Biclustering: Methods, Software and Application

Sebastian Kaiser

München 2011

Biclustering: Methods, Software and Application

Dissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften am Institut für Statistik
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

Vorgelegt von
Sebastian Kaiser
am 7. März 2011
in München

Erstgutachter: Prof. Dr. Friedrich Leisch, LMU München
Zweitgutachter: Prof. Dr. Luis A. M. Quintales, Universidad de Salamanca
Rigorosum: 12. Mai 2011

Research: the final frontier. These are the voyages of a young research fellow. His five-year mission: to explore strange new methods, to seek out new software and new applications, to boldly go where no man has gone before.

Based on Star Trek

Danksagung

Danken möchte ich:

... meinem Doktorvater Fritz Leisch für das Ermöglichen dieser Dissertation, für die vielen Freiheiten über die letzten Jahre, für die zahlreichen Möglichkeiten, das wissenschaftliche Leben kennen zu lernen und die immer offene Tür.

... den weiteren Gutachtern und Prüfern Luis Quintales, Helmut Küchenhoff, Christian Heumann und Volker Schmid für ihr Interesse und ihre Zeit.

... meinen Kollegen am Institut für Statistik für die angenehme Arbeitsatmosphäre und das gute Klima, insbesondere Manuel Eugster, der mir das Arbeiten zum Vergnügen hat werden lassen, und Carolin Strobl, die immer wieder für Abwechslung im Büro gesorgt hat.

... der Visualisierungsgruppe aus Salamanca, insbesondere Rodrigo Santamaria für die großartige Zusammenarbeit am biclust Paket.

... der Arbeitsgruppe aus Hasselt und Martin Sill für die Zusammenarbeit an den Bicluster Projekten.

... Sara Dolnicar und ihrer Arbeitsgruppe in Australien für den wunderschönen Aufenthalt und die tolle Zusammenarbeit.

... meinen Eltern, meiner Schwester Anna, meiner Oma Helga und meiner restlichen Familie für den Rückhalt und das schöne Leben.

... meiner Freundin Maria, möge es immer so perfekt bleiben wie es ist.

... Fabian Barth für die Mittagessen am Freitag und den Kontakt zur RSU - **Suit up!**

Abstract

Over the past 10 years, biclustering has become popular not only in the field of biological data analysis but also in other applications with high-dimensional two way datasets. This technique clusters both rows and columns simultaneously, as opposed to clustering only rows or only columns. Biclustering retrieves subgroups of objects that are similar in one subgroup of variables and different in the remaining variables. This dissertation focuses on improving and advancing biclustering methods. Since most existing methods are extremely sensitive to variations in parameters and data, we developed an ensemble method to overcome these limitations. It is possible to retrieve more stable and reliable bicluster in two ways: either by running algorithms with different parameter settings or by running them on sub- or bootstrap samples of the data and combining the results. To this end, we designed a software package containing a collection of bicluster algorithms for different clustering tasks and data scales, developed several new ways of visualizing bicluster solutions, and adapted traditional cluster validation indices (e.g. Jaccard index) for validating the bicluster framework. Finally, we applied biclustering to marketing data. Well-established algorithms were adjusted to slightly different data situations, and a new method specially adapted to ordinal data was developed. In order to test this method on artificial data, we generated correlated original random values. This dissertation introduces two methods for generating such values given a probability vector and a correlation structure.

All the methods outlined in this dissertation are freely available in the `R` packages `biclust` and `orddata`. Numerous examples in this work illustrate how to use the methods and software.

Zusammenfassung

In den letzten 10 Jahren wurde das Biclustern vor allem auf dem Gebiet der biologischen Datenanalyse, jedoch auch in allen Bereichen mit hochdimensionalen Daten immer populärer. Unter Biclustering versteht man das simultane Clustern von 2-Wege-Daten, um Teilmengen von Objekten zu finden, die sich zu Teilmengen von Variablen ähnlich verhalten. Diese Arbeit beschäftigt sich mit der Weiterentwicklung und Optimierung von Biclusterverfahren. Neben der Entwicklung eines Softwarepaketes zur Berechnung, Aufarbeitung und graphischen Darstellung von Bicluster Ergebnissen wurde eine Ensemble Methode für Bicluster Algorithmen entwickelt. Da die meisten Algorithmen sehr anfällig auf kleine Veränderungen der Startparameter sind, können so robustere Ergebnisse erzielt werden. Die neue Methode schließt auch das Zusammenfügen von Bicluster Ergebnissen auf Subsample- und Bootstrap-Stichproben mit ein. Zur Validierung der Ergebnisse wurden auch bestehende Maße des traditionellen Clusterings (z.B. Jaccard Index) für das Biclustering adaptiert und neue graphische Mittel für die Interpretation der Ergebnisse entwickelt.

Ein weiterer Teil der Arbeit beschäftigt sich mit der Anwendung von Bicluster Algorithmen auf Daten aus dem Marketing Bereich. Dazu mussten bestehende Algorithmen verändert und auch ein neuer Algorithmus speziell für ordinale Daten entwickelt werden. Um das Testen dieser Methoden auf künstlichen Daten zu ermöglichen, beinhaltet die Arbeit auch die Ausarbeitung eines Verfahrens zur Ziehung ordinaler Zufallszahlen mit vorgegebenen Wahrscheinlichkeiten und Korrelationsstruktur.

Die in der Arbeit vorgestellten Methoden stehen durch die beiden `R` Pakete `biclust` und `orddata` allgemein zur Verfügung. Die Nutzbarkeit wird in der Arbeit durch zahlreiche Beispiele aufgezeigt.

Contents

List of Figures

List of Tables

1. Introduction

Assume we are given a typical rectangular data matrix, in which rows correspond to objects and columns to variables. Examples from different application areas include

microarray gene expression data: objects = genes, variables = conditions or experiments

marketing data: objects = customers or consumers, variables = product features

text mining: objects = documents, variables = words

We are now interested in finding homogenous groups of objects, such as coregulated genes or market segments. In all applications, it is reasonable to assume that different groups may be defined by different variables. For example, if we want to sell a car, we must take into consideration that one group of customers will be mainly interested in price, consumption and safety features, while another group will be interested in vehicle handling, horse power and sportiness.

If the data matrix contains many variables but groups are defined by just a few of them, standard partitioning cluster algorithms, such as k-means, often lead to diffuse results, and it is impossible to make an accurate classification. This is a well-known problem; the simultaneous clustering of objects and the selection of variables for each cluster was first proposed by Hartigan (1972). However, this method found no application for almost 30 years. Clustering gene expression data brought biclustering back into focus because of the high dimensionality of the data sets.

The seminal paper by Cheng and Church (2000) was followed by a wide range of articles which proposed new algorithms (Getz et al., 2000; Lazzeroni and Owen, 2002; Tanay et al., 2002; Ben-Dor et al., 2003; Murali and Kasif, 2003), compared algorithms, (Madeira and Oliveira, 2004; Tanay et al., 2005; Prelic et al., 2006), and presented software implementations (Barkow et al., 2006).

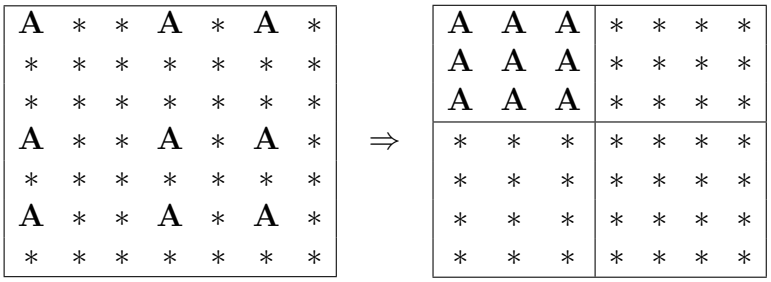

Fig. 1.1: Biclustering finds objects and variables with a similar value A and reports them as a bicluster (submatrix).

Newer developments show visualization (Santamaría et al., 2008) and validation (Santamaría et al., 2007) methods.

It is difficult to place biclustering in the cluster framework, since there are certain similarities as well as differnces. Kriegel et al. (2009) give an overview of clustering high-dimensional data. They consider biclustering to be a pattern-based clustering compared to other subspace- and correlation- cluster frameworks. Mechelen et al. (2004) examine two-mode clustering methods in more detail. They see biclustering as a two-mode clustering which allow cluster overlap and does not cluster the whole dataset.

This dissertation focuses on improving and advancing bicluster methods. Chapter 2 provides some theoretical background on biclustering and shows the different types of bicluster and the different structures of bicluster result sets. This chapter also includes an overview of popular bicluster algorithms. Chapter 3 introduces our ensemble method for bicluster algorithms. Instead of running an algorithm once, we combine multiple runs of one or more algorithms and combine the results to retrieve more stable and reliable results. The two methods described in Chapter 4 for drawing correlated ordinal values are necessary for testing bicluster algorithms which work on ordinal data. This type of data is used in Chapter 6, in which we present the theoretical background on two techniques, a binary and a mean mapping method. We test the performance of these methods and provide some examples of how they are used.

Our software package for calculating, validating and visualizing bicluster results is introduced in Chapter 5. We outline the theory of all methods used in the package, provide some illustrative examples, and give an overview of other available software. Chapter6 includes different applications of biclustering beyond the well-established microarray data analysis. We focus on marketing data, in particular binary and ordinal questionnaires from tourism research, but give also examples of sports data analysis. This chapter also contains some remarks on the performance of biclustering as compared to traditional k-means

and hierarchical clustering. The final Chapter summarizes the dissertation and gives a brief outlook of future developments.

Parts of this dissertation have been presented at scientific conferences, submitted to journals, published as journal articles and/or published as freely available software packages:

Chapter 3: Pfundstein (2010). Ensemble Methods for Plaid Bicluster Algorithm. *Bachelor Thesis*, 2010.

Chapter 4: Kaiser, Träger, and Leisch (2011). Generating Correlated Ordinal Random Values. *Submitted*, 2011.
Träger (2009). Generating Correlated Ordinal Random Values. *Diploma Thesis*, 2009.

Chapter 5: Kaiser and Leisch (2008). A Toolbox for Bicluster Analysis in R. *Compstat 2008—Proceedings in Computational Statistics*, 55(3), pages 201–208, 2008.
Sill, Kaiser, Benner, and Kopp-Schneider (2011). Robust biclustering by sparse singular value decomposition incorporating stability selection. *Bioinformatics*, 2011.
Khamiakova, Kaiser, and Shkedy (2011). Goodness-to-Fit and Diagnostic Tools Within the Differential Co-expression and Biclusters Setting. *Unpublished*, 2011.

Chapter 6: Dolnicar, Kaiser, Lazarevski, and Leisch (2011). BICLUSTERING Overcoming data dimensionality problems in market segmentation. *Journal of Travel Research*, 2011.

Appendix A: Kaiser, Santamaria, Sill, Theron, Quintales, and Leisch (2011). `biclust`: BiCluster Algorithms. R package version 1.0. `http://cran.r-project.org/package=biclust`.

Appendix B: Kaiser and Leisch (2010). `orddata`: Generation of Artificial Ordinal and Binary Data. R package version 0.1. `https://r-forge.r-project.org/projects/orddata/`.

2. Biclustering

As usual in cluster analysis, we start with an $n \times m$ data matrix A:

	y_1	$\dots$	y_i	$\dots$	y_m
x_1	a_{11}	$\dots$	a_{i1}	$\dots$	a_{m1}
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
x_j	a_{1j}	$\dots$	a_{ij}	$\dots$	a_{mj}
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
x_n	a_{1n}	$\dots$	a_{in}	$\dots$	a_{mn}

with objects X, variables Y and entries a_{ij}. The goal of bicluster analysis is to find subgroups A_{IJ} of objects $I = \{i_1, ..., i_k\}$, $k \leq n$, $I \subset X$ which are as similar as possible to each other on a subset of variables $J = \{j_1, ..., j_l\}$, $l \leq m$, $J \subset Y$ and as different as possible to the remaining objects and variables. Bicluster z is then defined as $BC_z = (I_z, J_z) = A_{I_z J_z}$.

A typical situation to calculate bicluster are a high dimensional dataset with many variables, so that normal cluster algorithms lead to diffuse results due to many uncorrelated variables. Also biclustering is useful if there is a assumed connection of objects and some of the variables in the dataset, e.g. some objects have 'similar' patterns for a given set of variables.

2.1 Bicluster Types

Just as in traditional clustering, there are many possibilities for calculating similarity within a bicluster, which is why so many different algorithms have been published. Madeira and Oliveira (2004) identified four major groups of structures inside the submatrices (examples are given in Figure 2.1):

1. Bicluster with constant values:

$$a_{ij} = \mu$$

2. Bicluster with constant values on rows or columns :

$$(a_{ij} = \mu + \alpha_i \text{ or } a_{ij} = \mu * \alpha_i) \text{ and } (a_{ij} = \mu + \beta_j \text{ or } a_{ij} = \mu * \beta_j)$$

3. Bicluster with coherent values:

$$a_{ij} = \mu + \alpha_i + \beta_j \quad \text{or} \quad a_{ij} = \mu * \alpha_i * \beta_j$$

4. Bicluster with coherent evolutions.

$$a_{ih} \leq a_{ir} \leq a_{it} \leq a_{id} \quad \text{or} \quad a_{hj} \leq a_{rj} \leq a_{tj} \leq a_{dj}$$

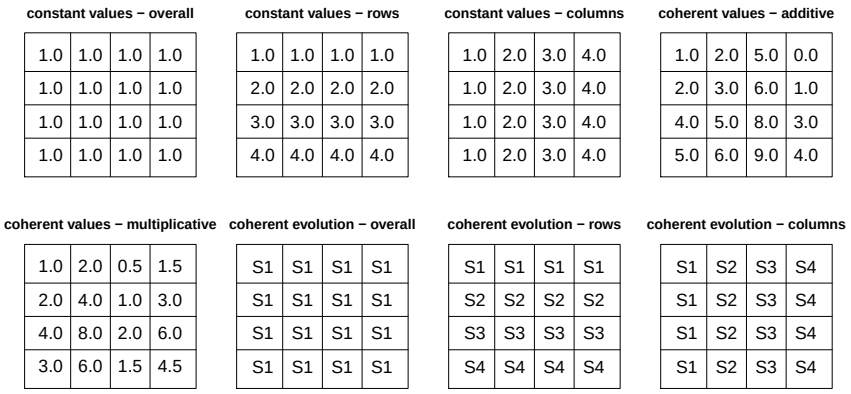

Fig. 2.1: Examples of different bicluster types.

In the simplest case, the algorithm is able to find subsets of rows and columns with constant values. Slightly enhanced methods can identify bicluster with either constant values on the rows or constant values on the columns. Other approaches look for coherent values on the columns or rows of the expression matrix. This means, each column or row can be calculated by simply adding or multiplying a constant. A further type aims to find bicluster with coherent evolutions. In other words, the exact numeric value of the matrix elements does not matter. Instead the algorithm searches for subsets of columns and rows with coherent behaviors. It is obvious that this case is accompanied by a loss of information, as the matrix has to be discretized since the exact numeric values of the matrix does not matter.

2.2 Bicluster Structures

Madeira and Oliveira (2004) also differentiate bicluster sets according to their relative structure. This structure describes how the observed bicluster can potentially be arranged (Examples are given in Figure 2.2):

1. Single bicluster.

2. Exclusive row and column bicluster.

3. Exclusive-rows or exclusive-columns bicluster.

4. Non-overlapping non-exclusive bicluster.

5. Arbitrarily positioned overlapping bicluster.

The simplest way of structuring bicluster is to find the single largest bicluster and to delete rows and columns from the data, allowing exclusive row and column bicluster to be obtained. Exclusive row or column bicluster can be obtained by deleting either row or columns. The true challenge indeed is to detect overlapping or at least non-exclusive bicluster.

Fig. 2.2: Examples of the different bicluster structures. Note that rows and columns of a single bicluster typically are not together in a block, but rather have to be reordered in a simplified block structure. The figure above is based on Madeira and Oliveira (2004).

2.3 Jaccard Index

The most important part for bicluster validation is the comparison of a found bicluster to a well known orallready found bicluster. We use an adaption of the Jaccard index (Jaccard, 1901) from ordinary clustering. To compare to bicluster it calculates the fraction of row-column combinations in both bicluster from all row-column combination in at least one bicluster:

$$jac(BC_i, BC_j) = jac_{ij} = \frac{|BC_i \cap BC_j|}{|BC_i \cup BC_j|}. \quad (2.1)$$

For two bicluster this an easy task, but for two sets of bicluster this is more complicated. We define the Jaccard index of two non-overlapping bicluster result sets as

$$jac(Bicres_1, Bicres_2) = \frac{1}{g} \sum_{i=1}^{g} \sum_{j=1}^{t} (\frac{|BC_i(Bicres_1) \cap BC_j(Bicres_2)|}{|BC_i(Bicres_1) \cup BC_j(Bicres_2)|}), \quad (2.2)$$

where g, t are the number of bicluster in bicluster result $1, 2$.

If there is overlapping, the Jaccard index has to be corrected with the maximum of the Jaccard index of the two result sets with themselves,

$$jac_c(Bicres_1, Bicres_2) = \frac{jac(Bicres_1, Bicres_2)}{\max(jac(Bicres_1, Bicres_1); jac(Bicres_2, Bicres_2))}$$

Since $\max(jac(Bicres_1, Bicres_1); jac(Bicres_2, Bicres_2))$ is 1 for non overlapping, we always use the corrected Jaccard index in calculations.

2.4 Bicluster Algorithms

Nowadays there is an abundance of algorithms for finding all kinds of bicluster structures. In the following we will outline the algorithms which will appear in this work. We will provide a small introduction of the algorithms (in alphabetic order) including our developments s4vd and Quest. A more detailed look at the algorithms used in our software package together with a working description is presented in Chapter 5. The latter section contains an incomplete listing of other notable algorithms.

2.4.1 Used Algorithms

Bimax

The Bimax algorithm of Prelic et al. (2006) is a method for finding subgroups of 1 values in a binary matrix. If only distinct values or intervals of the data are interesting, and if it is possible to mark them with 1s in a binary matrix, the Bimax algorithm finds subgroups containing only such values.

Prelic et al. (2006) used this method to compare different other algorithms to a constant benchmark, but the method is also useful in many other application fields where binary or quasi-binary data is used.

CC or δ -biclustering

The algorithm by Cheng and Church (2000) searches for bicluster with constant values, rows or columns. Starting from an adjusted matrix, they define a score

$$H(I,J) = \frac{1}{\|I\|\|J\|} \sum_{i \in I, j \in J} (a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2, \tag{2.3}$$

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 δ and above an *alpha*-fraction of the score the whole data.

ISA

The iterative signature algorithm of Bergmann et al. (2003) for bicluster contains very high or very low values. It starts with a random set of rows and iterates between normalized rows and normalized columns to find the largest subgroup of extreme values. Due to the normalization quantiles of the normal distributions can be used to identify extreme values. In each iteration the corresponding row or column vector is updated until changes no longer occur.

Plaid Models

The original plaid models for biclustering, defined by Lazzeroni and Owen (2002), fit layers k to the model

$$a_{ij} = (\mu_0 + \alpha_{i0} + \beta_{j0}) + \sum_{k=1}^{K} (\mu_k + \alpha_{ik} + \beta_{jk}) \rho_{ik} \kappa_{jk} + \varepsilon_{ij} \tag{2.4}$$

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. Turner et al. (2005) replaced the OLS with a binary least square algorithm and obtained better results.

Quest

We developed the Quest algorithm, which contains three methods dealing with different scale levels data, especially for biclustering questionnaire data. This

algorithm works like the Xmotifs algorithm if the answers are given on a nominal scale. For the ordinal scale, the algorithm looks for similar answers in an interval of a size set by a prespecified parameter. The interval contains d lower and d higher values than the starting class of the chosen respondents. In a continuous case, this interval is set by the quantile of a chosen distribution. It uses all previously found values, to calculate the mean value and uses a given parameter for the variance of this distribution. Since normal scores are used in such data the normal distribution is commonly used in this cases.

Spectral Biclustering

The bicluster algorithm described by Kluger et al. (2003) uses a singular value decomposition and the resulting eigenvalues and eigenvectors to retrieve bicluster from the data. This leads to a checkerboard bicluster structure. The algorithm is very sensitive to data variations and therefore needs a very careful preprocessing which Kluger et al. (2003) included into their algorithm. The number of bicluster is determined by a chosen upper border for the variance within the bicluster.

sv4d

In Sill et al. (2011) we optimized the idea of biclustering via a sparse singular value decomposition (Lee et al., 2010). A checkerboard bicluster structure is found forcing the row and column singular vectors to be very sparse. Lee et al. (2010) achieved this structure by interpreting the singular vectors as regression coefficients. Our approach finds these sparse singular values using stability selection methods, leading to more stable results which can also be interpreted more easily.

Xmotifs

Bicluster with coherent evolutions are represented by the Xmotifs algorithm of Murali and Kasif (2003). This algorithm searches for rows with constant values over a set of columns. For gene expression data, they call the bicluster "conserved genes expression motifs", shortened to "Xmotifs". For this application it is crucial to find a good preprocessing method. This is because the main purpose of the algorithm is to define a gene (row) state which is equal in the chosen samples (columns). This is called a conserved gene (row). One way of dealing with gene states is to simply discretize the data.

2.4.2 Other Algorithms

So many different algorithms are published today, that it is a difficult task indeed to keep track of all developments. In order to give the reader an idea of some of the new findings we will briefly list some additional methods. The original block clustering from Hartigan (1972) finds blocks in the data with minimal variance. The cMonkey framework of Reiss et al. (2006) models a bicluster using a Markov Chain process. The Coupled Two Way Clustering of Getz et al. (2000) combines single cluster algorithms in both dimension to obtain a bicluster result. The Samba algorithm (Tanay et al., 2002) includes a test method for significant results, while the order preserving sub matrix method of Ben-Dor et al. (2003) tries to find subgroups with equal ordered values. Sheng et al. (2003) use a Gibbs sampling to obtain the bicluster and Ji et al. (2006) obtain hierarchical ordered bicluster from their algorithm. Newer developments such as Hochreiter et al. (2010) use factor analysis to retrieve distinct bicluster. For a more detailed overview, see Madeira and Oliveira (2004).

3. Ensemble Method

As we stated above, bicluster outcomes vary in structure and type when different algorithms are used. Most of the algorithms we introduced are also very sensitive to small parameter changes and/or minor changes in the data. Additionally, some methods depend on the starting values and so lead to varying results on repeating runs. Unlike traditional clustering, biclustering does not deliver a perfect separation of the data, resulting in overlap in rows and/or columns. Furthermore, not every row or column must appear in a bicluster.

When dealing with these properties one can clearly see that one run with one bicluster algorithm may not lead to a stable result. This is especially true for new data sets without prior knowledge in the structure. In order to avoid the problem of parameter selection and to obtain more stable and reliable results, we propose an ensemble method for biclustering. Differnet to Hanczar and Nadif (2010) who only use one algorithm on different bootstrap samples, we employ one or more algorithms several times using different parameter settings and/or subsamples of the data. The resulting bicluster are combined using a similarity measure (e.g. Jaccard index). The combined bicluster are returned if they fulfill certain conditions, most of which are linked to the number of appearances in the various runs. Similar approaches exist for traditional clustering (Wilkerson and Hayes, 2010).

In the first part of this chapter, we describe the theoretical background of the ensemble method, including different possibilities for combining the results. We will then provide some illustrative examples based on both artificial and real data.

3.1 Ensemble Method

We propose an ensemble method in order to construct more stable and reliable bicluster. Even if a algorithm only allows single bicluster to be found, our method is able to detect an overlapping structure using either different parameter settings, sub-/bootstrap- samples, or both.

Ensemble methods use multiple runs of one or more algorithms on slightly modified conditions, either parameter modifications, stochastic starting points, or data sampling. This data must be combined using a weighting method in order to retrieve a useful result. The most popular methods are bagging or boosting (Bühlmann, 2010). The challenge of these methods is to define a combining algorithm which leads to meaningful results.

Our ensemble approach consists of three steps. In the first step algorithms and parameters must be chosen and it must be decided whether to use replication or alternatively, a sub- or bootstrap-sampling. In the second step the retrieved bicluster result is processed. The third and last step uses the combined bicluster to form a bicluster result set.

3.1.1 Initialization Step

As stated above in our approach, one can choose which algorithms to use in the ensemble method. One has to make a careful selection, since most algorithms search for different outcome structures.One must at least examine the combined results more closely, especially the proportion of bicluster from a specific algorithm. The desired parameter setting must be determined for each chosen algorithm. An expand grid from parameter intervals and a number of equal steps within these intervals should be used. One should also examine the parameter settings more carefully. It is usually recommended to do a parameter tuning of the algorithm on a small part of the data in advance in order to identify meaningful combinations. The number of repetitions should be set for stochastic algorithms. For all algorithms, we propose a sub-sampling of the data with around 80 percent of rows and columns to add some variation and to correct for outliers.

