* Chapman & Hall, London.

CHENG, Y. and CHURCH, G. M. (2000): Biclustering of expression data. In: *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, 1,93–103.

GOVEART, G. and NADIF, M. (2003): Clustering with block mixture models. *Pattern Recognition*, 36, 463–473.

HARTIGAN, J.A. (1972): Direct Clustering of a data matrix. *Journal of The American Statistical Association, 67,12079–12084.*

KLUGER, Y., BASRI, R., CHANG, J. T., and GERSTEIN, M. (2003): Spectral biclustering of microarray data: Coclustering genes and conditions. *Genome Research, 13,703–716.*

LAZZERONI, L. and OWEN, A. (2002): Plaid models for gene expression data. *Statistica Sinica, 12,61–86.*

MADEIRA, S. C. and OLIVEIRA, A. L. (2004): Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics, 1(1),24–45.*

VAN MECHELEN, I. and SCHEPERS, J. (2006): A unifying model for biclustering. In: *Compstat 2006 - Proceedings in Computational Statistics*, 81–88.

MURALI, T. and KASIF, S. (2003): Extracting conserved gene expression motifs from gene expression. In: *Pacific Symposium on Biocomputing*, 8,77–88.

PRELIC, A., BLEULER, S., ZIMMERMANN, P., WIL, A., BÜHLMANN, P., GRUISSEM, W., HENNING, L., THIELE, L., and ZITZLER, E. (2006): A systematic comparison and evaluation of biclustering methods for gene expression data. *Bioinformatics, 22(9),1122–1129.*

SANTAMARIA, R., THERON, R., and QUINTALES, L. (2007): A framework to analyze biclustering results on microarray experiments. In: *8th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL'07)* ,Springer, Berlin, 770–779.

TURNER, H., BAILEY, T., and KRZANOWSKI, W. (2005): Improved biclustering of microarray data demonstrated through systematic performance tests. *Computational Statistics and Data Analysis, 48,235–254.*