3.1.2 Combination Step

After calculating all N bicluster sets using the settings of the initialization step, the individual bicluster must be combined. One must first decide how many (n) of the bicluster from each set to use (If it is possible to set a maximum number in advance, this should be done in the initialization step). Each bicluster BC_i is compared to all other bicluster BC_j using a similarity measure. We propose two different similarity measures: Jaccards index and a correlations based approach. Both methods need thresholds to determine whether the bicluster are similar or not. As a result, an upper triangle matrix (similarity matrix) is retrieved. Groups of bicluster are then formed using one of the following algorithms.

Hierarchical Clustering

The obvious method for building groups of a similarity matrix is hierarchical clustering. Hanczar and Nadif (2010) suggest a hierarchical clustering of the similarity matrix using average linkage. Since the bicluster within a bicluster group should all have a similarity larger than the threshold, we propose using a complete linkage for the hierarchical approach. To obtain such groups, the hierarchical tree is pruned at the set threshold.

Quality Clustering

Alternatively, we suggest using a quality clustering (Heyer et al., 1999) of the similarity matrix. This is done by looking for the largest group with a similarity measure over the given threshold. This group is then deleted from the similarity matrix and the largest group in the remaining matrix is identified. This process continues until the similarity matrix is empty.

Scharl and Leisch (2006) showed that the largest group does not always lead to the best result. So instead the largest group, a random group is chosen using the size of the groups as weights. This approach reduces the possibility of combining two result bicluster into one group.

Similarity Measures

An important task in every segmentation or grouping is the selection of the similarity measure. We suggest two different methods: a correlation based approach and the Jaccard index.

Jaccard Index Approach The first method, which considers a bicluster as a subset, is based on the Jaccard index described in section 2.3. Two bicluster with one hundred percent similarity have got a Jaccard index of 1 and two bicluster with no equal elements have a Jaccard index of 0. After computing the Jaccard index for all bicluster combinations, the elements jac_{ij} of the matrix JAC represent the Jaccard index between bicluster BC_i and BC_j and can be compared with a threshold value t_{JAC}. In other words, two bicluster BC_i and BC_j are marked as similar when:

$$jac_{ij} \geq t_{JAC} \tag{3.1}$$

In order to decide which bicluster are similar, one must consider an appropriate threshold value. Since the correlation approach described in the next section suggests a divergence in each dimension of about 5% - which is in fact equal to

a similarity in elements of around $0.95 * 0.95 = 0.9025 \approx 90\%$ - the threshold for this approach could be set to 0.9.

Correlation Approach As an alternative, we propose a new method based on the correlation between the bicluster. More precisely, it focuses on the separate correlations between each pairwise row- and column-membership vector combination of the bicluster.

In other words two correlation matrices R^{Cor} and C^{Cor} are computed. The elements of these matrices (r_{ij}^{Cor} and c_{ij}^{Cor}) are correlation between the vectors $X^{(j)}$ and $X^{(i]}$ and the vectors $Y^{(j)}$ and $Y^{(i)}$ respectively.

$$r_{ij}^{Cor} = cor(X^{(i)}, X^{(j)}), c_{ij}^{Cor} = cor(Y^{(i)}, Y^{(j)}) \tag{3.2}$$

$Y^{(z)}$ ($Y^{(z)}$) is a binary representation of the rows (columns) included in bicluster BC_z. So $X^{(z)}$ ($Y^{(z)}$) at position l if row (column) l is in bicluster BC_z. Since the vectors are binary, the correlation had to be calculated with the Φ-coefficient. However, the Φ-coefficient is in fact equal to the Pearson correlation when applied on two binary variables. Two bicluster should not only be marked as similar with a match of a hundred percent (correlation of 1) but also with a small variation in rows or columns. One must find the smallest value at which bicluster should be marked as similar. An adequate divergence in each dimension is, for example, 5%. Since correlation can not be equated with percentage divergence, one must determine which correlation threshold leads to the allowed tolerance desired. In most cases, the dimensions of the data matrix are extremely different; therefore we suggest selecting threshold values t_R and t_C for each dimension separately. Therefore two row or two column vectors, i and j are marked as similar when:

$$r_{ij}^{Cor} \geq t_R \tag{3.3}$$

$$c_{ij}^{Cor} \geq t_C \tag{3.4}$$

3.1.3 Result step

A result set is generated from the groups of bicluster from the combination step. First, the groups are ordered according to the number of bicluster within. The higher the number of bicluster , the more support for the group, which leads to more reliability. One can define this score as

$$S_h = \frac{x}{N}, \tag{3.5}$$

where x is the number of bicluster in group h, N is the number of different runs. Since each bicluster can only be found once per run, the score is defined in the interval $[0, 1]$.

Next the proportio of bicluster containing a row-column combination is calculated for each group. This proportion can be interpreted as the probability that a row-column combination belongs to the resulting bicluster. All row-column combinations with a probability higher than a given threshold (we propose a threshold of 0.2) are reported as resulting bicluster.

The entire algorithm can be found in Table 3.1.

The algorithm of the ensemble method:

1. Choose bicluster algorithms and parameter settings.
2. Choose sub sampling, bootstrapping and/or number of repetition.
3. Run algorithms with chosen parameters.
4. Store best n bicluster of all N bicluster sets as a list.
5. Calculate a similarity matrix from that list.
6. Perform a grouping method (e.g. Hierarchical clustering or Quality Clustering).
7. Sort all groups by the number of single bicluster they contain.
8. Form a bicluster from every group using
   (a) Add up all bicluster in a matrix. Matrix contains the number of bicluster a row column combination is included in.
   (b) A row column combination belongs to this bicluster if the value in the matrix exceeds a fraction of the group size.

   and report as a bicluster result set.

Tab. 3.1: The algorithm of the ensemble method

3.2 Data Examples

To demonstrate the effects of the ensemble method, we calculated bicluster on artificial and gene expression data. Since the Plaid algorithm uses stochastic starting points and the results are very sensitive to parameter modifications, we compared the results of our method with single runs.

3.2.1 Artificial Data

The artificial data is used to demonstrate the advantages of our method in terms of overlapping and stability.

We hid up to 4 bicluster in a 1000×500 data matrix. The background of the matrix was standard normal distributed while the 50×50 bicluster followed a normal distribution with mean value $bcm_i = 3$ and variance $bcv_i = 0.1$. A heatmap of the data with the bicluster ordered in the upper right corner can be found in figure 3.1.

Three of the bicluster had a 10 percent overlap. We then ran the original Plaid algorithm and our ensemble method on n generated data sets. For the combining method we used the hierarchical and quality clustering and the Jaccard index as the similarity measure. In each run we used bootstrap-, sub- and no sampling. To compare the results with the hidden bicluster again, the Jaccard index was calculated between the outcome and the results. The results of one run using no sampling scheme is shown in figure 3.1. One can clearly see that the ensemble method found the hidden cluster structure while the original algorithm had some problems due to the overlapping structure of the bicluster. Off all the Over all sampling methods, the quality clustering showed the best result, retrieving nearly all original bicluster. The hierarchical cluster method was slightly worse and the original algorithm was the worst. Table 3.2 shows the mean value and standard deviation of the Jaccard index values of $n = 100$ runs. Comparing the sampling methods, the sub sampling outperformed bootstrap samples and full data. This is not a surprise because using full data makes it difficult to detect the overlapping structure.

3.2.2 Real Data

To demonstrate the behavior of the ensemble method on real data, we ran the algorithm on two micro-array datasets. Since there was no true bicluster structure in this data, we could only compare the stability of the results and the properties of the resulting bicluster. We compare the outcomes of the different algorithms on a small set of yeast data were time allowed multiple

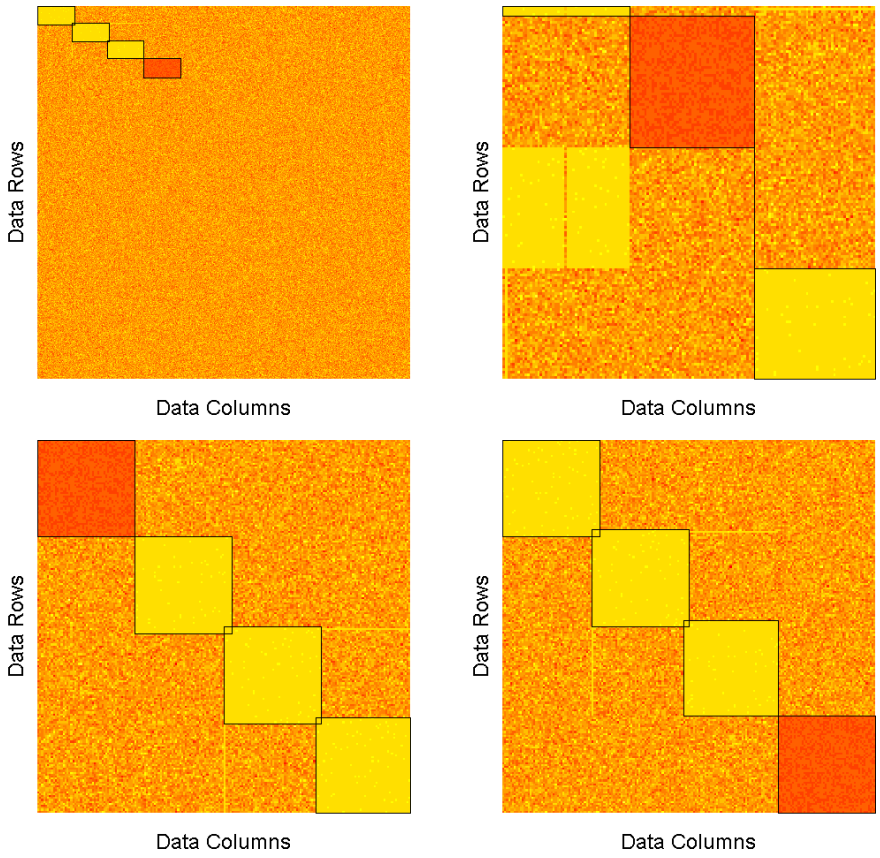

Fig. 3.1: Heatmap of bicluster results of one artificial data set: original data(top left), plaid (top right) and ensemble (hierarchical (bottom left) and quality clustering (bottom right))

	Original	Ensemble Quality Clustering	Ensemble Hierarchical Clustering
Mean	0.5633	0.9212	0.8571
Standard Deviation	0.37	0.14	0.17
	All Data	90% Subsampling	Bootstrap
Mean	0.8217	0.9705	0.8751
Standard Deviation	0.15	0.08	0.18

Tab. 3.2: Table of mean values and standard deviations of the Jaccard index using artificial data.

runs and show the results and differences of similarity measures on a large TCGA data set.

Yeast Data

This micro-array dataset used in Prelic et al. (2006) to present their bicluster technique is a sub sample of the Saccharomyces Cerevisiae organism (Yeast, containing 419 genes on 70 conditions. We again calculated bicluster with the original plaid algorithm and the ensemble method using both combining methods and the Jaccard index as a similarity measure. To compare stability, we ran the algorithm multiple times and compared the resulting bicluster sets with each other using the Jaccard index once again.

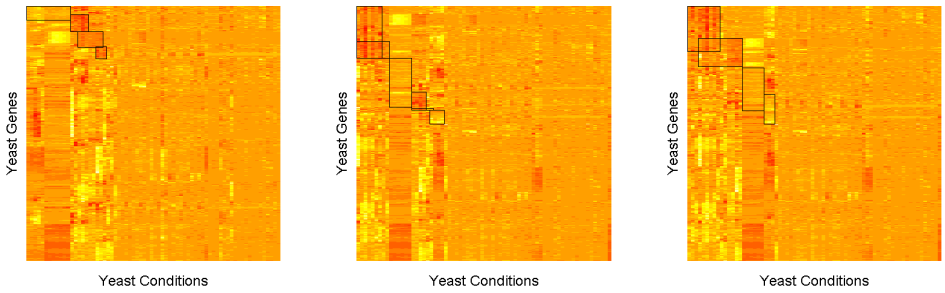

Fig. 3.2: Heatmap of bicluster results using original plaid algorithm (left) and the ensemble method with hierarchical (middle) and quality clustering (right) on the yeast data.

The bicluster results of the ensemble method showed the features we expected We found an overlapping structure and observed that the bicluster tend to be larger (For a visualization of one run see Figure 3.2).

	Original	Ensemble Quality Clustering	Ensemble Hierarchical Clustering
Mean	0.2688	0.7135	0.6755
Standard Deviation	0.10	0.15	0.14
	All Data	90% Subsampling	Bootstrap
Mean	0.7199	0.6691	NA
Standard Deviation	0.163	0.129	NA

Tab. 3.3: Table of mean values and standard deviations of the Jaccard index using yeast data.

The comparison of the runs also shows that the ensemble method is by far more stable than the original algorithm. Here it appears better to apply the algorithm on full data with a given number of repetitions rather than to use subsamples. This could be an effect of the small overlapping of bicluster in the result sets. Bootstrap samples do not actually work on real data, since the proportion of data point (around 2/3 of the full data) is too small to obtain

similar results. Table 3.3 shows the mean and standard deviation of the results compared within one method.

TCGA Data

To further demonstrate the effectiveness of our method, we additionally calculated bicluster on the TCGA data set (McLendon et al., 2008). All TCGA data are available for the public at the Data Coordinating Center (DCC) `http://tcga-data.nci.nih.gov`. The experiments were performed by the Broad Institute at MIT and Harvard using the Affymetrix (a manufacturer of DNA micro-arrays) micro-arrays in seven different institutes which are located throughout the United States. However, the TCGA data set we worked with had already been preprocessed by Nicholas D. Socci of the Computational Group of Memorial Sloan-Kettering Cancer Center (MSKCC - http://cbio.mskcc.org) in New York City.

The data consists of the RNA expression level of $n = 12042$ different human genes $G = (G_1, G_2, ..., G_n)$ and $m = 202$ samples. The vector $S = (S_1, S_2, ..., S_m)$ represents the different types of brain cancer (type C with 50 samples, M with 63 samples, N with 33 samples and P with 56 samples). The expression data was transformed with the natural logarithm, a common procedure when working with RNA data.

We applied the ensemble method to the data using the whole dataset in every run, varying only the `row.-` and `col.release` levels (from 0.51 up to 0.71 in steps of 0.02) and holding all other parameters constant (See Table 3.4 for the parameter setting). Both release levels were set equal in each run. Each model was computed 100 independent times. Altogether there were $T = 6567$ bicluster found.

Parameter	value
`cluster`	`"b"`
`fit.model`	`y ~ m + a + b`
`background`	`TRUE`
`shuffle`	`3`
`back.fit`	`0`
`max.layers`	`100`
`iter.startup`	`15`
`iter.layer`	`30`

Tab. 3.4: Used parameter settings for the ensemble method.

To demonstrate the differences between the similarity measures, we applied both measures to the 6567 bicluster.

Correlation Approach

First of all, one has to choose threshold values for the row- and column-correlation matrices. Due to the extremely different row and column sizes of the expression matrix, a different threshold should be chosen for each dimension. Table 3.5 shows examples of different threshold values and their corresponding allowed tolerance in rows and columns respectively. Obviously, since small thresholds allow too much variation in large bicluster and large thresholds allow too little variation in small bicluster there is no perfect threshold for all situations. Ideally threshold values should depend on the size of the expression matrix as well as on the size of the bicluster. However this topic requires further study; for the moment it is necessary to find a balance between the two extremes. As seen in Table 3.5, the row threshold was set to 0.95 and the column threshold to 0.9 since those two values allow the proposed divergence in each dimension of about 5%. That means that row vectors with a correlation greater than 0.95 and column vectors with a correlation greater than 0.9 are marked as similar. Thus, one is able to obtain the number of similar bicluster for each of the 6567 obtained bicluster.

	gene threshold			
size	0.8	0.85	0.9	0.95
25	0.2	0.16	0.08	0.04
50	0.2	0.14	0.1	0.04
100	0.2	0.14	0.1	0.05
150	0.2	0.14	0.1	0.05
200	0.2	0.15	0.1	0.05
400	0.2	0.15	0.1	0.05

	sample threshold			
size	0.8	0.85	0.9	0.95
25	0.16	0.12	0.07	0.04
30	0.16	0.14	0.06	0.03
40	0.15	0.13	0.06	0.02
50	0.14	0.12	0.07	0.03
80	0.13	0.09	0.05	0.02
100	0.1	0.08	0.04	0.03

Tab. 3.5: The table shows the allowed approximate percentage tolerance in genes, respectively samples depending on the correlation thresholds and bicluster sizes. Due to the different row ($length = 12042$) and column ($length = 202$) sizes of the expression matrix the bicluster sizes are also different in each dimension. Based on this values the row threshold was set to 0.95 and the column threshold to 0.9, which allows an variation in each dimension of around 5%.

Using the quality clustering without sampling weights, we calculated the similar bicluster (Shown in Figure 3.3). In fact, the majority of the bicluster was found just once or several times. We set the support to the upper 25%-quantile of similar bicluster. This allowed us to obtain a bicluster set of 58 remaining bicluster groups which we consider to be the real underlying bicluster in the data. The size of the bicluster varied in the gene dimension between 2 and 450 ($median = 139$; $mean = 153.6$) and in the sample dimension from 28 to

52 ($median = 41$; $mean = 39.3$). The bicluster were found between 25 and 94 times ($median = 41$; $mean = 35.9$). In total, 8909 genes were included in any bicluster, 1026 of which were unique. This makes a unique-gene-rate (# unique genes /# total genes) of 11.52%. Fifteen different genes were included in 29 different bicluster. The distribution of the bicluster scores is shown in Figure 3.5. The positive skewed distribution indicates once again, that there are some bicluster which seem more reliable than the rest.These have a higher score, because they were found more often.

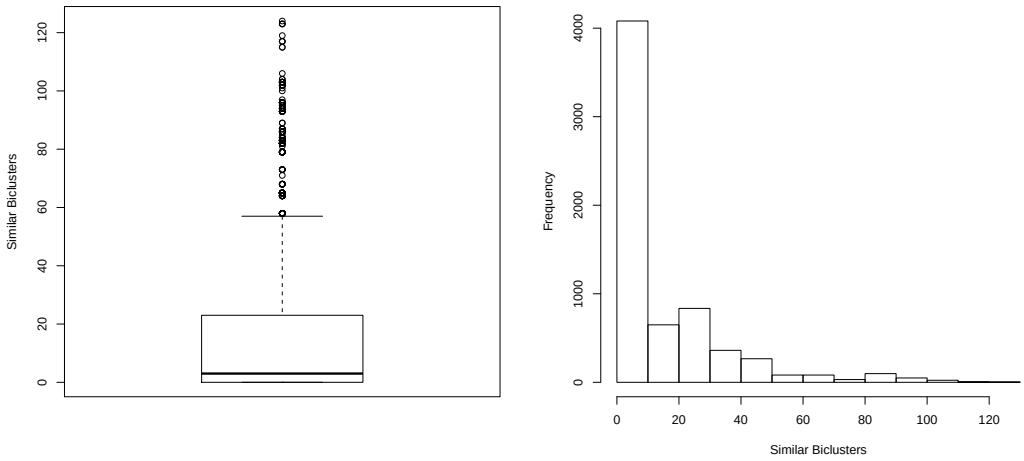

Fig. 3.3: Number of bicluster marked as similar for each observed bicluster. $min = 0$; $25\% - quantile = 0$; $median = 3$; $mean = 13.74$; $75\% - quantile = 23$; $max = 124$

Jaccard Index Approach

The threshold for the Jaccard Index was set to 0.9, as it allows nearly the same divergence between two bicluster as with the correlation approach. Thus bicluster with a Jaccard Index greater than 0.9 were marked as similar. The quantity of similar bicluster within this threshold is shown in Figure 3.4. The extremely positively skewed distribution again implies that some bicluster were found more often.

Again, only the upper $25\% - quantile$ of biclusters were kept, which leads to 63 remaining bicluster. The size of the bicluster varied in the gene dimension between 2 and 443 ($median = 141$; $mean = 147.8$) and in the sample dimension from 28 to 52 ($median = 41$; $mean = 40.4$), which is in fact quite similar to the results obtained from the correlation approach. The bicluster

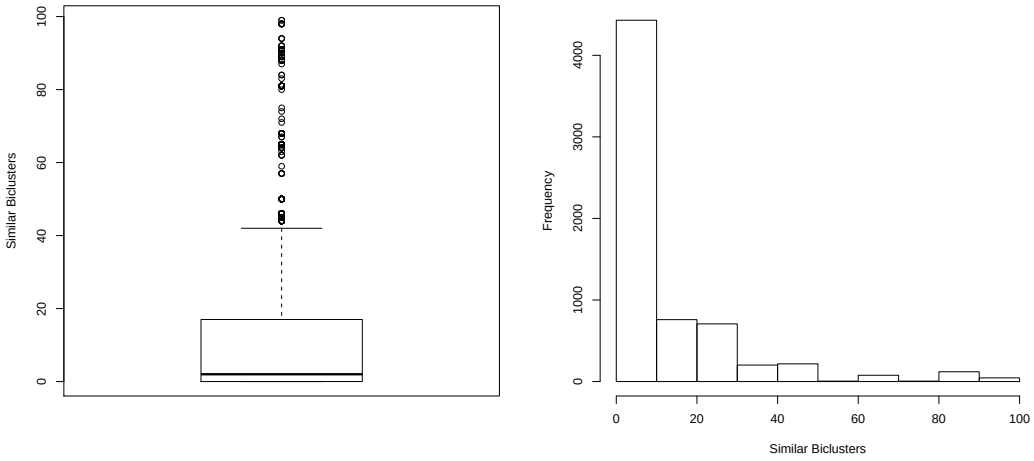

Fig. 3.4: Number of bicluster marked as similar for each observed bicluster. $min = 0$; $25\% - quantile = 0$; $median = 2$; $mean = 11.14$; $75\% - quantile = 17$; $max = 99$

were found between 19 and 89 times ($median = 25$; $mean = 29.60$). Furthermore the unique-gene-rate mentioned in the last section is also nearly the same (10.88%). The bicluster-score distribution can be found in Figure 3.5. A comparison of the scores obtain with the two methods reveals a quite similar, though shifted distribution.

Results in Comparison

Since both of the above described methods aim to obtain the best bicluster, it is important to know whether they truly lead to the same results, that is to the same bicluster. In order to determine the similarity between the two results, the bicluster were again compared with the Jaccard Index and a threshold value of 0.9.

In total 43 bicluster were found using each method. This makes for a similarity of 68.25% with reference to the results obtained by the Jaccard index approach, and a similarity of 74.14% with reference to the correlation approach results. Bicluster which were not observed with both methods had an average score of 0.021 (Jaccard Index approach), thus beneath the median (0.023) and the mean (0.027) total score. In contrast, the average score of bicluster which were obtained with each method is with 0.029 even above the $75\% - quantile$ (0.028) of the total score. In other words, our proposed score seems to provide information regarding the quality and especially the reliability of bicluster. Figure 3.6 shows a comparison of the two different score distributions, which

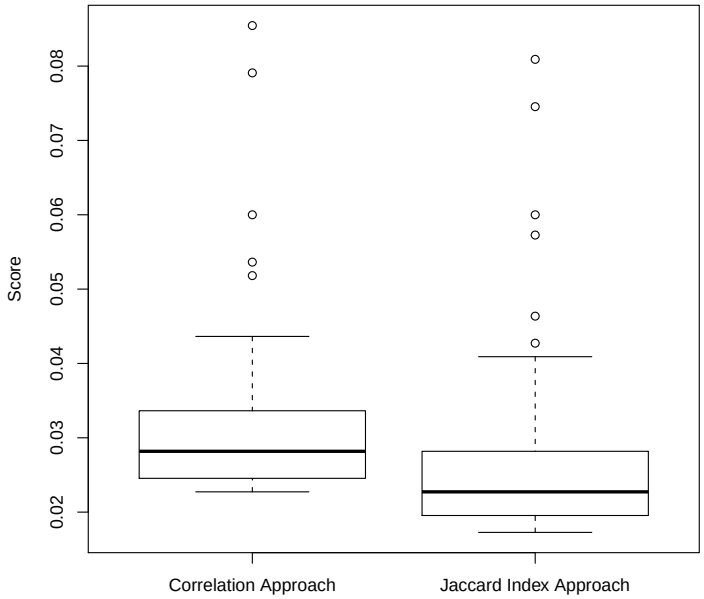

Fig. 3.5: Distribution of the bicluster scores. Correlation approach: $min = 0.023$; $25\% - quantile = 0.025$; $median = 0.028$; $mean = 0.033$; $75\% - quantile = 0.034$; $max = 0.085$. Jaccard Index approach: $min = 0.017$; $25\% - quantile = 0.020$; $median = 0.023$; $mean = 0.027$; $75\% - quantile = 0.028$; $max = 0.81$.

indeed indicates that bicluster found with both methods have a higher score than bicluster just found with one method.

Based on the significant overlap of the result it can be concluded, that both methods appear to have marked the same bicluster as the best. Secondly the score contains information about the quality of the bicluster in question. However, these analysis do not allow us to draw a conclusion about which method is more effective or powerful.

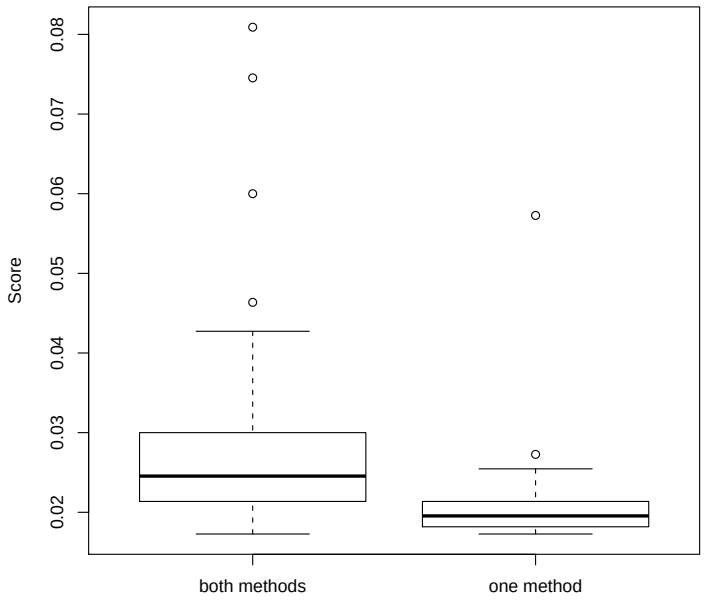

Fig. 3.6: Score of bicluster observed with only one method: $min = 0.017$; $25\% - quantile = 0.018$; $median = 0.019$; $mean = 0.022$; $75\% - quantile = 0.021$; $max = 0.057$. Observed with both methods: $min = 0.017$; $25\% - quantile = 0.021$; $median = 0.025$; $mean = 0.029$; $75\% - quantile = 0.030$; $max = 0.81$.

3.3 Conclusion

In this chapter, we proposed an ensemble method for bicluster analysis. This method takes results from different bicluster algorithm runs with modified parameters and/or a sub sampling of the data and combines the single bicluster obtained from that run. To combine the results, one must define a similarity measure. We proposed two similarity measures: Jaccard index and row/column correlations. The combined result (groups of bicluster) were then used to form a result set. Using this method,more stable results and otherwise undetectable overlapping bicluster can be detected. These advantages were demonstrated on simulated data. Additionally a real data example tested the method on a high dimensional data sets. Therefore, this method - which is applicable to every bicluster algorithm- allows one to obtain more stable and reliable results. Furthermore, overlapping bicluster structures can be detected even using algorithms which report only single bicluster.

4. Correlated Ordinal Data

A common method for testing a statistical model is the use of artificial data. A desired set of properties will be embedded in the dataset and then fitted models will be checked for the presence of these effects or how they behave under different experimental conditions. The Quest algorithm presented in Chapter 2 is able to find bicluster in ordinal data. For testing this method it is esential to have a tool at hand which delivers correlated ordinal values.

4.1 Introduction

The generation of arbitrary multivariate normal random numbers is straightforward: draw values from the standard normal distribution, multiply with an appropriate root of the desired covariance matrix, and add the mean. Other distributions mostly call for more complicated solutions, because linear combinations in most cases do not preserve the type of distribution. Sampling count variables with a given correlation is described in Erhardt and Czado (2010). For correlated binary data numerous methods have been proposed. For example Leisch et al. (1998) convert the desired covariance matrix for the binary data into a correlation matrix for normally distributed data. Therefrom normally distributed random numbers are drawn and binarised afterwards. For ordinal values only few suggestions can be found. Gange (1995) uses an iterative proportional fitting algorithm with pre specified probability structure of the marginals and pairwise and higher associations. Because of these higher order associations it becomes unpractical for large number of categories or variables. The method by Yu and Yuan (2004) works only for ordinal longitudinal data and needs an underlying regression model. Even more restrictions like independent and identical distribution among the variables are necessary for the method of Biswas (2004). A more general solution can be found in Demirtas (2006). His method relies on simulated binary variates as an intermediate step. Ordinal values are collapsed into binary ones, then corresponding binary correlations are computed in a way that ensures that reconversion delivers the original distribution properties. The first techniques (called binary conversion) proposed in the following is similar to the Demirtas (2006) approach,

but has fewer restrictions on the kind of correlations used. Also an alternative approach will be presented which outperforms the binary conversion in many situations and is suitable in more situations.

In what follows we give an introduction to the generation of correlated multivariate binary variates following Leisch et al. (1998). In Section 3 two techniques for generating multivariate ordinal variates are proposed. Section 4 shows some examples and compares the performanceces of the methods, and in the end we will give some concluding remarks.

4.2 Generation of Correlated Binary Random Variates

In this section we deal with variables which take only binary values, typically encoded by $\{0, 1\}$, and denoted by $A, B, \ldots$ or $A_1, A_2, \ldots$, respectively. Realizations of these random variables will be denoted by corresponding lower case letters. The distribution of a single variable A is fully determined by the value $p_A := \mathbb{P}(A = 1)$, which is also the expectation of A, i.e., $\mathbb{E}A = p_A$. The variance is given by $\mathrm{Var}(A) = p_A(1 - p_A)$.

Consider two binary random variables A and B which are not necessarily independent. Then the joint distribution of A and B is fully determined by p_A, p_B and either p_{AB}, $p_{A|B}$ or $p_{B|A}$ where

$$\begin{aligned} p_{AB} &:= \mathbb{P}(A = 1, B = 1) \\ p_{A|B} &:= \mathbb{P}(A = 1|B = 1) \\ p_{B|A} &:= \mathbb{P}(B = 1|A = 1) \end{aligned}$$

The remaining probabilities can easily be derived from Bayes Theorem.

This bivariate binary distribution can easily be generalized to the multivariate case, where $A = (A_1, \ldots, A_d)' \in \{0, 1\}^d$ is a vector with (possibly dependent) binary components. For a full description of an unrestricted distribution of A we need $2^d - 1$ parameters, e.g., the probabilities of all 2^d possible values of A (the last probability is determined by the condition that the sum equals 1).

A computationally fast method for generating samples from a binary vector $A = (A_1, \ldots, A_d)$ is the following: Let $X = (X_1, \ldots, X_d)$ be a d-dimensional normally distributed vector with mean μ and covariance matrix Σ. Normally distributed random variates can easily be transformed to binary values by componentwise thresholding: $a_i = 1 \Longleftrightarrow x_i > 0$. Due to the construction

$$p_{A_i} = \mathbb{P}(A_i = 1) = \mathbb{P}(X_i > 0)$$

and

$$p_{A_iA_j} = \mathbb{P}(A_i = 1, A_j = 1) = \mathbb{P}(X_i > 0, X_j > 0),$$

where $\mathbb{P}(X_i > 0)$ depends, for fixed variances, only on μ_i whereas $\mathbb{P}(X_i > 0, X_j > 0)$ depends on μ_i, μ_j and on the correlation between X_i and X_j.

Let Y_i be a 1-dimensional normally distributed random variable with mean μ_i and unit variance. Hence,

$$\mathbb{P}(Y_i > 0) = \mathbb{P}((Y_i - \mu_i) > -\mu_i) = \mathbb{P}((Y_i - \mu_i) \leq \mu_i)$$

where the second equality holds, because $(Y_i - \mu_i)$ is normally distributed with zero mean. If we choose μ_i to be the p_{A_i}-quantile of the standard normal distribution and restrict all variances to 1, then $\mathbb{P}(Y_i > 0) = p_{A_i}$. The mean vector μ is determined by the desired marginal probabilities p_{A_i} for the components of A.

What is still missing is a relation between the covariance matrix Σ_b of the binary variables and the covariance matrix Σ of the normal distribution. By specifying a covariance matrix only pairwise relations between the components of the d-dimensional sample can be specified. In the following we will restrict ourself to the bivariate case for ease of notation.

The correlation coefficient r_{AB} of two binary random variables A and B can be written as

$$r_{AB} = \frac{p_{AB} - p_A p_B}{\sqrt{p_A(1 - p_A)p_B(1 - p_B)}} \tag{4.1}$$

such that

$$p_{AB} = r_{AB}\sqrt{p_A(1 - p_A)p_B(1 - p_B)} + p_A p_B. \tag{4.2}$$

If A and B are converted from two normal random variables X and Y as described above, then p_{AB} can be related to the normal distribution by

$$p_{AB} = \mathbb{P}(X > 0, Y > 0) = \mathbb{P}(\bar{X} > -\mu_X, \bar{Y} > -\mu_Y) = L(-\mu_X, -\mu_Y, \rho),$$

where $\bar{X} := X - \mu_X$ and $\bar{Y} := Y - \mu_Y$ have a standard bivariate normal distribution with correlation coefficient $\rho = \rho_{XY}$; and

$$L(h, k, \rho) := \mathbb{P}(\bar{X} \geq h, \bar{Y} \geq k) = \int_h^\infty \int_k^\infty \phi(x, y; \rho) dy dx$$

with

$$\phi(x, y; \rho) = \frac{1}{2\pi\sqrt{1 - \rho^2}} \exp\left(-\frac{x^2 - 2\rho xy + y^2}{2(1 - \rho^2)}\right)$$

being the density function of $(\bar{X}, \bar{Y})$.

The values of $L(h, k, \rho)$ are tabulated (see the references in Patel and Read, 1982, p. 293f) or can be obtained by numerical integration or Monte Carlo

simulation (Leisch et al., 2009). The complete algorithm is summarized in Table 4.2.

Note that not every positive definite matrix is a valid covariance matrix for binary data. So some conditions on the common probabilities and therefore on the correlation matrix should be checked before the algorithm draws random numbers. The conditions, besides $0 \leq p_{A_i} \leq 1$, are

$$\max(p_{A_i} + p_{A_j} - 1, 0) \leq p_{A_i A_j} \leq \min(p_{A_i}, p_{A_j}) \qquad i \neq j$$

and

$$p_{A_i} + p_{A_j} + p_{A_k} - p_{A_i A_j} - p_{A_i A_k} - p_{A_j A_k} \leq 1 \qquad , i \neq j,\ i \neq k,\ j \neq k.$$

These conditions are necessary but not sufficient for $d \leq 3$.

4.3 Generation of Correlated Ordinal Random Variates

Without loss of generality we want to generate ordinal variables A taking integer values $\{1, 2, \ldots, k\}$. The corresponding distribution is defined by probability vector

$$p_A = \begin{pmatrix} \mathbb{P}(A = 1) \\ \mathbb{P}(A = 2) \\ \vdots \\ \mathbb{P}(A = k) \end{pmatrix} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_k \end{pmatrix},$$

for notational reasons we also need the distribution function

$$f_A(a) = \begin{cases} p_1 & , \quad a = 1 \\ p_2 & , \quad a = 2 \\ \vdots & , \quad \vdots \\ p_k & , \quad a = k \end{cases}.$$

When generating random numbers for d ordinal variables $A_1, \ldots, A_d$ the user needs to specify the marginal probabilities p_{A_i}, $i = 1, \ldots, d$ and a positive semi-definite correlation matrix

$$\boldsymbol{C} = \begin{pmatrix} \mathrm{Cor}(A_1, A_1) & \mathrm{Cor}(A_1, A_2) & \ldots & \mathrm{Cor}(A_1, A_d) \\ \mathrm{Cor}(A_2, A_1) & \mathrm{Cor}(A_2, A_2) & \ldots & \mathrm{Cor}(A_2, A_d) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cor}(A_d, A_1) & \mathrm{Cor}(A_d, A_2) & \ldots & \mathrm{Cor}(A_d, A_d) \end{pmatrix}.$$

Higher order interactions will not be taken into account. Note that because we use $\{1, 2, \ldots, k\}$ as possible values, observed values correspond to ranks and Pearson and Spearman correlation are identical (only the latter makes sense for ordinal data in general).

In the case of binary random variates the region were two variables simultaneously equal one determines the correlation between them. There is a direct link between their common probabilities and their correlation. With more than two categories this region is not that clear cut. The correlation now rather depends on other regions i.e. the common probabilities $\mathbb{P}(A = a, B = b)$ $a = 1, ..., k_A$ $b = 1, ..., k_B$ as well. Considering this, two randomization methods that allow specification of means and correlations will be presented in this section.

4.3.1 The Binary Conversion Method

Demirtas (2006) used a simple splitting rule to convert the ordinal variables into binary variables. The lower half of the categories is represented by the binary 0 and the upper half by binary 1. A simulation study is carried out every time new random variables are drawn to identify the binary correlations. In the following we show a closed form solution for a very similar algorithm. The main idea is to draw binary random variables with the correct correlation structure, and conditional on the outcome of the binary variable convert an independent uniform random to an ordinal variable with the desired marginals and correlations.

Let $\tilde{A} := \dfrac{A-1}{k-1}$ denote a linear transformation of A to new outcome values $0, \frac{1}{k}, \ldots, \frac{k-1}{k}$. The expectation is given by

$$\mathbb{E}(\tilde{A}) = \sum_{a=1}^{k} \frac{a-1}{k-1} p_a \tag{4.3}$$

We also define a new binary variable A_b with distribution

$$f(A_b) := \begin{cases} 1 - \mathbb{E}(\tilde{A}) & , \quad A_b = 0 \\ \mathbb{E}(\tilde{A}) & , \quad A_b = 1 \end{cases}$$

such that $\mathbb{E}(\tilde{A}) = \mathbb{E}(A_b)$. In addition we get

$$\begin{aligned} \mathbb{E}(\tilde{A}\tilde{B}) &= \sum_{a=1}^{k_{\tilde{A}}} \sum_{b=1}^{k_{\tilde{B}}} \frac{a-1}{k_{\tilde{A}}-1} \frac{b-1}{k_{\tilde{B}}-1} \mathbb{P}(\tilde{A} = \frac{a-1}{k_{\tilde{A}}-1}, \tilde{B} = \frac{b-1}{k_{\tilde{B}}-1}) \\ &= \sum_{a=1}^{k_A} \sum_{b=1}^{k_B} \frac{a-1}{k_A-1} \frac{b-1}{k_B-1} \mathbb{P}(A = a, B = b) \\ &= \mathbb{P}(A_b = 1, B_b = 1) = \mathbb{E}(A_b B_b) \end{aligned}$$

and therefore

$$\begin{aligned} \mathrm{Cov}(\tilde{A},\tilde{B}) &= \mathbb{E}(\tilde{A}\tilde{B}) - \mathbb{E}(\tilde{A})\mathbb{E}(\tilde{B}) = \mathbb{E}(A_b B_b) - \mathbb{E}(A_b)\mathbb{E}(B_b) \\ &= \mathrm{Cov}(A_b, B_b). \end{aligned} \tag{4.4}$$

Using $\mathrm{Var}(A_b) = \mathbb{E}(\tilde{A})(1 - \mathbb{E}(\tilde{A}))$ we get

$$\mathrm{Var}(\tilde{A}) = \sum_{a=1}^{k} (\frac{a-1}{k-1} - \mathbb{E}(\tilde{A}))^2 p_a$$

and analogously for $\mathrm{Var}(\tilde{B})$. Due to the linearity of the conversion $\mathrm{Cor}(\tilde{A},\tilde{B}) = \mathrm{Cor}(A,B)$. The function that maps the desired correlation $\mathrm{Cor}(A,B)$ on the binarised correlation $\mathrm{Cor}(A_b,B_b)$ is a straight line passing through the origin and with slope m that depends only on the probability vectors p_A and p_B:

$$\mathrm{Cor}(\tilde{A},\tilde{B}) = \mathrm{Cor}(A,B) = m\mathrm{Cor}(A_b,B_b) \tag{4.5}$$

For four examples of probability vectors for two variables this is shown in Figure 4.1.

Combining (4.5) and (4.4) gives

$$\begin{aligned} m^{-1} &= \frac{\mathrm{Cor}(A_b,B_b)}{\mathrm{Cor}(\tilde{A},\tilde{B})} = \frac{\frac{\mathrm{Cov}(A_b,B_b)}{\sqrt{\mathrm{Var}(A_b)\mathrm{Var}(B_b)}}}{\frac{\mathrm{Cov}(\tilde{A},\tilde{B})}{\sqrt{\mathrm{Var}(\tilde{A})\mathrm{Var}(\tilde{B})}}} = \sqrt{\frac{\mathrm{Var}(\tilde{A})\mathrm{Var}(\tilde{B})}{\mathrm{Var}(A_b)\mathrm{Var}(B_b)}} \\ &= \sqrt{m_A m_B}, \end{aligned}$$

with $m_A = \mathrm{Var}(\tilde{A})/\mathrm{Var}(A_b)$ and $m_B = \mathrm{Var}(\tilde{B})/\mathrm{Var}(B_b)$.

Using

$$\begin{aligned} \sum_{a=1}^{k_A} (\frac{a-1}{k_A-1} - \mathbb{E}(\tilde{A}))^2 \mathbb{E}(\tilde{A}) &= \mathbb{E}(\tilde{A})(1-\mathbb{E}(\tilde{A})) + \sum_{a=1}^{k_A} (\frac{a-1}{k_A-1})^2 \mathbb{E}(\tilde{A}) - \mathbb{E}(\tilde{A}) \\ &= \sum_{a=1}^{k_A} (\frac{a-1}{k_A-1})^2 \mathbb{E}(\tilde{A}) - 2\mathbb{E}(\tilde{A})^2 + \mathbb{E}(\tilde{A})^2 \\ &= -\mathbb{E}(\tilde{A})^2 + \sum_{a=1}^{k_A} (\frac{a-1}{k_A-1})^2 \mathbb{E}(\tilde{A}) \end{aligned}$$

we get

$$\mathrm{Var}(\tilde{A}) = \mathrm{Var}(A_b) + \mathbb{E}(\tilde{A}^2) - \mathbb{E}(A_b).$$

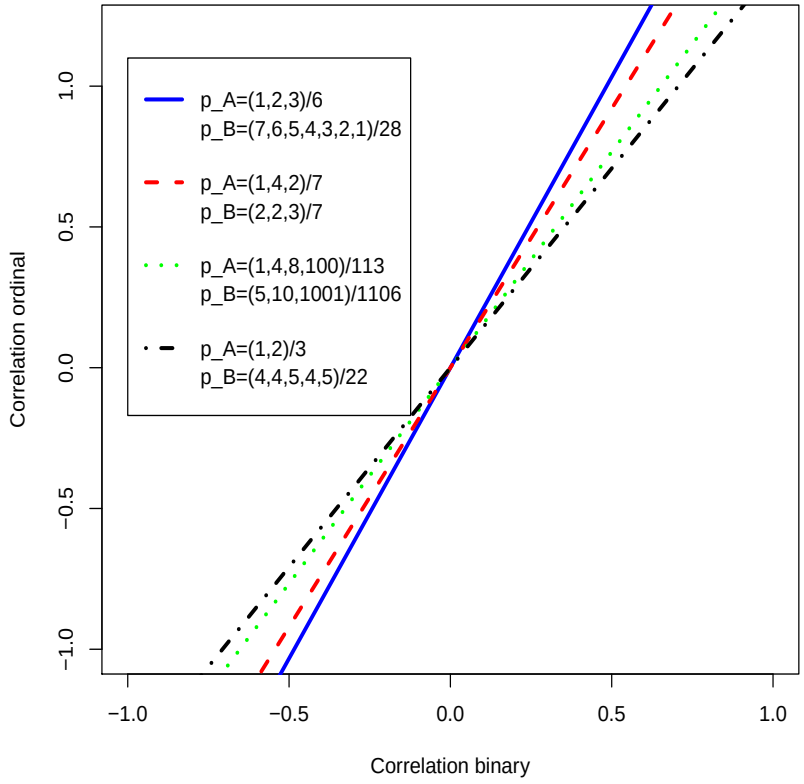

Fig. 4.1: Linear transformation functions. The m-factors to translate ordinal correlation specifications to binary correlations.

The conditional distribution of A given A_b is

$$f(A|A_b) = \begin{cases} f(A|A_b = 0) =: f_0(A) \\ f(A|A_b = 1) =: f_1(A) \end{cases}$$

For $A_b = 1$ the conditional distribution $f_1(A)$ is simply

$$f_1(A) = \frac{\frac{a-1}{k-1} p_a}{\mathbb{E}(\tilde{A})} = \frac{(a-1)p_a}{\sum_{l=2}^{k}(l-1)p_l},$$

for $A_b = 0$ we can use

$$\begin{aligned}\mathbb{P}(A_b = 0) &= 1 - \mathbb{E}(\tilde{A}) \\ &= 1 - \sum_{a=1}^{k} \frac{a-1}{k-1} p_a = 1 - \frac{1}{k-1}(\mathbb{E}(A) - 1) \\ &= \frac{k - \mathbb{E}(A)}{k-1} = \frac{\sum_{a=1}^{k} k p_a - \sum_{a=1}^{k} a p_a}{k-1} = \\ &= \sum_{a=1}^{k} \frac{k-a}{k-1} p_a = \sum_{a=1}^{k-1} \frac{k-a}{k-1} p_a,\end{aligned}$$

to obtain

$$f_0(A) = \frac{\frac{k-a}{k-1} p_a}{1 - \mathbb{E}(\tilde{A})} = \frac{(k-a)p_a}{\sum_{l=1}^{k-1}(k-l)p_l}.$$

The resulting cumulative distribution functions are therefore

$$F_0(A) = \frac{\sum_{l=1}^{a} \frac{k-l}{k-1} p_l}{1 - \mathbb{E}(\tilde{A})}$$

and

$$F_1(A) = \frac{\sum_{l=2}^{a} \frac{l-1}{k-1} p_l}{\mathbb{E}(\tilde{A})}.$$

The final algorithm is to draw binary variables A_b with a certain correlation structure. In addition we independently draw from the uniform distribution $U(0,1)$ and use the inversion method with $F_1(A)$ and $F_0(A)$ to obtain ordinal values. The binary variables A_b shift the distribution of A to the left or right to get correlations, the particular choice of A_b guarantees that the overall marginal probabilities are still correct. The whole algorithm is summarized in Table 4.3

Figure 4.1 shows that not all correlations can be calculated because the binary correlations are restricted to $[-1, 1]$. Hence, the correlation range of the algorithm is smaller than that of the method in Demirtas (2006). But while they use simulation runs we have an analytical solution for the transformation which leads to far shorter run times. Since range may be more important than speed, the next section gives an alternative approach with broader range.

4.3.2 The Mean Mapping Method

Our mean mapping method to generate ordinal random numbers with a given correlation structure generalizes the concepts of Leisch et al. (1998) from the

binary to the ordinal case. Let X again be a random variable with standard normal distribution $N(0,1)$. To get an ordinal variable with cumulative distribution F we cut X at the $F(a)$-quantiles q of the standard normal distribution:

$$\mathbb{P}(A = a) = \mathbb{P}(q_{F_A(a-1)} < X < q_{F_A(a)}) \qquad a = 1, \ldots, k_A \qquad X \sim N(0,1), \tag{4.6}$$

Figure 4.2 shows an example for $k = 4$ categories.

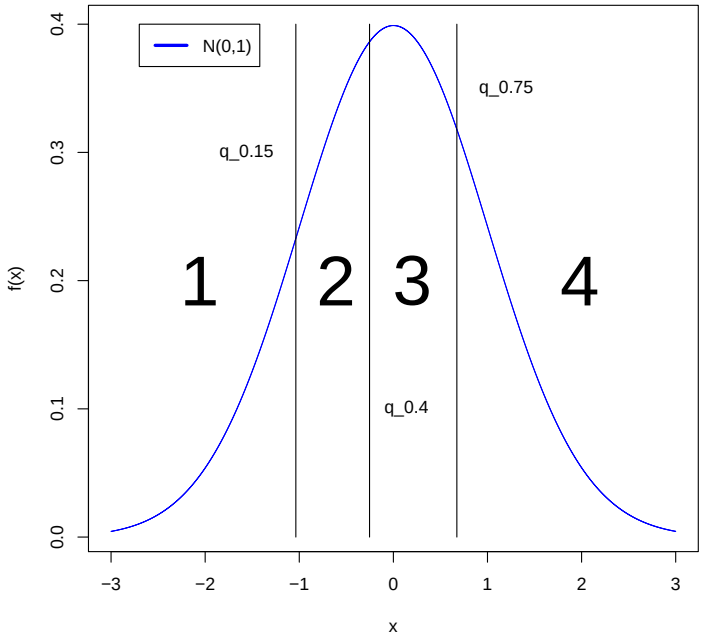

Fig. 4.2: Thresholding the normal distribution.
$\boldsymbol{p}_A = (0.15 \quad 0.25 \quad 0.35 \quad 0.25)^\mathsf{T} \Rightarrow \boldsymbol{q}_A \approx (-1.04 \quad -0.25 \quad 0.67 \quad +\infty)^\mathsf{T}$

Let A and B be two ordinal variables obtained by cutting X and Y, respectively. The joint probabilities can then be written as

$$\begin{aligned} &\mathbb{P}(A = a, B = b) && (4.7)\\ &= F_{AB}(a,b) - F_{AB}(a-1,b) - F_{AB}(a,b-1) + F_{AB}(a-1,b-1) \\ &= \Phi_{XY}(q_{F_A(a)}, q_{F_B(b)}, \rho_{XY}) - \Phi_{XY}(q_{F_A(a-1)}, q_{F_B(b)}, \rho_{XY}) && (4.8)\\ &\quad -\Phi_{XY}(q_{F_A(a)}, q_{F_B(b-1)}, \rho_{XY}) + \Phi_{XY}(q_{F_A(a-1)}, q_{F_B(b-1)}, \rho_{XY}) \end{aligned}$$

with q being a quantile of the univariate standard normal distribution and $\mathbb{P}(X < h, Y < k) = \Phi_{XY}(h, k, \rho_{XY})$ the bivariate standard normal distribution function with correlation coefficient ρ_{XY}. Equation (4.9) links probabilities $\mathbb{P}(A = a, B = b)$ to ρ_{XY}. For the binary case $\mathbb{P}(A = 1, B = 1) = \mathbb{E}(AB)$ defines the whole distribution. Hence, the natural generalization for the ordinal

case would be to evaluate the relationsship between $\mathbb{E}(AB)$ and ρ_{XY} on a regular grid and interpolate the results. For this we would need to specify the complete joint distribution of F_{AB}. By rearranging terms we can find a scalar (called τ below) which only depends on the marginal distribution of A and B and the desired correlation $\mathrm{Cor}(A, B)$.

The expectaion of AB is defined as

$$\begin{aligned}\mathbb{E}(AB) &= \sum_{a=1}^{k_A}\sum_{b=1}^{k_B} ab\,\mathbb{P}(A=a, B=b) \\ &= \sum_{a=1}^{k_A}\sum_{b=1}^{k_B} ab\,\big(F_{AB}(a,b) - F_{AB}(a-1,b) \\ &\quad -F_{AB}(a,b-1) + F_{AB}(a-1,b-1)\big) \\ &= \sum_{a=1}^{k_A}\sum_{b=1}^{k_B} m_{ab}F_{AB}(a,b). \end{aligned} \tag{4.9}$$

By simple algebra we get the multiplicities m_{ab} as

$$\begin{array}{lrlll} m_{ab} = ab - a(b+1) - (a+1)b + (a+1)(b+1) = & 1, & a < k_A & b < k_B \\ m_{ab} = a[b-(b+1)] = -a = & -k_A, & a = k_A & b < k_B \\ m_{ab} = b[a-(a+1)] = -b = & -k_B, & a < k_A & b = k_B \\ m_{ab} = ab = & k_A k_B, & a = k_A & b = k_B \end{array} \tag{4.10}$$

Combining Equations (4.9) and (4.10) gives

$$\begin{aligned}\mathbb{E}(AB) &= \sum_{a=1}^{k_A-1}\sum_{b=1}^{k_B-1} F_{AB}(a,b) - k_B\sum_{a=1}^{k_A-1} F_A(a) \\ &\quad -k_A\sum_{b=1}^{k_B-1} F_B(b) + k_A k_B.\end{aligned}$$

We use the first term of this equation as proxy τ which will be linked to ρ_{XY}. Rearranging terms in the usual definition of the correlation gives

$$\begin{aligned}\tau_{AB} &= \sum_{a=1}^{k_A-1}\sum_{b=1}^{k_B-1} F_{AB}(a,b) \\ &= \mathrm{Cor}(A,B)\sqrt{\mathrm{Var}(A)\mathrm{Var}(B)} + \mathbb{E}(A)\mathbb{E}(B) \\ &\quad -k_A k_B + k_A\sum_{b=1}^{k_B-1} F_B(b) + k_B\sum_{a=1}^{k_A-1} F_A(a),\end{aligned}$$

which depends only on the marginal distribution of A and B and correlation $\mathrm{Cor}(A, B)$. We now evaluate the relationship between ρ_{XY} and τ_{AB} on a regular grid and interpolate results. Inverting this relationsship gives the necassary

ρ_{XY} for given τ_{AB}. Drawing random numbers now amounts to drawing bivariate normal variates with zero mean, unit variance and correlation ρ_{XY}. These are then cut at quantiles defined by the marginal distributions of A and B, respectively. Generalization to more than two ordinal variates is again straightforward. The complete algorithm for the mean mapping method can be found in Table 4.4.

Feasible Correlation Matrices

Although wider ranges of correlations are possible within the mean mapping method there are some restriction to the correlation matrix. First of all the correlation range $[-1; 1]$ is not possible for most of the probability vectors. The maximal correlation is reached if all variables are ordered from the lowest category to the highest. Beside that the normal correlations are most of the time higher than the specified ordinal correlation, so if the correlations are to high they get greater 1 in the normal conversion and are so not feasible.

4.4 Simulation and Comparison

For comparison of the two methods we generated random ordinal values from both methods and compared the results with respect to runtime and precision. As the restrictions on the correlation matrix are stronger for the binary conversion method than for the mean mapping method, matrices are chosen which are feasible for both methods. As dimensions d and number of categories k we used 3, 6 and 9 in both cases. One million random values were drawn for each algorithm with each of the 9 setups.

4.4.1 Performance

The runtime of the algorithms is depicted in Figure 4.3. It can be seen that the runtime of the binary conversion method is very low even for the case with 9 variables and 9 categories. The runtime of the mean mapping method depends on both, the numbers of categories and the number of variables.

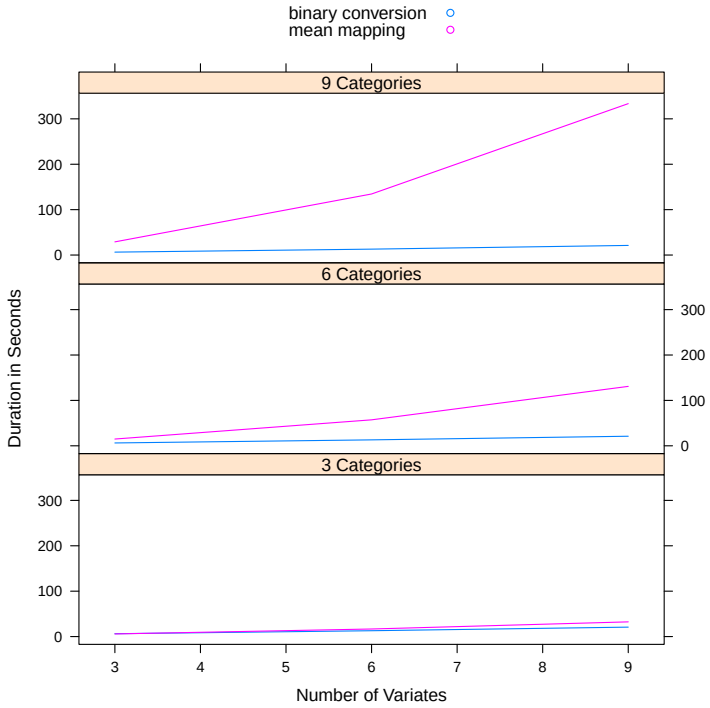

Fig. 4.3: Runtime of binary and mean mapping method.

4.4.2 Accuracy

Figures 4.4, 4.5 and 4.6 give information about how exact the methods generate random numbers. For this purpose the following quantities were calculated: *Average absolute distance of correlation matrix entries:*

$$\mu_C = \frac{1}{q^2} \sum_{i=1}^{q} \sum_{j=1}^{q} |\boldsymbol{C}_{[i,j]} - \hat{\boldsymbol{C}}_{[i,j]}|$$

Maximum absolute distance of correlation matrix entries:

$$m_C = \max_{i,j}(|\boldsymbol{C}_{[i,j]} - \hat{\boldsymbol{C}}_{[i,j]}|)$$

Average absolute distance of probability vector entries:

$$\mu_P = \sum_{i=1}^{q} \sum_{a_i=1}^{k_{A_i}} |\boldsymbol{P}_{[i,a_i]} - \hat{\boldsymbol{P}}_{[i,a_i]}|$$

with $\hat{\boldsymbol{C}}$ the empirical correlation matrix computed from the observed random numbers and $\hat{\boldsymbol{P}}$ relative frequencies of the cases computed from the observed random numbers.

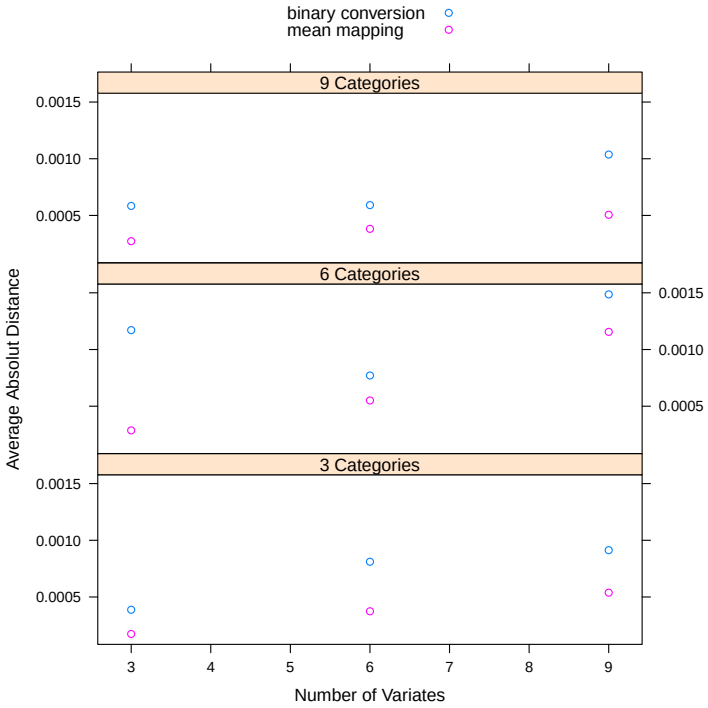

Fig. 4.4: Average absolute differences of sample and input correlations.

Figure 4.4 shows that all values for μ_C do not exceed 0.003 with the largest average distance being at $\mu_C = 0.002967$ which is a good result. One can also note that the mean mapping method is the numerically most stable. A similar result is indicated by figure 4.5 which presents the m_C values. Again both methods are similar, but the mean mapping method is better.

Figure 4.6 shows that both methods have similar low values for μ_P, which had to be expected because all methods use a categorization of the normal distribution which is analytically exact. One can also see that for more categories μ_P does slightly shrink, which is what we can expect due to the increased number of observations $\hat{\boldsymbol{P}}_{[i,a_i]}$ that enter the formula. Summarizing the results, μ_C, m_C and μ_P show that both methods have sufficient precision for most practical applications.

4.4.3 Comparison with Demirtas

In Demirtas (2006) different setups were use to show the flexibility of the algorithm. In this section we show that the mean mapping approach can cover all these setups and can also extend these setups to higher correlations. Table 4.1 contains two examples of correlation matrices which were used by

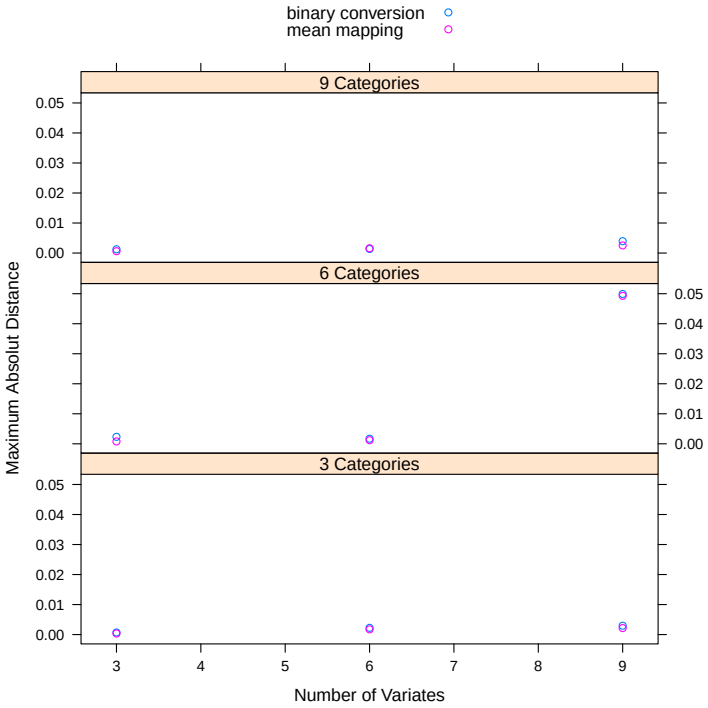

Fig. 4.5: Maximum absolute differences of sample and input correlations.

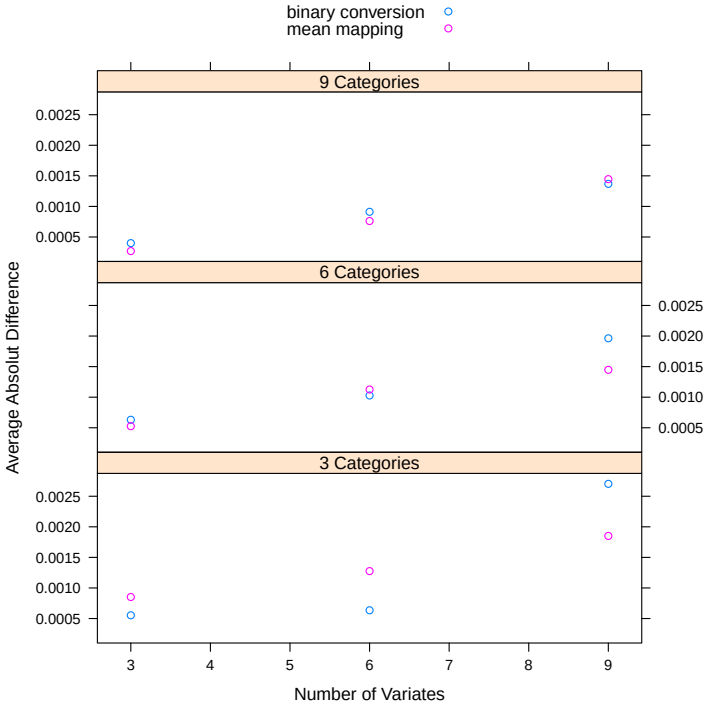

Fig. 4.6: Average absolute differences of sample and input probabilities.

Demirtas (2006) and a third matrix which is not feasible for his method. As marginal probabilities we used

$$P_{A1} = \begin{pmatrix} 0.05 \\ 0.25 \\ 0.55 \\ 0.15 \end{pmatrix}, p_{A2} = \begin{pmatrix} 0.10 \\ 0.10 \\ 0.10 \\ 0.70 \end{pmatrix}, p_{A3} = \begin{pmatrix} 0.20 \\ 0.15 \\ 0.25 \\ 0.40 \end{pmatrix}$$

	Cor	Mat	1	Cor	Mat	1	Cor	Mat	3	
	A1	A2	A3	A1	A2	A3	A1	A2	A3	
A1	1	0.4	0.3	1	0.5	0.25	1	0.7	0.7	A1
A2	0.4	1	0.4	0.5	1	0.5	0.7	1	0.7	A2
A3	0.3	0.4	1	0.25	0.5	1	0.7	0.7	1	A3

Tab. 4.1: Three example correlation matrices

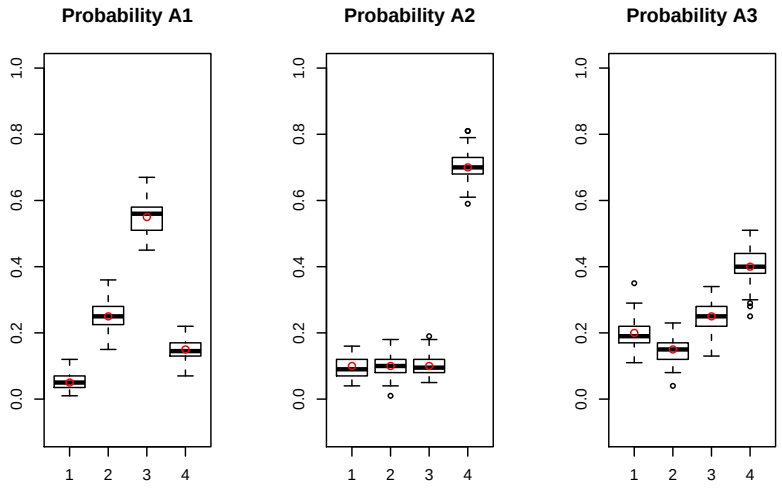

Fig. 4.7: Boxplot of frequency of 100 simulation runs with Correlation matrix 1. Red circles show input probabilities.

Figure 4.7 shows the frequencies of 100 simulation runs were 100 random ordinal variates were drawn. The red circles represent the desired values, which are close to the median of the observed values in each case. Figure 4.8 shows the three values of the upper triangle of the observed correlation matrices with the red circles again representing the desired correlation. It can be seen, that the algorithm works quite good in all three scenarios.

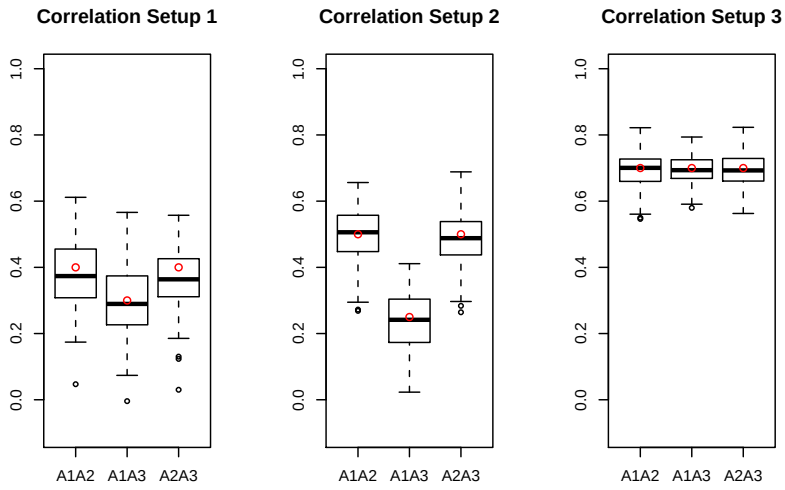

Fig. 4.8: Boxplot of correlations of 100 simulation runs. Red circles show input correlations.

4.5 Conclusions

In this chapter we presented two new methods for generating ordinal values with given correlation structure. The binary method is very fast but has the disadvantage that the set of feasible correlation matrices is limited by the algorithm. The mean mapping method overcomes this problem and is as accurate as the binary solution at the price of longer runtime. With the use of ordinal values in questionnaires it is essential for one of our simulation studies to have a tool at hand which provides us such values. But the method is not limited for our purpose and is also useful for testing of the growing number of statistical methods working on samples of ordinal values. A freely available open source implementation (`ordata` ,Kaiser and Leisch (2010) for the statistical computing environment R (R Development Core Team, 2011) is described in Appendix B.

Step	Expression
1	Calculate the probabilities $$h_{t_2,t_3}^{t_1} := \int_h^\infty \int_k^\infty \phi_{X_1X_2}(g_{t_2}, g_{t_3}, \boldsymbol{\Sigma}_{t_1})dxdy$$ with $(X_1, X_2) \sim N(\mathbf{0}, \boldsymbol{\Sigma}_{t_1})$ and covariance matrix $$\boldsymbol{\Sigma}_{t_1} = \begin{pmatrix} 1 & g_{t_1} \\ g_{t_1} & 1 \end{pmatrix}$$ where $g_{t_1} = \frac{t_1}{20}$ and $t_1 = -20, -19, \ldots, 20$, $g_{t_2} = \frac{t_2}{20}$ and $t_2 = 0, 1, \ldots, 20$ and $g_{t_3} = \frac{t_3}{20}$ and $t_3 = 0, 1, \ldots, 20$ receiving grid $\boldsymbol{g}_{1,2,3}$.
2	Fit a function $f_{h\|g_{t_2},g_{t_3}}(h) : h \to g_{t_1}$ to grid $\boldsymbol{g}_{1,2,3}$.
3	Set $$h^* = \text{Cor}(A_1, A_2)\sqrt{\text{Var}(A_1)\text{Var}(A_2)} + \mathbb{E}(A_1)\mathbb{E}(A_2)$$ and calculate the correlation coefficient $$f_{h\|g_{t_2},g_{t_3}}(h^*) = \text{Cor}(X_1, X_2).$$
4	Repeat step 3 for all combinations (i, j) of variables, receiving $$\boldsymbol{C}^{bin} = \begin{pmatrix} \text{Cor}(X_1, X_1) & \text{Cor}(X_1, X_2) & \ldots & \text{Cor}(X_1, X_d) \\ \text{Cor}(X_2, X_1) & \text{Cor}(X_2, X_2) & \ldots & \text{Cor}(X_2, X_d) \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cor}(X_d, X_1) & \text{Cor}(X_d, X_2) & \ldots & \text{Cor}(X_d, X_d) \end{pmatrix},$$ which is the multivariate normal correlation matrix.
5	Sample n times from the d-dimensional normal distribution with covariance matrix $\boldsymbol{C}^{bin}$ and mean vector $$\boldsymbol{\mu}^{bin} = \begin{pmatrix} -q_{p(A_1)} \\ -q_{p(A_2)} \\ \vdots \\ -q_{p(A_d)} \end{pmatrix}$$ receiving the $n \times d$ sample matrix $\boldsymbol{S}^{bin}$.
6	Reconvert $\boldsymbol{S}^{bin}$ to the ordinal sample matrix $\boldsymbol{S}$ using

	$\boldsymbol{S}_{[ei]} = \begin{cases} 1 & ,0 \leq \boldsymbol{S}^{bin}_{[ei]} \\ 0 & ,0 > \boldsymbol{S}^{bin}_{[ei]} \end{cases}$

Tab. 4.2: Generation of multivariate binary random numbers via Leisch et al. (1998)

Step	Statement
1	Calculate the weighted mean $\mu_1 := \mathbb{E}(A_1^b) = \sum_{a=1}^{k_1} \frac{a-1}{k_1-1} p_a$.
2	Calculate the binarised variance $\mathrm{Var}(A_1^b) = \mathbb{E}(\tilde{A})(1-\mathbb{E}(\tilde{A}))$.
3	Calculate the ordinal variance $\mathrm{Var}(\tilde{A}_1) = \sum_{a=1}^{k_1} (\frac{a-1}{k_1-1} - \mathbb{E}(\tilde{A}))^2 p_a$.
4	Calculate the slope $m_1 := \mathrm{Var}(\tilde{A}_1)/\mathrm{Var}(A_1^b)$.
5	Do steps 1-4 for each of the d variables receiving $$\boldsymbol{\mu}^b = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \vdots \\ \mu_d \end{pmatrix} \quad and \quad \boldsymbol{m} = \begin{pmatrix} m_1 \\ m_2 \\ m_3 \\ \vdots \\ m_d \end{pmatrix}.$$
6	Calculate the new binary correlation matrix via $$\mathrm{Cor}(A_i^b, A_j^b) = \begin{cases} \mathrm{Cor}(A_i, A_j)/\sqrt{m_i m_j} & , \quad i \neq j \\ 1 & , \quad i = j \end{cases}$$ getting $$\boldsymbol{C}^b = \begin{pmatrix} \mathrm{Cor}(A_1^b, A_1^b) & \mathrm{Cor}(A_1^b, A_2^b) & \dots & \mathrm{Cor}(A_1^b, A_d^b) \\ \mathrm{Cor}(A_2^b, A_1^b) & \mathrm{Cor}(A_2^b, A_2^b) & \dots & \mathrm{Cor}(A_2^b, A_d^b) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cor}(A_d^b, A_1^b) & \mathrm{Cor}(A_d^b, A_2^b) & \dots & \mathrm{Cor}(A_d^b, A_d^b) \end{pmatrix}$$
7	Sample n times from d-dimensional binary distribution with correlation matrix of the binary distribution $\boldsymbol{C}^b$ and mean vector $\boldsymbol{\mu}^b$ getting the $n \times d$ binary sample matrix $\boldsymbol{S}^b$
8	Draw from $U(0,1)$ md times receiving $m \times d$ matrix $\boldsymbol{U}$.
9	Reconvert $\boldsymbol{S}^b$ for each variable with originally more than two categories to the ordinal sample matrix $\boldsymbol{S}$ using the samples from 7 and 8 and the assignment $$\boldsymbol{S}_{[ei]} = k : \quad \begin{cases} F_0(k-1) < \boldsymbol{U}_{[ei]} < F_0(k) & , \quad if\ \boldsymbol{S}^b_{[ei]} = 0 \\ F_1(k-1) < \boldsymbol{U}_{[ei]} < F_1(k) & , \quad if\ \boldsymbol{S}^b_{[ei]} = 1 \end{cases}$$ for $k \in \{1, 2, \dots, k_i\}$ with cumulative distribution functions $$F_0(A) = \frac{\sum_{l=1}^{a} \frac{k-l}{k-1} p_l}{1 - \mathbb{E}(\tilde{A})}$$ and

	$$F_1(A) = \frac{\sum_{l=2}^{a} \frac{l-1}{k-1} p_l}{\mathbb{E}(\tilde{A})}$$
.	for each entry $\boldsymbol{S}_{[ei]}$ independently.

Tab. 4.3: Generating multivariate ordinal random numbers via binary conversion method

Step	Expression
1	Calculate the probability $$h_{1,2}^t := \sum_{a_1=1}^{k_1-1} \sum_{a_2=1}^{k_2-1} \boldsymbol{\Phi}_{X_1X_2}(q_{F_{A_1}(a_1)}, q_{F_{A_2}(a_2)})$$ with $(X_1, X_2) \sim N(\mathbf{0}, \boldsymbol{\Sigma}_t^{1,2})$ and covariance matrix $$\boldsymbol{\Sigma}_t^{1,2} = \begin{pmatrix} 1 & g_t \\ g_t & 1 \end{pmatrix}$$ where $g_t = \dfrac{t}{100}$ and $t = -100, \ldots, 100$, receiving grid $\boldsymbol{g}_{1,2}$.
2	Fit a function $f_{1,2}(h) : h \to g$ to grid $\boldsymbol{g}_{1,2}$.
3	Set $$h^* = \mathrm{Cor}(A_1, A_2)\sqrt{\mathrm{Var}(A_1)\mathrm{Var}(A_2)} + \mathbb{E}(A_1)\mathbb{E}(A_2) \\ -k_1k_2 + k_1 \sum_{a_2=1}^{k_2-1} F_{A_2}(a_2) + k_2 \sum_{a_1=1}^{k_1-1} F_{A_1}(a_1)$$ and calculate the correlation coefficient $$f_{1,2}(h^*) = \mathrm{Cor}(X_1, X_2).$$
4	Repeat steps 1-3 for all combinations (i, j) of variables, receiving $$\boldsymbol{C}^a = \begin{pmatrix} \mathrm{Cor}(X_1, X_1) & \mathrm{Cor}(X_1, X_2) & \ldots & \mathrm{Cor}(X_1, X_d) \\ \mathrm{Cor}(X_2, X_1) & \mathrm{Cor}(X_2, X_2) & \ldots & \mathrm{Cor}(X_2, X_d) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cor}(X_d, X_1) & \mathrm{Cor}(X_d, X_2) & \ldots & \mathrm{Cor}(X_d, X_d) \end{pmatrix},$$ which is the multivariate normal correlation matrix.
5	Sample n times from the d-dimensional normal distribution with covariance matrix $\boldsymbol{C}^a$ and mean vector $\boldsymbol{\mu}^a = \mathbf{0}$ receiving the $n \times d$ sample matrix $\boldsymbol{S}^a$.
6	Reconvert $\boldsymbol{S}^a$ to the ordinal sample matrix $\boldsymbol{S}$ using $$\boldsymbol{S}_{[ei]} = k : \{F_{A_i}(k-1) < \Phi(\boldsymbol{S}_{[ei]}^a) < F_{A_i}(k)\}$$ for $k \in \{1, 2, \ldots, k_i\}$ and Φ being the univariate standard normal distribution function.

Tab. 4.4: Generation of multivariate ordinal random numbers via the mean mapping method.

5. Software

The success of a method depends largely on the availability of software implementations. For biclustering there is a growing number of stand alone tools, but there has been no integration in standard software so far. To change this, we developed the R (R Development Core Team, 2011) package `biclust`, which contains a comprehensive selection of algorithms, preprocessing, visualization, and validation methods. Using the capabilities of R we can easily visualize bicluster solutions using parallel coordinates and heatmaps, or evaluate the stability of cluster solutions using the bootstrap. The newly developed bicluster membership plot provides a graphical overview of the variables that define each bicluster. The goal of this unified toolbox is to enable researchers in diverse fields to test bicluster methods on their two-dimensional data and to compare different approaches.

In the following section we describe the structure of the `biclust` package, give a short introduction to the theoretical background of the algorithms, and demonstrate the additional pre- and postprocessing methods on an artificial data example. In Section 5.2 we give illustrative examples of how the package is used on microarray data. Additionally, we provide an overview of other existing R packages and stand alone software.

5.1 Package biclust

The package collects various approaches to biclustering into one bundle, by providing the results as an entity of Class `Biclust`. This S4-class contains all the information needed for postprocessing results. The class consists of five slots `Parameters`, `RowxNumber`, `NumberxCol`, `Number` and `info`. The slot `Parameters` contains parameters and algorithm used, `Number` contains the number of bicluster found, and `info` shows additional results from the algorithms as a list. The `RowxNumber` and `NumberxCol` slots represent the bicluster that have been found. They are both logical matrices of dimension (rows of data $\times$ number of bicluster found) with a TRUE-value in `RowxNumber[i,j]` if row i is in bicluster j. `NumberxCol` functions in the same way for the columns, but

for computational reasons, the rows of the matrix in this slot represent the number of bicluster and the columns represent the columns of data.

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 needs the preprocessed data matrix `x`, a bicluster algorithm represented as a `Biclustmethod` Class, and additional arguments (`...`)as input. This structure allows us to easily include a new method in the package which already contains algorithms representing all four major outcome classes. As stated in Kaiser and Leisch (2008) and in Section 5.2 of this chapter these algorithms deliver completely different result sets.

5.1.1 Algorithms

This section briefly describes the algorithms in the package and how they are used. The algorithms are applied to a normal distributed artificial data set containing a hidden bicluster:

```
> set.seed(1234)
> artdata <- matrix(rnorm(5000), 100, 50)
> x <- 1:100 %in% sample(1:100, 10)
> y <- 1:50 %in% sample(1:50, 10)
> artdata[x, y] <- rnorm(100, 3, 0.1)
```

The data is then compared to the expected result, an instance of `Biclust-Class` containing the hidden bicluster.

```
> library(biclust)
> artres <- new("Biclust", Parameters = list(),
+     RowxNumber = as.matrix(x), NumberxCol = t(as.matrix(y)),
+     Number = 1, info = list())
```

CC

The CC method implements the algorithm by Cheng and Church (2000) and searches for bicluster with constant values. As stated in Chapter 2 they define a score

$$H(I,J) = \frac{1}{\|I\|\|J\|} \sum_{i \in I, j \in J} (a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2, \tag{5.1}$$

where a_{iJ} is the mean of row i,a_{Ij} is the mean of column j and a_{IJ} is the overall mean. Cheng and Church (2000) consider a subgroup a bicluster if the score

is below a `delta` level and above a `alpha`-fraction of the overall score. The algorithm itself has three major steps:

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

These steps are repeated until a maximum number of bicluster is obtained or until no more bicluster can be found.

```
> set.seed(1234)
> rescc <- biclust(artdata, method = BCCC(), alpha = 1.5,
+     delta = 0.3, number = 10)
```

Plaid

This method uses the plaid model of Lazzeroni and Owen (2002) and an adaption of the original code from Turner et al. (2005), to represent bicluster with constant rows or columns. The original algorithm was fitting layers k to the model

$$Y_{ij} = (\mu_0 + \alpha_{i0} + \beta_{j0}) + \sum_{k=1}^{K} (\mu_k + \alpha_{ik} + \beta_{jk}) \rho_{ik} \kappa_{jk} + \varepsilon_{ij} \tag{5.2}$$

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 belong to the layer. Once the residuals are computed, bicluster are calculated as follows:

1. Update all parameters one after another `iter.layer` + `iter.startup` 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 this implementation, OLS is replaced with a binary least square algorithm.

```
> set.seed(1234)
> resplaid <- biclust(artdata, method = BCPlaid(), back.fit = 2,
+     shuffle = 3, fit.model = ~m + a + b, iter.startup = 5,
+     iter.layer = 30, verbose = F)
```

Spectral

The bicluster algorithm described by Kluger et al. (2003) results in a bicluster with coherent values. It includes several preprocessing steps, such as normalization, independent scaling (`"irrc"`), bistochastization (`"bistochastization"`) and log interactions (`"log"`). The following steps are performed to find relevant submatrices:

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

The quantity of bicluster depends on the number and value of the eigenvalues. The algorithm returns a checkerboard structure, from which bicluster are reported if they full fill the `minGenes`, `minCondition` and `withinVar` conditions.

```
> set.seed(1234)
> resspect <- biclust(artdata, method = BCSpectral(),
+     normalization = "irrc", numberOfEigenvalues = 2,
+     minr = 2, minc = 2, withinVar = 1)
```

Xmotifs

As described in Chapter 2 bicluster with coherent evolutions are represented by the Xmotifs algorithm of Murali and Kasif (2003). Since it searches for conserved gene states good preprocessing is crucial. One way to working with gene states is to discretize the data (for example with function `discretize()`). Once the data matrix represents the states, the algorithm chooses a random column `ns` times and then performs these steps:

1. Choose an `sd` subset of columns `nd` times and collect all rows with equal state in this subset, including the above column.
2. Collect all columns where these rows have the same state.
3. Return the bicluster if it has the most rows of all the bicluster found and if it is also larger than an `alpha` fraction of the data.

To collect more than one bicluster, one can rerun the calculation without the rows and columns found or just return the smaller combinations found before.

```
> set.seed(1234)
> artdata.xmotif <- discretize(artdata, nof = 14)
> resxmotif <- biclust(artdata.xmotif, method = BCXmotifs(),
+     ns = 150, nd = 150, sd = 3, alpha = 0.1, number = 50)
```

Bimax

Alongside the algorithms representing all four outcome classes, the package also includes the Bimax algorithm of Prelic et al. (2006) using their original and fast C Code. As described in Chapter 2 the algorithm searches in a binary matrix. A data matrix can be transformed into a binary matrix using for example the `binarize()` function. The idea behind the Bimax algorithm is to partition binary data into three submatrices, one of which contains only 0s and therefore can be discarded. The algorithm is then recursively applied to the remaining two submatrices U and V; the recursion ends if the current matrix represents a bicluster, that is, contains only 1s. In order to avoid overlaps, the next bicluster is found starting the basic algorithm on data excluding the rows of the already found bicluster. The algorithm works as follows:

1. Divide the data matrix in two column sets CU and CV by drawing a random row with at least a prespecified minimum of 1s, CU are then columns where this row is 1, CV the others.

2. Divide the rows: RU are those rows that contain only 0s in column set CV, RV are those rows that contain only 0s in columns set CU, the remaining rows are called RUV.

3. Report matrices U [rows = RU + RUV, columns = CU] and V [RUV + RV, ALL] and delete matrix W [RU, CV].

4. Repeat Steps 1 to 3 on submatrices U and V until minimum size is reached and report matrices containing only 1s. If U and V do not share any rows and columns, the two matrices can be processed independently from each other. However, if U and V have a set of rows in common, special care is necessary to only generate those bicluster in V that share at least one common column with CV. See Prelic et al. (2006) for more details.

The original C Code, invoked by

```
> set.seed(1234)
> artdata.bimax <- binarize(artdata, threshold = 2)
> resbi <- biclust(artdata.bimax, method = BCBimax(), minr = 2,
+     minc = 2, number = 50)
```

finds all possible one exclusive matrices without ordering. This leads to too many bicluster and the interesting large bicluster are often mist. We use this algorithm in our modified repeated Bimax method to obtain the largest bicluster. The repeated Bimax works as follows:

1. Run the ordinary Bimax and throw away all bicluster if a bigger bicluster is found.
2. Stop if no bigger bicluster is found or a maximum column (`macc`) number is reached
3. Store the found matrix as a bicluster. Delete the rows in this bicluster from the data and start over.
4. Repeat steps 1 to 3 until no new bicluster is found.

The repeated Bimax is calculated using

```
> resrepbi <- biclust(artdata.bimax, method = BCrepBimax(),
+     minr = 2, minc = 2, number = 50, maxc=50)
```

Quest

Although all of these algorithms are developed for gene or microarray data, the package is not limited to this application. Another possible application is questionnaire data, which is frequently used in social science. Questionnaires are used to classify respondents based on the answers they give. It therefore makes sense not to use the whole questionnaire, but rather groups of questions in which the respondents give similar answers. For example, as shown in Chapter 6, by using the Bimax algorithm, one is able to find subgroups of tourists doing the same activities during their holidays.

We developed the Quest algorithm, which contains three methods for working with different scale levels, especially for biclustering this type of questionnaire data. Quest functions like the Xmotifs algorithm if the answers are given on a nominal scale (`BCQuest`) except that the rows (respondents) of the bicluster found are eliminated from the ongoing calculations. For the ordinal scale, the algorithm (`BCQuestord`) looks for similar answers in an interval set

by a prespecified parameter d. The interval contains d lower and d higher classes than the starting class of the chosen respondents. In a continuous case (`BCQuestmet`), this interval is set by the quantile (`quant`) of a normal distribution with variance `vari`.

The algorithms works similar to the Xmotifs algorithm:

1. Choose a random row x_r `ns` times
2. Choose an `sd` subset J_{sd} of columns `nd` times and collect all rows i with

   Quest: $a_{ij} = a_{rj}$

   Questord: $a_{ij} \in [a_{rj} - d; a_{rj} + d]$

   Questmed: $a_{ij} \in [a_{rj} - vari * z_{1-quant}; a_{rj} + vari * z_{1-quant}]$

   for all $j \in J_{sd}$ in I_{found}
3. Collect all columns j with

   Quest: $a_{ij} = a_{rj}$

   Questord: $a_{ij} \in [a_{rj} - d; a_{rj} + d]$

   Questmed: $a_{ij} \in [a_{rj} - vari * z_{1-quant}; a_{rj} + vari * z_{1-quant}]$

   for all already found rows i in J_{found}.
4. Save the largest subgroup (I_{found}, J_{found}) as a bicluster if it is larger than an `alpha` fraction of the data.
5. Delete rows I_{found} from the data and repeat Steps 1 - 4 until the maximum number is reached or no more bicluster is found.

Results are calculated calling

```
> biclust(x, method = BCQuest(), ns = 10, nd = 10, sd = 5,
+     alpha = 0.05, number = 100)
> biclust(x, method = BCQuestord(), d = 1, ns = 10, nd = 10,
+     sd = 5, alpha = 0.05, number = 100)
> biclust(x, method = BCQuestmet(), quant = 0.25, vari = 1,
+     ns = 10, nd = 10, sd = 5, alpha = 0.05, number = 100)
```

5.1.2 Ensemble Method

The package also implements of the ensemble method described in Chapter 3. The ensemble method is applied using the ensemble function

```
> resensemble <- ensemble(x, confs = plaid.grid(),
+     rep = 20, maxNum = 5, similar = jaccard2, thr = 0.8,
+     simthr = 0.7, subs = c(1, 1), bootstrap = FALSE,
+     support = 0, combine = qt, ...)
```

with the parameters

x: Data Matrix

confs: Matrix containing parameter sets

rep: Number of repetitions for each parameter set

maxNum: Maximum number of bicluster taken from each run

similar: Function to produce a similarity matrix of bicluster

thr: Threshold for similarity

simthr: Proportion of row column combinations in bicluster

subs: Vector of proportion of rows and columns for sub-sampling. Default c(1,1) means no sub-sampling.

bootstrap: If TRUE, bootstrap sampling is used

support: Proportion of the runs, which must contain the bicluster in order to have enough support to report it (between 0 and 1)

combine: Function to combine the single bicluster

...: Arguments passed to the combine function.

An important element of this method is the `confs` parameter. If one chooses to use the plaid model with varying parameter settings, the function `plaid.grid()` takes the committed parameters (vectors) of the BCPlaid functions and expands a parameter grid containing all possible combinations. The package contains grid functions for all the algorithms in the package, e.g. `grid.cc()` for the BCCC method, and other methods from different R packages.

At present,the package contains three combining methods:

hcl Hierarchical clustering (We suggest complete linkage)

qt Quality clustering by choosing the biggest cluster

sqt Quality clustering sampling proportional to size

and two similarity measures:

jaccard2 Jaccard index

correl Correlation approach

5.1.3 Bicluster Extraction

In addition to the usual print method, which shows the number of bicluster found and the first five bicluster dimensions,

```
> print(rescc)

An object of class Biclust

call:
        biclust(x = artdata, method = BCCC(), alpha = 1.5,
           delta = 0.3, number = 10)

Number of Clusters found:  10

First  5  Cluster sizes:
                  BC 1 BC 2 BC 3 BC 4 BC 5
Number of Rows:      18   14   12   12   10
Number of Columns:   14   15   15   12   12
```

and a `summary()` method listing all bicluster dimensions, the software package includes several other methods for working with the bicluster.

We can extract the bicluster using `bicluster(x, BicRes, number)`. This function returns a list containing all bicluster specified in `number`. The default setting is `1:BicRes@Number`, which returns all bicluster found. Each part of this list contains a data matrix with the entries from data `x` included in the bicluster.

If one is only interested in the row and column indices, `biclusternumber(BicRes, number)` delivers these as a similar list.

5.1.4 Bicluster Validation

Jaccard Index

Some additional methods deal with validating the results. The Jaccard index defined in section 2.3 compares two bicluster results `Bicres1` and `Bicres2` by calling `jaccardind(Bicres1, Bicres2)`.

It it useful to compare the calculated results between each other or with the true allocation as illustrated in the artificial data example:

```
> result <- c(jaccardind(artres, resplaid), jaccardind(artres,
+     resxmotif), jaccardind(artres, rescc), jaccardind(artres,
+     resspect), jaccardind(artres, resbi))
> names(result) <- c("BCPlaid", "BCXmotifs", "BCCC",
+     "BCSpectral", "BCBimax")
> result
```

```
    BCPlaid   BCXmotifs        BCCC  BCSpectral     BCBimax
1.000000000 0.015000000 0.002531646 0.800000000 0.158241757
```

```
> boxplot(as.data.frame(result), ylab = "Jaccard")
```

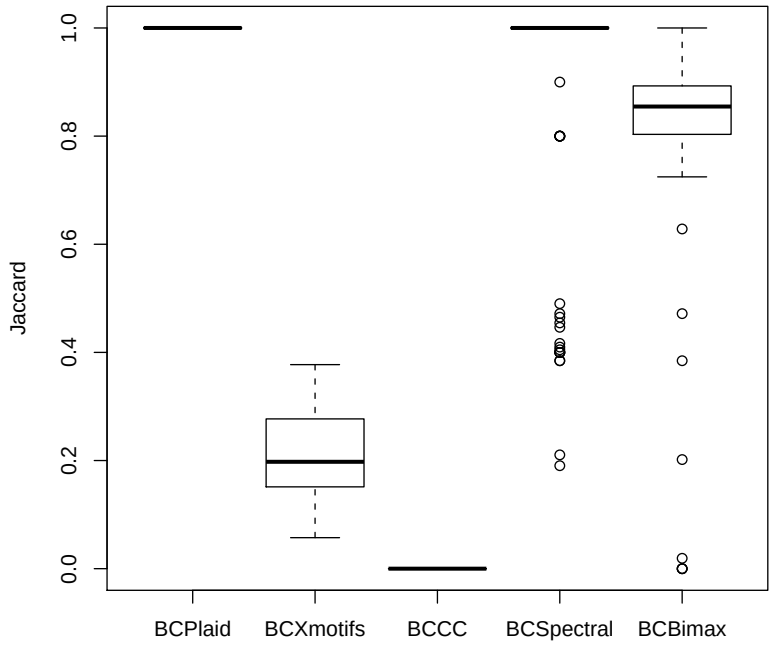

Fig. 5.1: Boxplot of Jaccard index from 100 simulation runs.

Figure 5.1 shows the same calculation as boxplots of 100 simulation runs and there are no major changes to the one run result. The Plaid, Spectral and Bimax methods were able to identify the hidden bicluster, though the latter two produce some errors, while Xmotifs and CC did not find the bicluster. The problem of the latter two algorithms was the high likelihood to find some constant row and column subgroups or subgroups in the same discrete state in the noise.

Variance Measure

In addition to methods for comparing two results the package also contains constance and coherence measures for single bicluster. Building on the work of Madeira and Oliveira (2004) for classifying bicluster, the function `constantVariance` returns the corresponding variance of rows as the average of the sum of Euclidean distances between all rows of the bicluster x:

$$\frac{1}{nrow(x) * (nrow(x) - 1)} \sum_{i=1}^{nrow(x)} \sum_{j=1}^{nrow(x)} \left(\frac{1}{ncol(x)} \sum_{k=1}^{ncol(x)} (x[i,k] - x[j,k])^2 \right)$$

`dimension == 'col'` does the same for columns using $t(x)$ and `dimension == 'both'` return the weighted mean of row and column calculation. `additiveVariance, multiplicativeVariance` and `signVariance` do the same calculation after an additive, a multiplicative or a sign transformation is applied to the bicluster.

```
> constantVariance(artdata, resplaid, 1, dimension = "both")

[1] 0.2352616

> additiveVariance(artdata, resplaid, 1, dimension = "both")

[1] 0.3099963

> multiplicativeVariance(artdata, resplaid, 1,
+     dimension = "both")

[1] 0.1031517

> signVariance(artdata, resplaid, 1, dimension = "both")

[1] 2.133500
```

Goodness of Fit

A second idea of validating single bicluster is to fit a model to the bicluster and compare it with the same model on the remaining data. The package contains two different approaches dealing with a linear model

$$a_{ij} = \mu + \alpha_i + \beta_j + \epsilon_{ij}$$

suggested by Kostka and Spang (2004). Chia and Karuturi (2010) fit this model on the rows within the bicluster with the columns (1) within the bicluster and (2) outside the bicluster. Using this estimates they propose two scores to validate the bicluster type:

$$T_w(BC_z) = \frac{1}{k_z} \sum_{i \in I_z} (\bar{a}_{i.})_w^2 - \frac{E_w(BC_z)}{l_{zw}}$$

$$B_w(BC_z) = \frac{1}{l_{zw}} \sum_{j \in J_{zw}} (\bar{a}_{.j})_w^2 - \frac{E_w(BC_z)}{k_z},$$

where

$$E_w(BC_z) = \frac{1}{k_z * l_{zw}} \sum_{i \in I_z, jin J_{zw}} (a_{ij} - \bar{a}_{i.} - \bar{a}_{.j} + \bar{a}_{..})^2,$$

and w =1,2 (1 denotes the columns within the bicluster z and 2 outside). Chia and Karuturi (2010) distinguish three major types of bicluster: (1) T type, which has strong row effect (constant columns), (2) B type, which has strong column effect (constant rows) and (3) μ type (constant values), which has both strong column and row effect. For the estimation of effect strength and ranking of bicluster they proposed to use the SB score:

$$SB = \log(\frac{\max(T_1 + a; B_1 + a)}{\max(T_2 + a; B_2 + a)}),$$

where a is a small constant to avoid high values if T_2 and B_2 are very small. A higher SB score indicates better separation between columns within and outside the bicluster. Last they define a score

$$TS = \log(\frac{T_m + a}{B_m + a}),$$

where $m = 1$ if $SB > 0$ and $m = 2$, if $SB < 0$. By defining a threshold for this score it is possible to classify a bicluster as T or B Type automatically.

In the package this scores are calculated using the `ChiaKaruturi()` function which takes the data, the bicluster result, and the number of the bicluster of interest as input.

```
> resChia <- ChiaKaruturi(artdata, bicResult = resplaid,
+     number = 1)
> resChia
```

```
   Tscore1 Bscore1    Tscore2    Bscore2  SBscore       TSscore
1 9.016207 9.01664 0.01258911 0.09101706 4.492646 -4.796266e-05
```

In Khamiakova et al. (2011) the F-Statistics inside and outside the bicluster are compared. First the F- Statistics (F_{row} and F_{col}) for the row

$$a_{ij} = \mu + \alpha_i + \epsilon_{ij}$$

and column

$$a_{ij} = \mu + \beta_j + \epsilon_{ij}$$

model are calculated. Then a given number of bootstrap or sub samples of columns outside the bicluster are drawn. On each sample again both F-Statistics were calculated. To see if a row or column effect is visible in the bicluster the percentage of F- Statistics higher than the F-Statistic in the bicluster are reported as row or column bootstrap p-value. If this value is smaller than a given threshold a row or column effect is visible. The bootstrap p-values together with the F-Statistics of the bicluster are calculated using

```
> Bootstrap <- diagnoseColRow(artdata, bicResult = resplaid,
+     number = 1, nResamplings = 999, replace = TRUE)
> diagnosticPlot(bootstrapOutput = Bootstrap)
```

The `diagnosticPlot()` function plots a histogram of the F-Statistic of the samples with the value within the bicluster marked as green line. An example of the plot is given in Figure 5.2.

5.1.5 Bicluster Visualization

For the most part, results are still validated by analyzing graphical representations. Our package includes popular techniques for bicluster visualization.

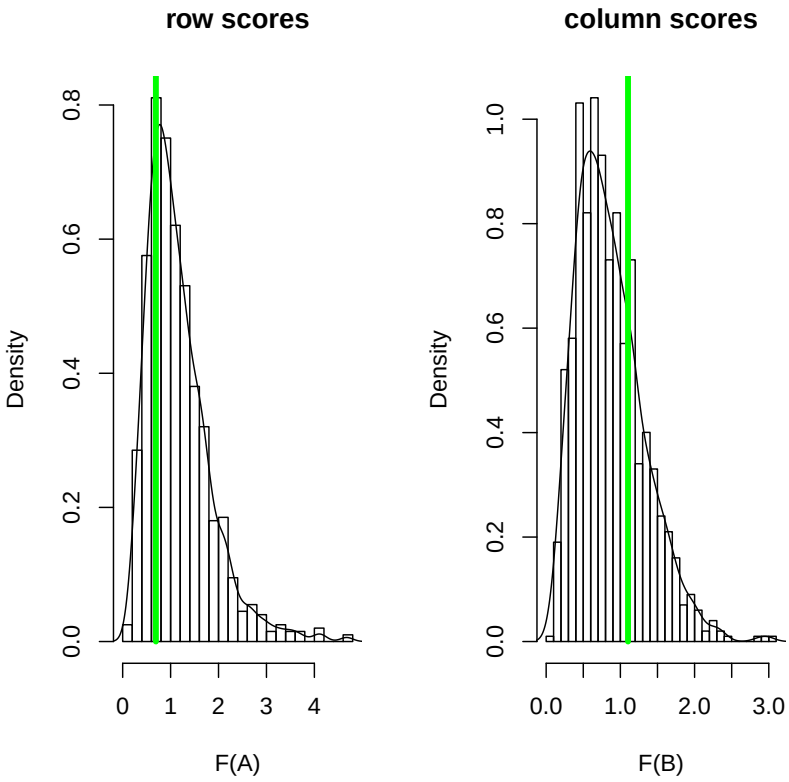

Fig. 5.2: Diagnostic plot for artificial data: Since the bicluster is constant over rows and columns no row or column effect is visible.

Heatmap

Standard visualization for gene expression data is a heatmap. This heatmap can be plotted using `drawHeatmap()` without a bicluster result. Given a bicluster result, the rows and columns are rearranged so that the bicluster is shown in the upper right corner. For a closer look at the bicluster, the parameter `local` shows a heatmap containing only the rows and columns of the bicluster. Figure 5.3 illustrates how the heatmap rearranges the artificial dataset to represent the hidden bicluster.

```
> drawHeatmap(x = artdata)
> drawHeatmap(x = artdata, bicResult = resplaid, number = 1,
+     local = F)
```

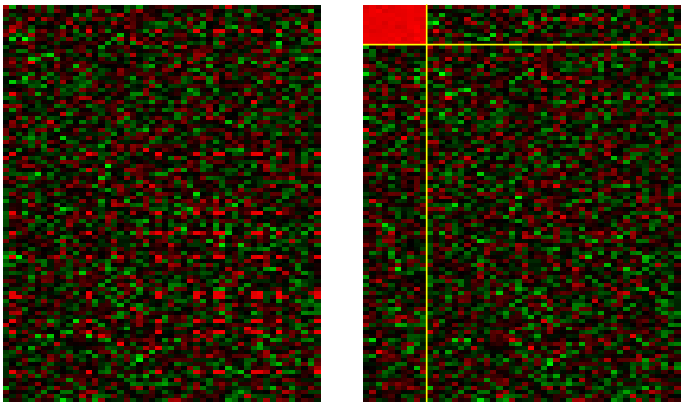

Fig. 5.3: Heatmap of artificial data (left) and reordered with plaid result to identify the bicluster (right).

Simple rearrangement is not enough to show multiple bicluster in one heatmap (especially when they are overlapping). For this purpose a second heatmap function

```
> heatmapBC(x, bicResult, number = 1:4, local = FALSE, ...)
```

is included in the package. It uses `image()`to draw the heatmap and is able to plot all bicluster given in `number` in one heatmap. Examples are given in Figure 3.1 in Chapter 3.

Parallel Coordinates

Secondly the package uses parallel coordinates to visualize the bicluster result. Since the newest biclustering developments for biclustering originate from gene

expression data, this plot of results is state-of-the-art. Compared to traditional parallel coordinates plots, both dimensions are equally important. One can either plot both dimensions in one graph, calling `plotBoth=TRUE`, or one can choose between columns (`plotcol=TRUE`) or rows to represent the lines. In order to compare the values within the bicluster with those outside, `compare=TRUE` shows the outside values as gray lines. For additional information The command `info=TRUE` adds a title with additional information (number of bicluster, rows and columns).Figure 5.4 shows the parallel coordinates of rows and columns in the Plaid result of the artificial data.

```
> parallelCoordinates(x = artdata, bicResult = resplaid,
+     number = 1, plotBoth = TRUE, plotcol = TRUE,
+     compare = TRUE, info = F, bothlab = c("Rows", "Cols"),
+     ylab = "Value", col = 2, las = 2)
```

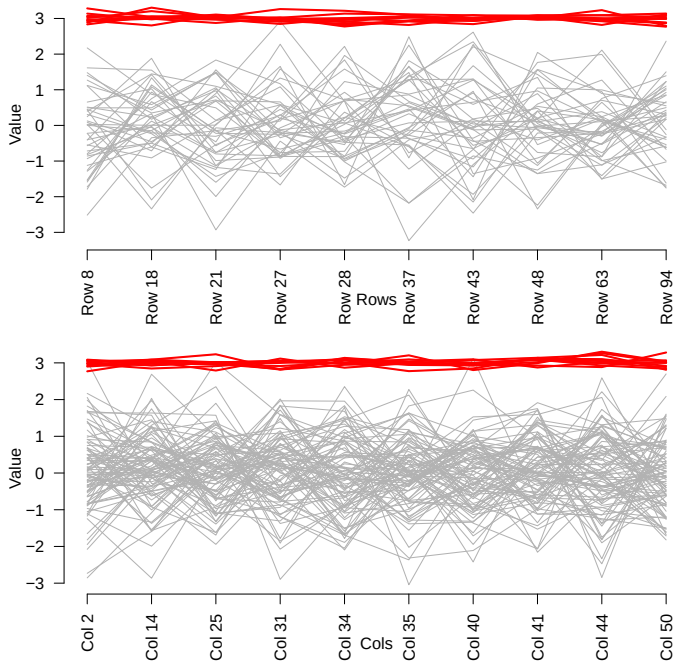

Fig. 5.4: Parallel coordinates of columns(upper) and rows(lower)

Membership Plot

If the number of columns (e.g. in questionnaires or preselected conditions in a microarray experiment) is lower than 50, the bicluster results can be presented in a membership plot (`biclustmember`). This plot is meant to compare the bicluster to each other or to a normal cluster result. Each bicluster is

represented as a bar of stacked rectangles, where each rectangle represents a column in the data set. If the column is not include in the specific bicluster, the rectangle is plain white. If a bicluster contains a variable, the corresponding rectangle is filled with a color, which represents the mean value of this variable for all the objects within the bicluster. Depending on the variable mid, the rectangle contains the global mean value for this variable in the middle (`mid = TRUE`) or on the right half (`mid = FALSE`). If the membership plot is called on a cluster result (`clustmember`) there are no white rectangles. The default color code is `diverge_hcl(101, h = c(0, 130))` which is red for the minimal value, gray for values around the median, and green for maximal values. It is possible to set ones' own color palettes in the `color` argument. The `which` argument allows the variables to be ordered. The ordering can be the result of a hierarchical clustering or the outcome of `bicorder`.

The plot is invoked by

```
> biclustmember(bicResult, x, mid = TRUE, cl_label = "",
+     which = NA, main = "Bicluster Unsorted", xlab = "Cluster",
+     color = diverge_hcl(101, h = c(0, 130)), ...)
```

Figure 5.5 shows an example.

Bicluster Barchart

Alternative to the membership plot a classic barchart can be used to visualize such a bicluster result. Only variables within the bicluster were represented by a bar, and the mean value outside the bicluster is shown as a red dot. The corresponding function in the package is

```
> biclustbarchart(x, bicRes, which = NULL, ...)
```

An example is shown in Figure 5.6

5.1.6 Little Helpers

The many different approaches to biclustering have led to the development of a large number of small helper functions. For example, we can use the following call to order the bicluster for some of the visualizations above:

```
> bicorder(bicResult, cols = TRUE, rev = FALSE)
```

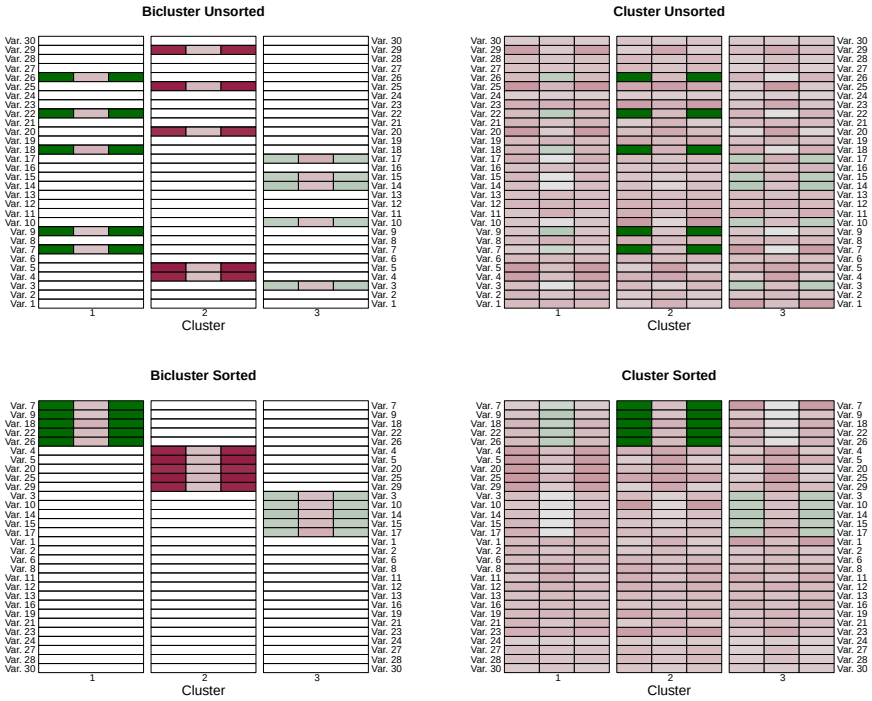

Fig. 5.5: Membership graph with (bottom) and without(top) `bicorder` ordering. Bicluster results are on the left, and k-mean results on the right side.

Here the columns are ordered according to their appearance in a bicluster (Using `cols = FALSE` the rows can be ordered). The result is a vector in which column numbers from the first bicluster are listed first and columns which do not appear in any bicluster are last (using `rev = TRUE` the order can be reversed). Ordering is quite simple if there is no overlapping in the data, but a more sophisticated approach is needed for overlapping. For example columns which appear in a subsequent bicluster must be arranged at the end of the first bicluster and at the top of the second bicluster.

Furthermore bicluster results are still cluster results, so traditional cluster post-processing works as well. The package includes small helper files which transform the bicluster result so that it can be used in many other application or R functions. The call

```
> writeBiclusterResults(fileName, bicResult, bicName, geneNames,
+     arrayNames, append = FALSE, delimiter = " ")
```

will write all bicluster in `bicResult` to the file `filename` using `geneNames` for the row names and `arrayNames` for the column names. To convert the bicluster into a normal cluster result,

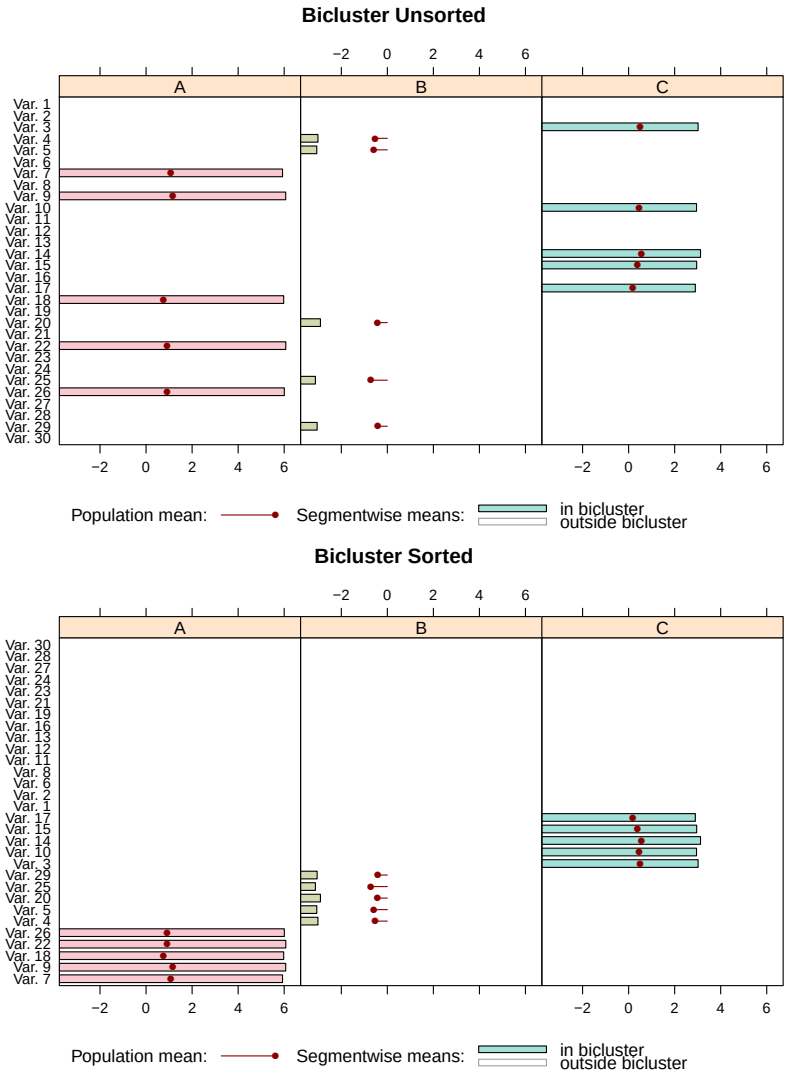

Fig. 5.6: Barchart Graph with (bottom) and without(top) `bicorder` Ordering.

```
> writeclust(bicResult, row = TRUE)
```

will return a normal cluster result for rows (for column set `row=FALSE`), with non-clustered rows sorted to Cluster 0. The number of validation, visualization and helper functions in the package is constantly growing in order to keep up with all major advances in bicluster analysis.

5.2 Illustrations

To illustrate the functionality of the package, the following section shows how to calculate bicluster on the `BicatYeast` dataset, which is included in the package. In several examples this dataset shows how the bicluster can be validated and how the visualization methods are applied. Also, a cross-validation study shows the reliability of the single algorithms.

5.2.1 Yeast Data

Prelic et al. (2006) use this microarray dataset to present their bicluster technique. This is a subsample of the Saccharomyces Cerevisiae organism (Yeast) and contains 419 genes on 70 conditions. After loading the data,

```
> library(biclust)
> data(BicatYeast)
```

we had to preprocess using binarized or discretized data.

```
> xmotifdata <- discretize(BicatYeast, nof = 10, quant = TRUE)
> bimaxdata <- binarize(abs(BicatYeast), threshold = 2)
```

After calculating all the different bicluster results

```
> Bicatresplaid <- biclust(x = BicatYeast, method = BCPlaid(),
+     back.fit = 2, shuffle = 3, fit.model = ~m + a + b,
+     iter.startup = 5, iter.layer = 30, verbose = F)
> Bicatresxmotif <- biclust(x = xmotifdata, method = BCXmotifs(),
+     ns = 200, nd = 200, sd = 4, alpha = 0.05, number = 50)
> Bicatrescc <- biclust(x = BicatYeast, method = BCCC(),
+     delta = 0.01, number = 50)
> Bicatresspect <- biclust(x = BicatYeast, method = BCSpectral(),
+     withinVar = 4)
> Bicatresbi <- biclust(x = bimaxdata, method = BCBimax(),
+     minr = 5, minc = 5, number = 50)
```

we found that the results contained different bicluster with different numbers of rows and columns. For example, the result of the `BCXmotifs` algorithm

```
> Bicatresxmotif
```

```
An object of class Biclust

call:
        biclust(x = xmotifdata, method = BCXmotifs(), ns = 200,
            nd = 200, sd = 4, alpha = 0.05, number = 50)

Number of Clusters found:  38

First  5  Cluster sizes:
                   BC 1 BC 2 BC 3 BC 4 BC 5
Number of Rows:      23   15   14   20   10
Number of Columns:    5    6    5    5    5
```

found more cluster with equal numbers of columns, while the `BCPlaid` result

```
> Bicatresplaid


An object of class Biclust

call:
        biclust(x = BicatYeast, method = BCPlaid(), back.fit = 2,
            shuffle = 3, fit.model = ~m + a + b,
            iter.startup = 5, iter.layer = 30, verbose = F)

Number of Clusters found:  7

First  5  Cluster sizes:
                   BC 1 BC 2 BC 3 BC 4 BC 5
Number of Rows:      23   28   26   24    3
Number of Columns:   12    5    7    3    3
```

contained fewer bicluster in various column sizes. A first validation can be done by simply comparing the Jaccard index of all 5 results. The matrix `jacresult` contains the result of comparing each algorithm to all other algorithm calling `jaccardind()`. This matrix is filled, calling

```
> jacresult[1, 2] <- jaccardind(Bicatresplaid, Bicatresxmotif)
```

on all possible combinations.

```
> jacresult
```

```
            Alogrithmus
Alogrithmus    BCPlaid BCXmotifs      BCCC BCSpectral   BCBimax
  BCPlaid    1.0000000 0.0128998 0.0044838  0.0069520 0.0000000
  BCXmotifs  0.0128998 1.0000000 0.0321175  0.0042682 0.0060214
  BCCC       0.0044838 0.0321175 1.0000000  0.0116795 0.0000000
  BCSpectral 0.0069520 0.0042682 0.0116795  1.0000000 0.0000000
  BCBimax    0.0000000 0.0060214 0.0000000  0.0000000 1.0000000
```

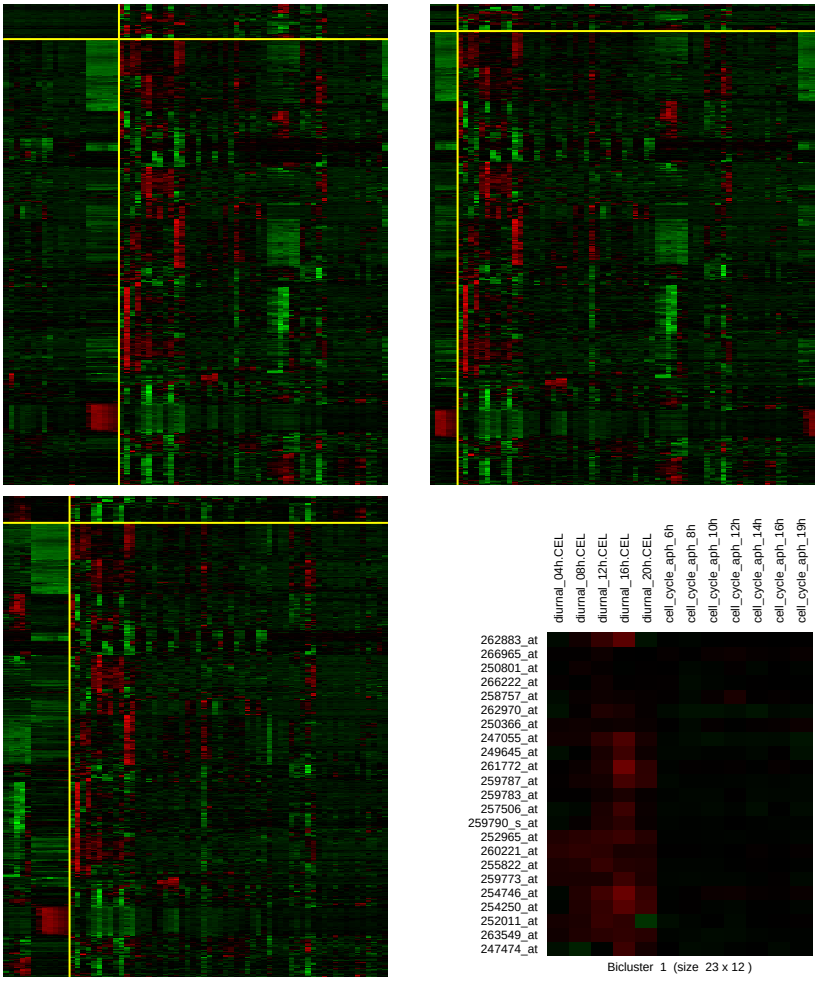

Fig. 5.7: Heatmap of first bicluster in CC, Xmotif and Plaid results and a closer look at the plaid bicluster.

As mentioned above, the algorithms search for completely different submatrix structures, so it is not surprising that the Jaccard index of different `Bicluster-Result` is close to zero. This indicates that the bicluster found are completely different to each other. Concentrating on one structure (e.g constant biclus-

ter), we can perform a second validation by calculating the `constantVariance` measure. If we perform

```
> varresult <- constantVariance(x = BicatYeast,
+     resultSet = Bicatrescc, number = 1)
```

for the first bicluster in each result, we obtain

```
> varresult
```

```
   BCPlaid  BCXmotifs       BCCC BCSpectral    BCBimax
   1.06805    0.59070    0.58055    1.68799    2.44496
```

on all 5 result sets. The calculations show the expected result, since CC searches for constant bicluster and Xmotifs contains genes in the same state.

To evaluate whether the plaid result represents bicluster in an acceptable constant state, we can plot a heatmap (`drawHeatmap` and compare it to the heatmap of a constant bicluster. Figure 5.7 shows the heatmap of the first bicluster in the CC, the Xmotif, and the Plaid result produced by

```
> drawHeatmap(x = BicatYeast, bicResult, number = 1, local = F)
```

In the lower right corner, a heatmap of the first bicluster in the Plaid result is pictured without the remaining data. This is done by setting `local = TRUE`. Here it is possible to identify the genes and conditions by name, since they are drawn as labels.

The `parallelCoordinates` plot, shows the expression level of a gene over all conditions or vice versa. Figure 5.8 clearly shows that the Plaid result identifies gene condition combinations with an expression level lower than 0, especially in the zoomed version(`compare = FALSE`) in the lower row. The result also shows much more variance than the CC result. The graphs are generated using

```
> parallelCoordinates(x = BicatYeast, bicResult, number = 1,
+     info = TRUE, compare = TRUE, col = 1)
```

on the Plaid and CC results.

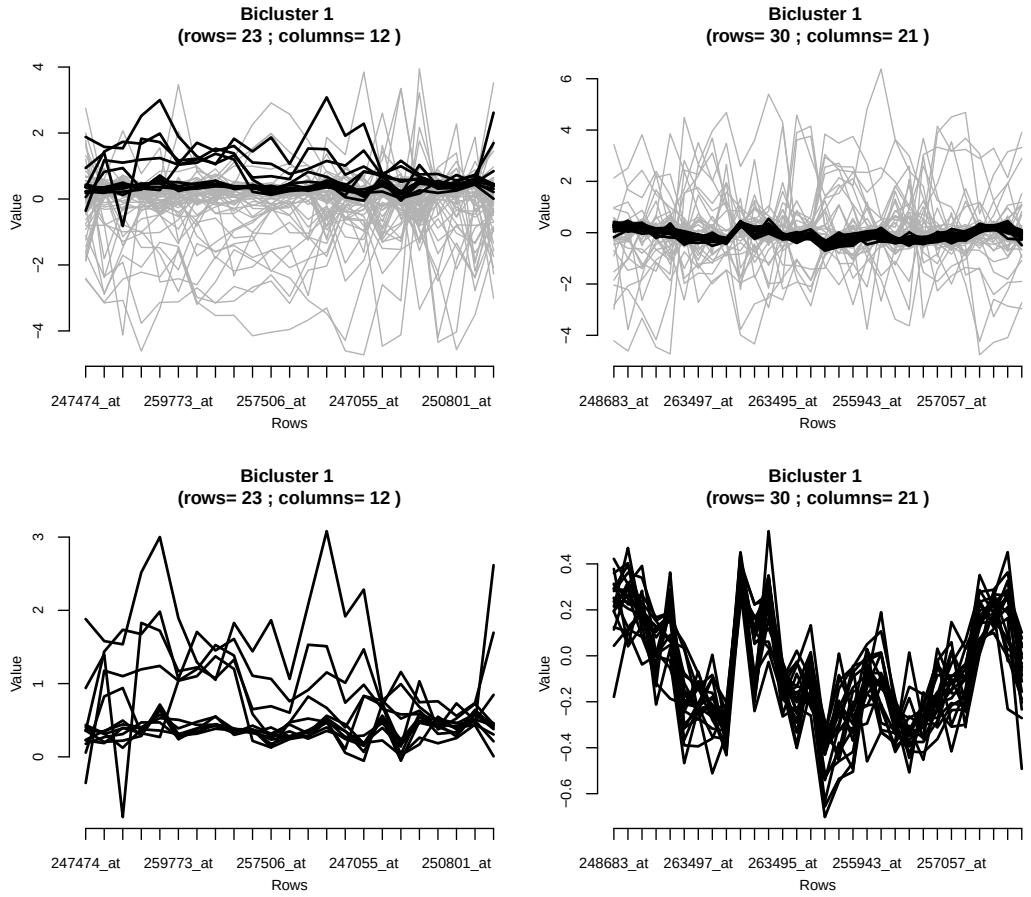

Fig. 5.8: Parallel Coordinates of bicluster 1 of Plaid(left) and CC (right).

5.2.2 Bootstrap Cross-Validation

In order to evaluate, the reliability of the calculations, a pseudo cross-validation was run. Fifty bootstrap samples were taken from the real data:

```
> bootsample[[1]] <- BicatYeast[sample(1:419, 419,
+     replace = TRUE), sample(1:70, 70, replace = TRUE)]
```

The bicluster were calculated with all 5 algorithms and the result was compared with the result of the corresponding algorithm on the whole dataset using the Jaccard index. So once again, 1 is a perfect match with the complete dataset, and 0 is a perfect mismatch.

Figure 5.9 shows the result of the 50 bootstrap samples as boxplots. The CC method nearly always found the primary combination because the score from

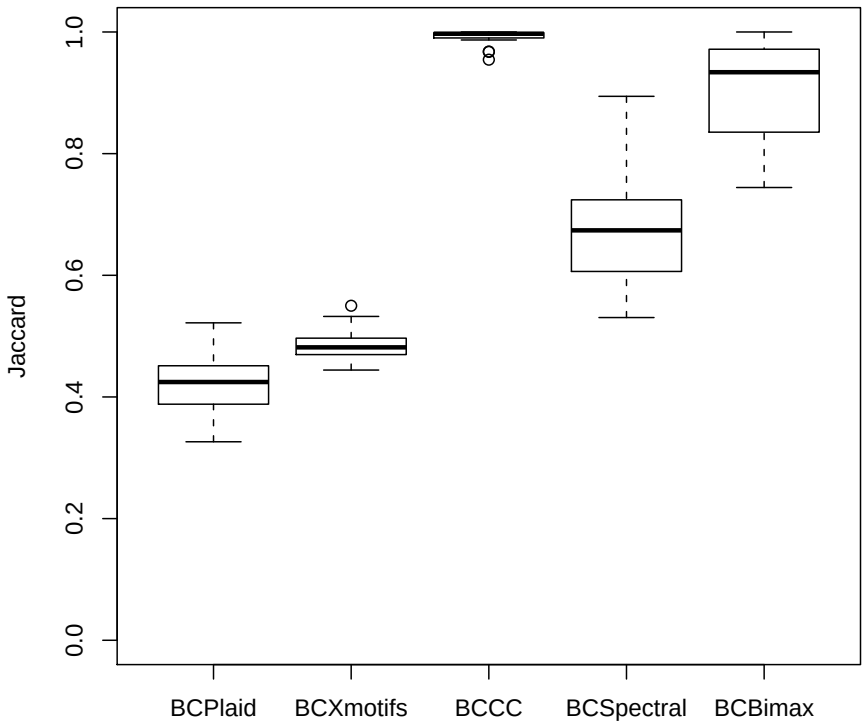

Fig. 5.9: Boxplot of bootstrap cross-validation results for Jaccard index.

these combinations does only slightly change if rows or columns were removed. The Bimax method also found the same bicluster as applied to the whole dataset. Removing ones from a matrix with ones has no real influence on the result. In the Plaid and Xmotifs algorithms changing the rows and columns led to a dislocation in the result set, having equal rows cause the algorithms to drop other rows previously included in the bicluster.

5.3 Other Software

In addition to our `biclust` package there are several other R packages and stand alone softwares associated with biclustering. In cooperation with other working groups, we have two unpublished packages hosted on R-Forge. Below, we will give a short overview of (a) our R-Forge projects, (b) other available R packages and (c) stand alone software associated with bicluster analysis.

5.3.1 R-Forge Packages

We are currently working on two bicluster software packages, one with a group at the DKFZ in Heidelberg and on one with a group in Hasselt, Belgium. These packages are available on R-Forge (`http://r-forge.r-project.org/`, Theußl and Zeileis (2009)).

sv4d

The `sv4d` package (Sill and Kaiser, 2011) implements our sv4d algorithm (Sill et al., 2011) and the biclustering by means of sparse singular value decomposition by (Lee et al., 2010). The package also includes an alternative heatmap plot for overlapping bicluster.

BiclustGUI

The `BiclustGUI` package (Pramana and Kaiser, 2010) contains an R Commander (Fox, 2010) Plug-In for bicluster analysis. It enables R Commander users to calculate and analyze bicluster. We implemented almost all functions of the `biclust` package and some of the R packages mentioned below (`isa2` and `fabia`).

Figure 5.10 shows the input windows for the CC calculation and the bicluster plots. This package enables inexperienced R users to run bicluster calculations.

5.3.2 R Packages

The number of R packages is growing and with it the number of packages on CRAN (R Development Core Team, 2011) that are associated with bicluster algorithms or results. Graphical user interfaces like `rattle` (Williams, 2009), use the functionality of our package to retrieve some bicluster results. Moreover there are packages which support a specific algorithm. These are listed below.

ISA

The Iterative Signature Algorithm of Bergmann et al. (2003) is used in three R packages. The `isa2` package (Csardi, 2009) includes the main implementation of the algorithm, the `eisa` package (Csardi, 2010) optimizes the algorithm for data from the bioconductor project (Gentleman et al., 2004) and the `ExpressionView` package provides useful visualization for bicluster results. All of these packages include an interface to the biclust package. The results of `isa2` and `eisa` can be converted to a `BiclusterResult` object, after which all the functions of the `biclust` package can be used. A `BiclusterResult` object can be read into the `ExpressionViewer`. These techniques are thus available for the results of the algorithms in our package.

Fabia

The `fabia` package implements the Fabia algorithm of Hochreiter et al. (2010). It is available on bioconductor (Gentleman et al., 2004) and includes some additional visualization tools. Those plots are especially useful if a bicluster algorithm is based on a singular value decomposition. The `fabia` package also contains example datasets used by the authors to compare different bicluster algorithms. An interface for using our `biclust` package is also available, in this package.

cMonkey

The `cMonkey` package (Reiss, 2011) implements the idea of Reiss et al. (2006). It also contains some validation datasets and comes with a detailed description and support website. The package includes the usual plot and output methods but does not yet include an interface to our package.

5.3.3 Stand Alone Software

Outside the R environment, there are hundreds of tools which deal with either bicluster calculation or bicluster validation. We will mention two of those programs: one calculates bicluster using 5 different algorithms and one visualizes bicluster results.

BicAT

The Bicluster Analysis Tool (BicAT, Barkow et al. (2006) is a Java-based graphical user interface for calculating, analyzing, and visualizing bicluster

results. It includes the algorithms Bimax (Prelic et al., 2006), CC (Cheng and Church, 2000), xMotifs (Murali and Kasif, 2003), ISA (Bergmann et al., 2003), and OPSM (Ben-Dor et al., 2003). It also includes heatmap and parallel coordinate plots and some summary statistics and output functions.

BicOverlapper

The BicOverlapper (Santamaría et al., 2008) tool is also Java-based. It is a framework for supporting the visual analysis of gene expression by means of biclustering (Santamaría et al., 2008). It includes the traditional plots such as heatmap and parallel coordinates as well as the possibility to visualize different sets of bicluster simultaneously. BicOverlapper has is an interface to R so that it is possible to draw results obtained with R.

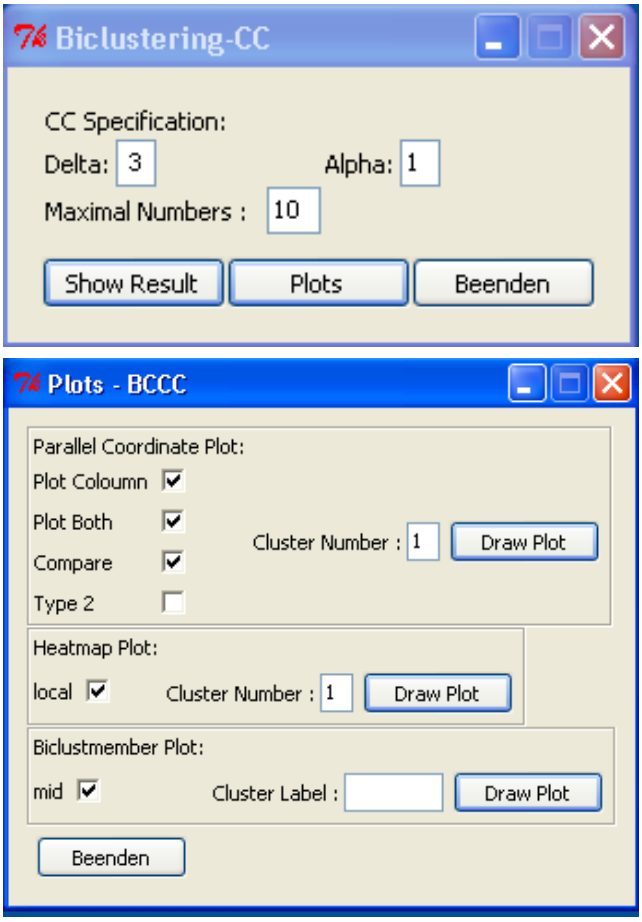

Fig. 5.10: `BiclustGUI` Input Windows: CC(above) and Plots(below)

5.4 Conclusion

This chapter demonstrated how to use the R package `biclust`, a toolbox for bicluster analysis. The package contains a continually growing number of bicluster algorithms, validation and visualization methods, and other helpful tools for bicluster analysis. Wes demonstrated that the algorithms in the package lead to completely different results, and in a simulation studies we applied the algorithms and other functions of the package to typical data situations. Moreover, we provided an additional yeast data example and gave an overview of other R packages and stand alone software.

6. Application on Marketing Data

Since biclustering overcomes problems of ordinary clustering on high dimensional two-way data it is not only useful in biological data analysis but also in any other field providing such data. In this chapter we use biclustering for market segmentation in tourism research. In this filed a lot of data is produced using questionnaires with binary or ordinal outcomes. We also focus on the evaluation of the performance of the biclustering algorithm comparing the Bimax algorithm to k-means and hierarchical clustering and the ordinal Quest algorithm on artificial data produced by the algorithms presented in Chapter 4. Additionally examples from shopping basket analysis and sports data are given.

6.1 Introduction

Market segmentation 'is essential for marketing success: the most successful firms drive their businesses based on segmentation' (Lilien and Rangaswamy, 2002). It enables tourism businesses and destinations to identify groups of tourists who share common characteristics and therefore make it possible to develop a tailored marketing mix to most successfully attract such subgroups of the market. Focusing on subgroups increases the chances of success within the subgroup thus improving overall survival and success chances for businesses and destinations in a highly competitive global marketplace.

The potential of market segmentation has been identified a long time ago ((Claycamp and Massy, 1968; Smith, 1956) and both tourism industry and tourism researchers continuously aim to gain more market insight to a wide range of markets through segmentation (according to Zins (2008) eight percent of publications in the Journal of Travel Research are segmentation studies) as well as to improve segmentation methods to be less prone to error and misinterpretation. One of the typical methodological challenges faced by tourism segmentation data analysts is that a large amount of information (responses to many survey questions) is available from tourists, but the sample sizes are typically too low given the number of variables used to conduct segmentation analysis (Formann, 1984). This is methodologically problematic because all

methods used to construct or identify segments (Dolnicar and Leisch, 2010) explore the data space looking for groups of respondents who are close to each other. If the data space is huge (e.g. 30-dimensional if 30 survey questions are used as the segmentation basis) and only a small number of respondents are populating this space (e.g. 400), there is simply not enough data to find a pattern reliably, resulting in a random splitting of respondents rather then the construction of managerially useful segments which can be reproduced and therefore used as a firm basis of strategy development. See also Hastie et al. (2003) for a recent discussion of the 'curse of dimensionality'.

Empirical evidence that this dimensionality problem is very serious in tourism market segmentation is provided by a review of segmentation studies (Dolnicar, 2002) which concludes that, for the 47 a posteriori segmentation studies reviewed, the variable numbers ranged from 3 to 56. At the same time the sample sizes ranged from 46 to 7996 with a median of only 461 respondents. Note that the median sample size of 461 permits the use of only 8 variables (Formann, 1984), less than the vast majority of tourism segmentation studies use.

The optimal solution to this problem is to either collect large samples that allow segmentation with a large number of variables or to conduct a series of pre-tests and include only the subset of most managerially relevant and non-redundant survey questions into the questionnaire (reducing the number of variables in the segmentation task).

Often this is not possible because, for instance, surveys are instruments designed by tourism industry representatives and the segmentation data analyst does not have the opportunity to make changes to the questionnaire. In such cases the traditional solution for the problem of large numbers of variables was to conduct so-called 'factor-cluster analysis' (Dolnicar and Grün, 2008), where the raw data is first factor analyzed and the factor scores of the resulting factors are used to compute the segmentation solution. This approach has the major disadvantage of solving one methodological problem by introducing a number of new ones: (1) the resulting segmentation solution is no longer located in the space of the original variables, but in the space of factors and can thus only be interpreted at an abstract factor level, (2) with typical percentages of variance explained of between 50 and 60%, almost half of the information that has been collected from tourists is effectively discarded before even commencing the segmentation task, (3) factor-cluster analysis has been shown to perform worse in all data situations, except in cases where the data follows exactly the factor model used with respect to revealing the correct segment membership of cases (Dolnicar and Grün, 2008), and (4) it assumes, that the factor model is the same in all segments.

This leaves the segmentation data analyst, who is confronted with a given data set with many variables (survey questions) and few cases (tourists), in the situation of having no clean statistical solution for the problem.

In this chapter we use biclustering on tourism data. In so doing, it is not necessary to eliminate variables before clustering or condensing information by means of factor analysis. The problem biclustering solves on biological data is similar to the high data dimensionality problem discussed above in the context of tourism segmentation: large numbers of genes for a small number of conditions. It therefore seems worthwhile to investigate whether biclustering can be used as a method to address the problem of high data dimensionality in data-driven segmentation of tourists.

Please note that throughout this chapter we understand the term market segmentation to mean 'dividing a market into smaller groups of buyers with distinct needs, characteristics or behaviors who might require separate products or marketing mixes' Kotler and Armstrong (2006).

6.1.1 Biclustering on Marketing Data

The starting point is a data matrix resulting from a consumer survey where the rows correspond to respondents/tourists and the columns to survey questions.The aim of biclustering here is to find segments of respondents who answered groups of questions as similar as possible to each other, and as different as possible to other respondents.

As stated above, not all questions are used for the segmentation. Instead, a subgroup of questions is identified for each segment. This subgroup of questions is selected because members of the segment responded to them in a similar way. Given the right data structure all bicluster algorithm can be adapted for application on market data. Because of the different types and structures of the outcome, it is crucial to choose the correct algorithm for the data structure and problem at hand.

For example, when tourists are asked which activities they engaged in during a vacation, responses are typically recorded in a binary format. It is therefore important that the algorithm chosen can deal with binary data. Furthermore, it is only interesting to define segments as engaging in the same activities. It is not a relevant characteristic of a segment if members have not engaged in the same vacation activities. Therefore, an algorithm needs to be chosen in this case where only positive responses are taken into consideration for the computations.

Significant differences between segments with respect to sociodemographic and other background variables that have not been used to form the groups can be

tested in the same way as they are for any clustering algorithm; biclustering does not require any specific procedures.

6.2 When to Use Biclustering

If the data analyst does not face a data dimensionality problem and results from standard techniques yield good solutions, there is no need to use biclustering. If, however, the number of variables that need to be included is too large given the sample size, or standard techniques yield diffuse results, biclustering offers a methodologically clean and managerially attractive solution for the following reasons:

6.2.1 Automatic Variable Selection

Biclustering can analyze data sets with a large number of variables because it searches for subgroups in respondents and questions and finds parts of the data where respondents display similar answer patterns across questions.

While there are no formal rules for how many variables per respondent can reasonably be grouped with exploratory clustering algorithms, the recommendation for parametric models, more specifically for latent class analysis, is to use at least 2k cases (k = number of variables), preferably 5*2k of respondents for binary data sets (Formann, 1984). This requirement would further increase if ordinal data were to be used. For the median sample size as reported in the review of segmentation studies by Dolnicar (2002) this would mean that no more than 6 variables could be included in the segmentation base. Similar rules of thumb apply for other clustering procedures, with exact numbers depending on how many parameters are to be estimated per cluster (Everitt et al., 2009; Hastie et al., 2003).

Traditional clustering algorithms weigh each piece of information equally, so responses to all survey questions are viewed as equally important in constructing a segmentation solution. However, this may not actually be desirable. The assumption underlying the factor-cluster approach, for example, is that not all survey questions are equally important and that they therefore can be condensed into factors that load on different numbers of underlying survey questions. Also, if thorough pretesting of questionnaires is not undertaken, it is very likely that some survey questions will have been included that are not actually critical to the construction of segments.

Biclustering solves this problem without data transformation. By using questions with respect to which a substantial part of the sample gave similar re-

sponses, invalid items are automatically ignored because they never demonstrate such systematic patterns. This feature of biclustering is of immense value to data analysts because they can feel confident that the inclusion of weaker, less informative items do not bias the entire segmentation results and because they do not need to rely on data preprocessing using variable selection methods before segmenting the data.

6.2.2 Reproducibility

One of the main problems with most traditional partitioning clustering algorithms as well as parametric procedures frequently used to segment markets, such as latent class analysis and finite mixture models, is that repeated computations typically lead to different groupings of respondents. This is due to the fact that consumer data are typically not well structured (Dolnicar and Leisch, 2010) and that many popular algorithms contain random components, most importantly, random selection of starting points. Biclustering results are reproducible such that every repeated computation leads to the same result. Reproducibility provides users of segmentation solutions with the confidence that the segments they choose to target really exist and are not merely the result of a certain starting solution of the algorithm. Note that one of the most popular characteristics of hierarchical clustering is its deterministic nature; however, hierarchical clustering becomes quickly unfeasible for larger data sets (e.g., dendrograms with more than 1,000 leaves are basically unreadable).

6.2.3 Identification of Market Niches

Many empirical data sets that form the basis for market segmentation are not well structured; they do not contain density clusters. Therefore, clustering algorithms do not identify naturally occurring groups of consumers, but instead construct them. Many clustering algorithms have a known tendency to group units into certain patterns (e.g., single-linkage hierarchical clustering produces chain structures, and k-means clustering tends to produce spherical groups of roughly equal size). As a consequence, it is often difficult to identify small market niches. Biclustering enables the identification of niches because the algorithm inherently looks for identical patterns among subgroups of respondents related to subgroups of questions. Niches are identified when groups with high numbers of matches are identified. A high number of matches is a strict grouping criterion, thus extracting a group with few members a market niche. A less strict criterion (fewer required matches) would lead to the identification of a larger sub-market that is less distinct.

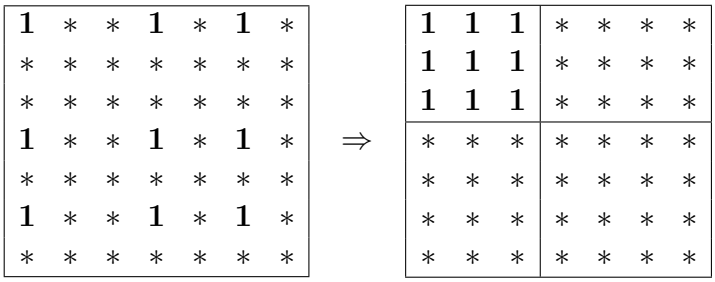

Fig. 6.1: Biclustering finds objects and variables with a similar value 1 and reports them as a bicluster (submatrix).

6.3 Binary Data

The Bimax algorithm (Prelic et al., 2006) is suitable for the example of segmenting tourists based on their vacation behavior: it searches for submatrices in a binary matrix where all entries in the identified row and column combination are one (Figure 6.1). As the original algorithm described in Chapter 5 leads to overlapping submatrices (meaning that a respondent could be assigned to multiple segments), we use the repeated Bimax algorithm, to prohibit overlapping of cluster memberships (but permitting overlapping is also possible, if preferred). Normally the algorithm takes a minimum segment size, but this minimum segment size does not have to be set. It is up to the researchers to decide whether or not to use it and how large the smallest segment size should be. For our example, we decided that a segment containing less than 5% of the population is unlikely to comply with the substantiality criterion that Philip et al. (2001) endorse for market segments, prescribing a minimum size for a segment to be worth targeting. The selection of the smallest segment size is comparable to the decision of how many segments to choose when using conventional partitioning and hierarchical clustering algorithms: it requires an assessment on the side of the data analyst.

6.3.1 Data

The data set used for this illustration is a tourism survey of adult Australians which was conducted using a permission based internet panel. Panel members were offered an incentive for completion of surveys, shown to be effective in increasing response rate (Couper, 2000; Deutskens et al., 2004). Participants were asked questions about their general travel behavior, their travel behavior on their last Australian vacation, benefits they perceive of undertaking travel, and image perceptions of their ideal tourism destination. Information was also

collected about the participants age, gender, annual household income, marital status, education level, occupation, family structure, and media consumption.

The variables used are activities that tourists engaged in during their vacation. This example is chosen for two reasons: (1) vacation activity segments are highly managerially relevant because they enable destinations or tourism providers to develop tourism products and packages to suit market segments with different activity patterns, and (2) data about vacation activities is an example of a situation where one is usually confronted with a very high number of variables that cannot be reduced without unacceptable loss of information. In the present data set 1,003 respondents were asked to state for 44 vacation activities whether they engaged in them during their last vacation. Note that according to Formann (1984), 44 binary variables would require 87,960,930,222,080 respondents to be surveyed in order to be able to run latent class analysis to identify or construct market segments.

6.3.2 Results

Biclustering results are shown in Figure 6.2 where each resulting market segment is represented by one column and each survey question (vacation activity) by one row. Black fields indicate vacation activities that all segment members have in common. The middle square of those black fields represents the mean value for this vacation activity among all respondents, ranging from 0 (white) to 1 (black) on a greyscale. The lighter the grey, the lower the level of engagement for the entire sample in a particular vacation activity, making agreement among segment members in those variables particularly interesting.

As can be seen, 11 clusters complied with the criterion of containing at least 50 respondents. This restriction can be abandoned, leading to a larger number of segments being identified. This was not done in this analysis because the 11 market segments captured 77% of the total sample. The 11 resulting segments are characterized by distinct patterns of activities. Note that, as opposed to traditional algorithms, all members of a cluster engage in all the activities that are highlighted in the chart. This makes the segments resulting from biclustering much more distinct, but has the disadvantage of being more restrictive, thus leading to segments of smaller size.

Before individual segments are interpreted it should be noted that some of the segments depicted in Figure 6.2 could also have been included in other segments but have been separated out because they have a number of additional vacation behaviors in common. For example, all members of Segment 1 (74 respondents, 7% of the sample) engage in 11 vacation activities: relaxing, eating in reasonably priced eateries, shopping, sightseeing, visiting industrial attractions (such as wineries, breweries, mines, etc.), going to markets, scenic

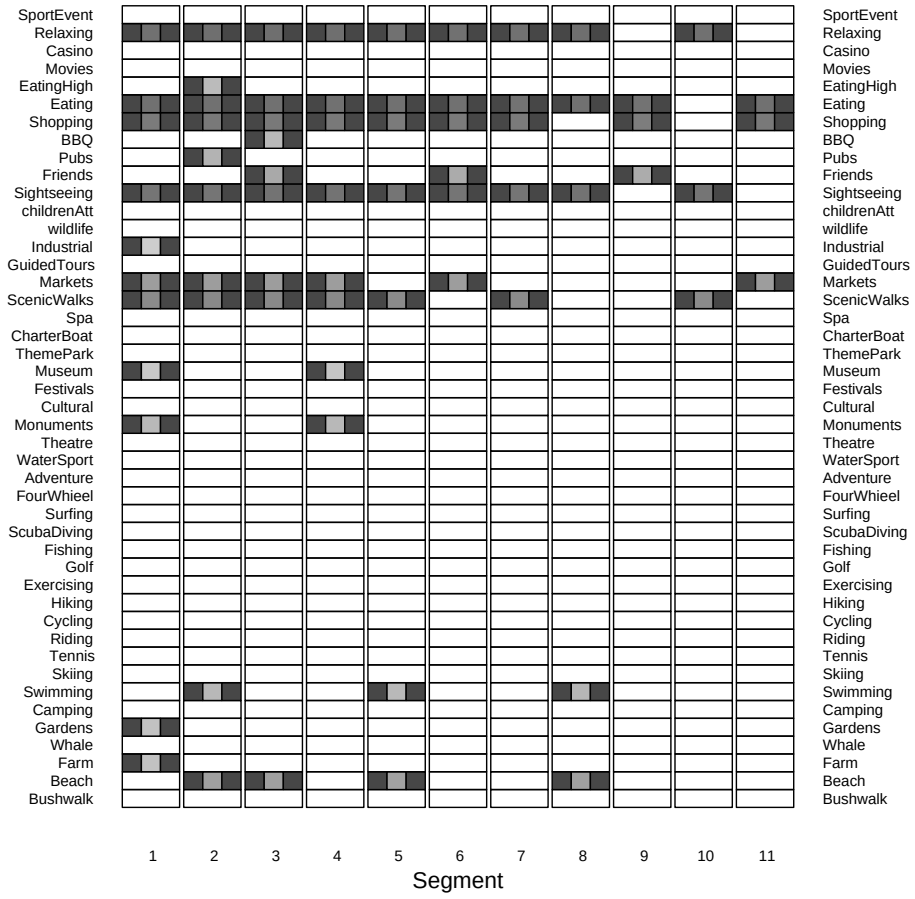

Fig. 6.2: Biclustering Plot for Vacation Activities

walks, visiting museums and monuments, botanic and public gardens, and the countryside/farms. The most characteristic vacation activities for this segment (as highlighted by the lighter grey middle section of the black bar in Figure 6.2) are visiting industrial attractions, museums, and monuments, because relatively few respondents in the total sample engage in those activities (30%, 34%, and 42%). Theoretically, Segment 1 could have been merged with Segment 11 (51 respondents, 5% of the sample), which only has three vacation activities in common (eating in reasonably priced eateries, shopping, and going to markets) to produce a larger segment containing members of both groups. This is deliberately not done because the much more distinct Segment 1 enables more targeted marketing opportunities than the more general Segment 11.

Segment 2 members (59 respondents, 6% of the sample) relax, eat in reasonably priced restaurants, shop, go sightseeing, and go to markets and on scenic walks. But they also eat in upmarket restaurants, go to pubs, go swimming, and

enjoy the beach. This latter group of variables differentiates them clearly from Segment 1. Segment 3 (55 respondents, 6% of the sample) is characterized in addition to the activities they share with one of the other two segments by going on picnics and BBQs, and visiting friends and relatives. Segments 7 (91 respondents, 9% of the sample), 9 (103 respondents, 10% of the sample), and 11 (51 respondents, 5% of the sample) are relatively generic segments, each of which could be merged with Segment 1, 2, or 3 if a larger segment is needed with fewer common vacation activities. For example, members of Segment 10 (80 respondents, 8% of the sample) only have three activities in common: relaxing, sightseeing, and going on scenic walks. Segment 4 (50 respondents, 5% of the sample) could be merged with Segment 1. It engaged in the same activities, except for not visiting public and botanic gardens and the countryside/farms. Segment 5 (75 respondents, 8% of the sample) could be merged with Segment 2. These members, as opposed to Segment 2 members, do not eat in upmarket restaurants and they do not go to pubs and markets. Segment 6 (79 respondents, 8% of the sample) is different from Segment 3 in that members of this segment do not go on picnics and BBQs, scenic walks, and to the beach. Segment 9 members have three activities in common: they all like to eat out in reasonably priced restaurants, they like to shop, and they visit friends and relatives. Finally, Segment 8 (51 respondents, 5% of the sample) members all relax, eat in reasonably priced eateries, go sightseeing, to the beach, and swim.

The segments identified by the biclustering algorithm also show external validity: they differ significantly in a number of sociodemographic and behavioral variables that were not used to construct the segments. For example, segments differ in the number of domestic holidays (including weekend getaways) they take per year (ANOVA p-value $= 0.004$). Members of Segment 2 go on the most (6.5) domestic holidays per year, closely followed by members of Segments 1 (5.8) and 10 (5.7). The fewest domestic holidays are taken by Segments 4 (3.9) and 6 (3.7). A similar pattern holds for overseas vacations (ANOVA p-value < 0.0001), with Segment 2 vacationing overseas most frequently, 1.4 times a year on average.

With respect to the number of days spent on the last domestic vacation, differences between segments are also highly significant (ANOVA p-value < 0.0001). Members of Segment 3 tend to stay longest (10.8 days on average), followed by members of Segments 2 (9.7 days) and 9 (8.4 days). Segments with particularly short average stays on their last domestic holiday include Segments 10 (5.9 days) and 11 (5.8 days).

Further, significant differences exist with respect to a number of dimensions related to travel behavior: information sources used for vacation planning, in particular tour operators (Fisher's exact test p value < 0.0001), travel agents (Fisher's exact test p-value $= 0.006$), ads in newspapers/journals (Fisher's ex-

act test p-value < 0.0001), travel guides (Fisher's exact test p-value $= 0.023$), radio ads (Fisher' s exact test p-value $= 0.031$), TV ads (Fisher's exact test p-value < 0.0001), and slide nights (Fisher's exact test p-value $= 0.002$), whether members of various segments take their vacations on weekends or during the week (Fisher's exact test p-value $= 0.001$), with or without their partner (Fisher's exact test p-value $= 0.005$), with or without an organized group (Fisher's exact test p-value $= 0.01$), whether their last domestic vacation was a packaged tour (Fisher's exact test p-value < 0.0001), whether they rented a car (Fisher's exact test p-value < 0.0001), and how many people were part of the travel party on their last domestic vacation (ANOVA p-value $= 0.031$).

Additional significant differences were revealed with respect to sociodemographics and media behavior: age (Fisher's exact test p-value $= 0.012$), level of education (Fisher's exact test p-value $= 0.031$), frequency of reading the newspaper (Fisher's exact test p-value $= 0.004$), and frequency of listening to the radio (Fisher's exact test p-value $= 0.002$).

6.3.3 Comparison with Popular Segmentation Algorithms

The aim of this section is to compare the Bimax algorithm with the two very popular algorithms in tourism segmentation: k-means clustering and Ward's clustering. A few introductory remarks are needed before this comparison is undertaken. Clustering data always leads to a result. It also leads to a result when wrong methodological decisions are made, for example, an unsuitable distance measure is used or too many variables are used given the size of the sample. Comparisons of algorithms based on final results (e.g., resulting segment profiles and descriptions) can therefore only be made if the resulting solutions from all algorithms are valid. To be valid, the following condition must be met: (1) no methodological violations must have occurred (e.g., using too many variables given a small sample size) and (2) results must be reproducible or reliable.

First we compared stability of three algorithms (Bimax, k-means, and Ward's clustering) on bootstrap samples. Then, in a second step, we produced artificial binary data with a variable percentage of ones and try to find hidden bicluster with the three methods. K-means and Ward's clustering were chosen because they have been identified as the most frequently used algorithms in tourism segmentation studies (Dolnicar, 2002).

To measure stability, we use the Adjusted Rand Index (Lawrence and Arabie, 1985). The Rand Index (Rand, 1971) takes values between 0 and 1 and is computed as $A/(A + D)$, where A is the number of all pairs of data points which are either put into the same cluster by both partitions or put into different

clusters by both partitions. D is the number of all pairs of data points that are put into one cluster in one partition, but into different clusters by the other partition. This raw index is usually adjusted for unequal cluster sizes and agreement by chance (Lawrence and Arabie, 1985). A value of one of the adjusted indices indicates identical partitions, zero agreement due to chance.

Bootstrap Samples

In this first comparison we draw 200 bootstrap samples (rows of the data are resampled with replacement) of the original data and compared the outcomes with the result on the original, unsampled data. Note that the 200 bootstrap samples are different and therefore identical segmentation results cannot emerge from the 200 repeated computations, as they would if the same original data set would be used to compute 200 repeated computations. Note also that throughout this manuscript we refer to stability in the sense of stability over repeated computation on the original or bootstrapped samples, we do not refer to stability of segments over time. Because no natural clusters (Dolnicar and Leisch, 2010) exist in the empirical data under study we constructed 12 clusters using both k-means and Ward's clusters. This number is comparable to the 11 clusters emerging from the bicluster analysis plus the ungrouped cases. All computations were made using R package `flexclust`. K-means was repeated 10 times to avoid local optima.

Figure 6.3 shows the results of the bootstrap sampled computations. Bimax significantly outperforms both k-means and Ward's clustering. The average values of the Rand index were 0.53 for biclustering, 0.42 for k-means, and 0.37 for Ward's clustering.

From this first comparison, it can be concluded that biclustering outperforms the two most popular segmentation algorithms, k-means, and Ward's clustering, with respect to its ability to produce reproducible segmentation solutions on our example dataset.

Artificial Data

To illustrate that our findings are no exception of the dataset used and universally valid, we performed a simulation study to measure the stability on artificial data. We hid 4 bicluster (submatrices containing only ones) of varying size in a 1000×50 binary matrix. We varied the percentage of one values in the noise from 30% to 40%. Note, 50% and more does not make sense, since ones should mark the values of interest. We again calculated all three algorithms on the data and compared the results to the true cluster result derived

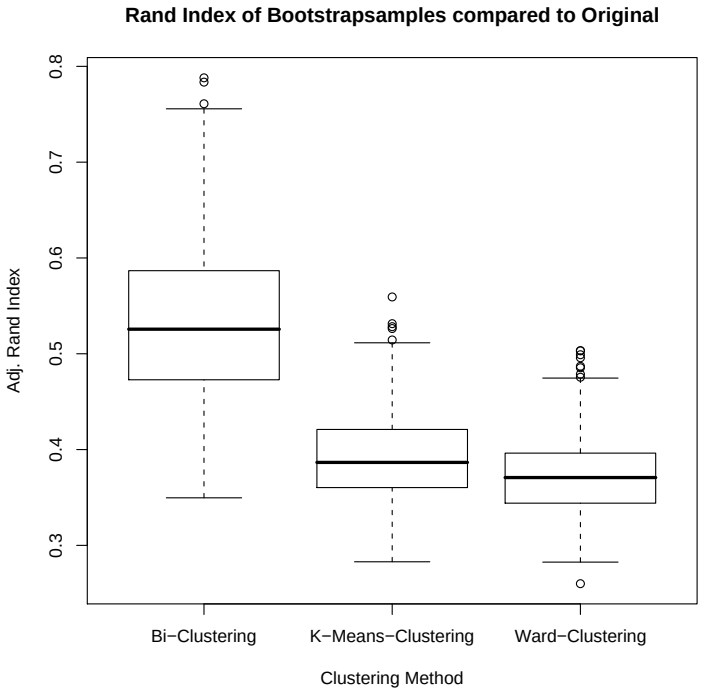

Fig. 6.3: Comparison Results for Bootstrap Sampling

from the hidden bicluster. The mean and standard deviations of the Rand index values of 1000 runs are shown in Table 6.1.

Again the Bimax algorithm outperforms the other methods by far. Very low Rand index values of k-mean and Ward's clustering appear when two or more of the hidden bicluster are very small and contain only 5% or 10% of the rows. Whenever 80% or more of the data rows were used in the hidden bicluster the k-means solution was better than the Ward clustering solution and only slightly worser than the Bimax result. A look on bootstrap runs on the artificial data show a similar picture as in the example above. Bimax results are by fare more stable than results from k-means and Ward's clustering.

Overall	Bimax	K-means	Ward
Mean	0.999	0.818	0.798
Standard Deviation	0.001	0.147	0.065
Large Cluster	Bimax	K-means	Ward
Mean	0.999	0.956	0.852
Standard Deviation	0.001	0.015	0.024

Tab. 6.1: Table of mean values and standard deviations of the Rand index using artificial data.

The results of this simulation study show, that the results of our first comparison can be generalized. On binary datasets where subgroups of ones are the reason for segmentation, biclustering outperforms k-means and Ward's clustering by means of stability and subgroup detection.

6.3.4 Shopping Basket Data

Another possible application for a binary bicluster algorithm is the segmentation of shopping basket data. We use the 'ZUMA subset' of the GFK Nürnberg household panel data (Papastefanou, 2001). The data consists of 470825 shopping baskets of 40000 households. The data is a binary coded with one if one of the 65 product categories was bought and 0 if not. The data was analyzed before, for example in Leisch and Grün (2006). Applying the repeated Bimax algorithm to this kind of data does not result in a good segmentation. The problem is that the largest bicluster has always the minimum column size allowed. This is due to too many rows compared to only 65 columns. But instead of using it as a segmentation tool one can use it to preselect association rules. Using the Bimax algorithm one is able to detected the x product categories bought together the most. For example the five products bought together the most are milk, cream cheese, quark, yogurt and hard cheese so only dairy goods. This product have been bought together 3183 times from 1169 (nearly 3 %) different households. The 10 most bought together products were Milk, coffee, tea, beer, beverage without gas, cream cheese, coffee cream quark, yogurt and hard cheese so only dairy goods and drinks. This combination was bought 67 times from 17 different households. The calculation of this most bought products is no easy task. There are 8.259.888 possible combination of 5 products and 179.013.799.328 for 10 products. So evaluating all diffrent combination would take to long an biclustering is a good assistance for such tasks.

6.4 Ordinal Data

A more challenging task is clustering ordinal data. Since the distance between items is not interpretable, our ordinal Quest approach works without a distance measure. As stated in Chapter 2 and Chapter 5 the algorithm delivers bicluster of respondents and questions. A respondent belongs to a bicluster if he answered a question within an interval of categories.

For example in Figure 6.4 this interval consists of 3 adjacent categories (ABC). In this section we will first give a data example of ordinal data in a questionnaire before we present a small simulation study on the performance of the ordinal Quest algorithm using the method of Chapter 4.

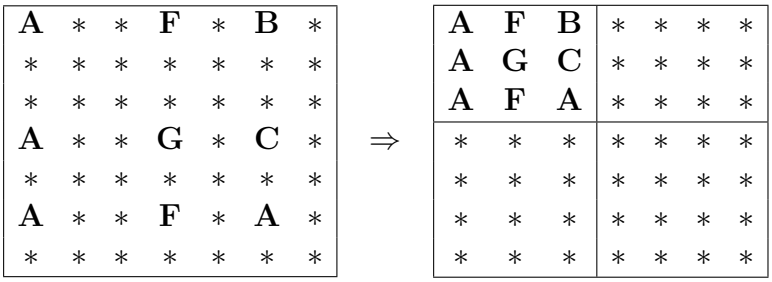

Fig. 6.4: Biclustering of Ordinal Data using the Questord Algorithm

Ordinal Data Example

For an empirical illustration we again use a survey of adult Australians from 2006. This time they were asked about their attitude toward unpaid help. Answers to 150 questions about their experience with unpaid help from 1300 respondents were collected. We concentrate on a block of 12 questions. They had to answer on a seven point scale from 'Do not believe' (0) to 'Strongly believe'(6) if giving unpaid help to an organization would enbale them to give something back to society, to socialize and meet new people, to learn new skills, and so on.

Figure 6.5 shows the barchart of the ordinal Quest result. A typical answer pattern can be seen in Cluster A. The respondents in this cluster always took the highest category 'Strongly believe'. Another finding is that the lowest category was not included in any of the bicluster (there was no 0 answer in any of the questions), but the second lowest answer is chosen on all questions in Cluster D and I. Although the algorithm is able to find bicluster if different answers on different questions only Cluster C and E show such a variation. Noe that each cluster only contains about 5 - 8% of the data, so only 55 % of the data is clustered.

Artificial Data

Again we want to test the performance of biclustering on ordinal data. In Chapter 4 we introduced a method for this purpose. With this tool we are able to determine the probabilities of the categories as well as the correlation structure. To test the algorithm we generated multiple datasets (Size 1000×100) of uncorrelated equal distributed ordinal values with 6 categories. We hid 4 bicluster in this data with different category possibilities and correlation. We used a constant correlation within a bicluster using either 0.85 or 0.95, and various different probability vectors for the categories. Additionally we sometimes added a constant bicluster to the data. Since there is no other

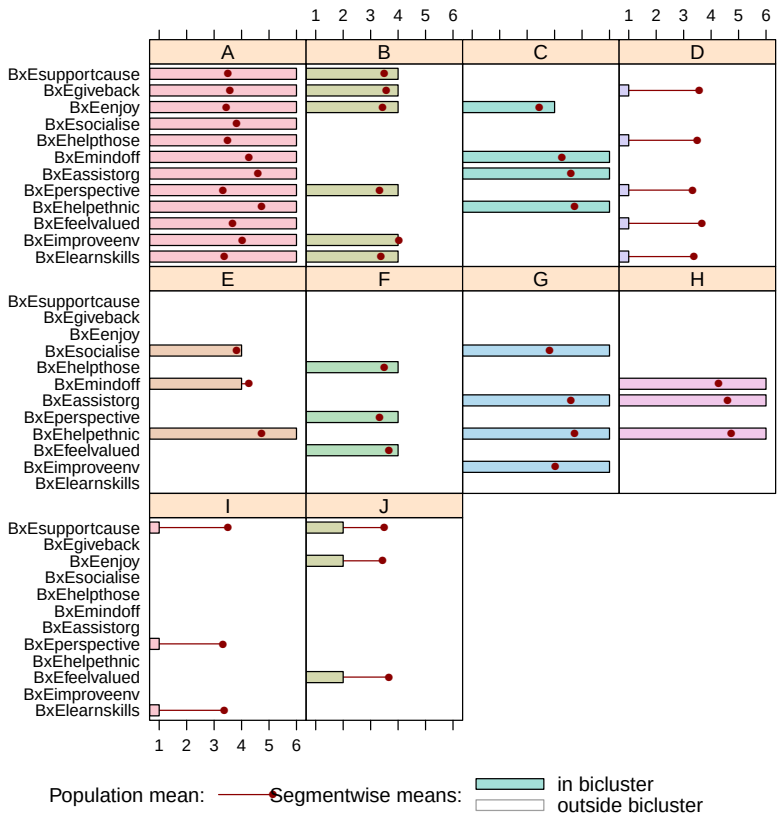

Fig. 6.5: Barchart Plot on Reward Question in the Australian Unpaid Help Survey.

bicluster algorithm defined on ordinal data so far we used the Xmotif algorithm (nominal Quest leads to the same result) to compare the results.

Table 6.2 shows the result of the Jaccard and the Rand Index of both methods compared to the hidden bicluster structure in 100 runs. The low values of the Jaccard index are as expected, since the correlation structure inside the bicluster leads to a dropping of columns. The segmentation itself measured with the Rand index is better which indicates that the rows of the bicluster are found. The Xmotif algorithms with looks for constant answers to one question performs better than the ordinal Quest algorithm. Also the Jaccard index is higher for the Xmotif algorithm. The problem of the Quest algorithm can be seen in Figure 6.6. The constant bicluster (yellow) is not perfectly found and mixed up with noise values. Also too few questions were added to each bicluster. One reason for this weak performance are the parameters used to find the bicluster. Xmotif and Quest were started with the same number of starting tries and the ordinal algorithms seems to need by far more tries.

	Rand Quest	Rand Xmotif	Jaccard Quest	Jaccard Xmotif
Mean	0.60	0.87	0.30	0.37
Standard Deviation	0.02	0.14	0.07	0.09

Tab. 6.2: Table of mean values and standard deviations of the Jaccard and the Rand index using correlated ordinal data.

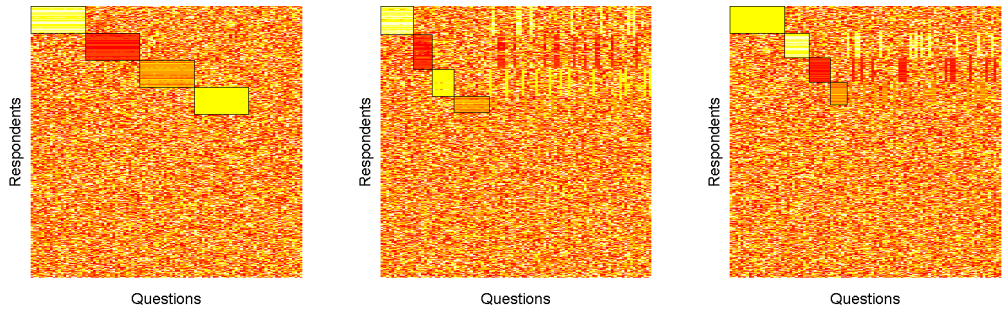

Fig. 6.6: Heatmap of bicluster results using the hidden cluster structure(left), the Xmotif algorithm (right), and the ordinal Quest algorithm (middle).

From this simulation study can be concluded that the ordinal Quest algorithm is not able to find correlation structures in the data. It only searches for loose answer patterns and is not able to detect correlated answers. Even a method which searches for constant answer patterns (Xmotifs) performs better on this data. But to find a sub-correlation- structure in such kind of data other algorithms are necessary.

6.5 Sports Data

Major sport events play an important role in our society and the analysis of sports data is becoming even more popular. Traditionally baseball is the most analyzed sport, this activity even has its own society (Society for American Baseball Research (SABR)) and publication organs (The Baseball Research Journal (BRJ), The National Pastime). Most of the data used for the calculations are two-way-datasets consisting of a large number of performance statistics for players or teams. Since a baseball team has a large number of players and since at least 40 different statistics are collected during each game or season, biclustering is a good method for clustering or segmenting this data. In this section we give an example how such an bicluster analysis can be conducted for baseball statistics and how biclustering can be easily transfered to other sports.

6.5.1 Biclustering MLB Data

We used the official statistics of the Mayor League Baseball (MLB, `http:\\www.mlb.com`) from the 2009 season to illustrate the results of biclustering on baseball data. During a season the league collects about 60 performance measures from every active players (about 40 per team) on each of the 27 teams. Our analysis concentrated on the 28 hitting stats officially collected. We removed all Pitchers and players without a plate appearance in that year. The resulting data set then included 631 players and 28 performance stats. The goal of a bicluster algorithm here was to find homogeneous players over a chosen set of performance statistics report this group together with the selected statistics. Since the stats are measured on different scales, it was important to do preprocess and scale the data. We used the Quest and Xmotif algorithms on the data to find patterns of similar performance, so we needed a discrete representation. To do so, we ranked every single statistic and recoded the data with 1 (lowest 10 per cent of the players) to 10 (highest 10 percent of the players).

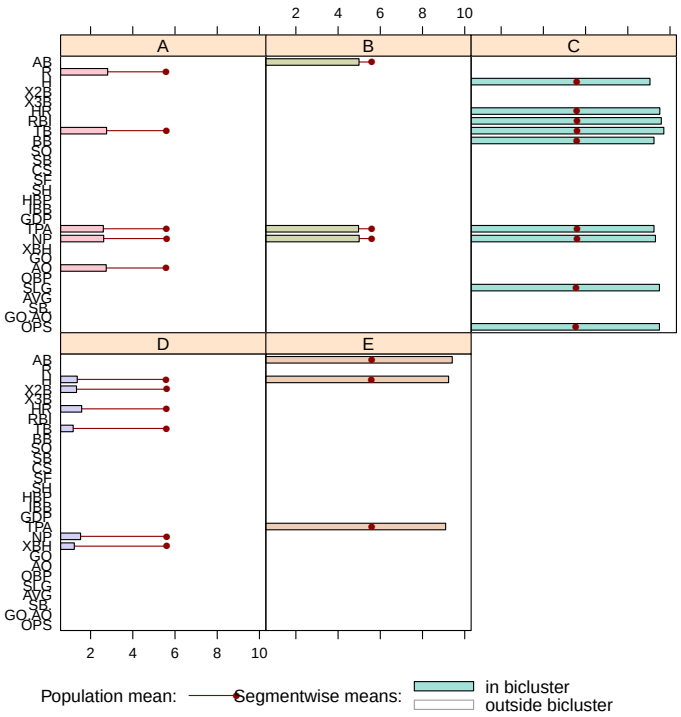

Fig. 6.7: Bicluster Barchart of an Ordinal Quest Run on MLB Hitting Statistics(2009).

Figure 6.7 shows the barchart resulting from an ordinal Quest algorithm run. Most of the time, players are grouped together when they have either low

or high values on the selected statistics. The comparison of the bicluster on the position of players inside (Figure 6.8) show the expected results. For example First Baseman and Designated Hitter are overrepresented in the third bicluster. These positions were reserved for "Power Hitter" responsible for the run production. For this reason bicluster 3 shows high values in the so-called power statistics (Runs, Home Runs, RBI and OBP). Catchers, in contrast, are underrepresented in the fourth bicluster, which shows high values in At Bats and Plate Appearances. This is obvious because the catchers job is very demanding and leads to fewer games played.

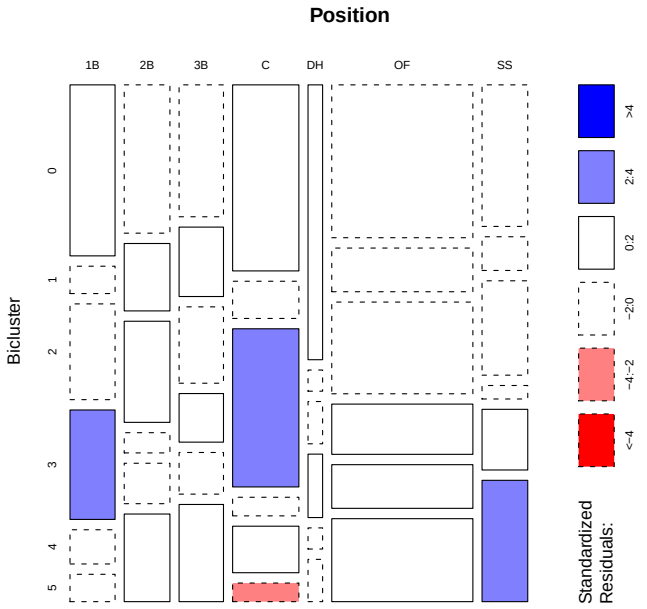

Fig. 6.8: Mosaicplot of Player Positions against Bicluster Membership.

6.5.2 Other Sport Data

In other ball sports, there is a trend of collecting team and player data and condensing it into meaningful statistics. Even in soccer, considered the most unpredictable of the popular ball games, the analysis of collected data is attracting more and more attention. At the moment, team-based analysis is the major focus in published work. For example Eugster et al. (2010) analyzed the second leg home field advantage in the UEFA Champions league). However, businesses have started to collect more individual player statistics,

paving the way for complex data analysis such as biclustering in sports other than baseball.

6.6 Conclusions

The aim of this chapter was to introduce biclustering for market segmentation analysis. The biclustering algorithm overcomes limitations of traditional clustering algorithms as well as parametric grouping algorithms. Specifically, it can deal with data containing relatively few respondents but many items per respondent, it undertakes variable selection simultaneously with grouping, it enables the identification of market niches, and its results are reproducible. The disadvantage of biclustering in the context of market segmentation is that the segments are defined in a very restrictive way (because it is expected that all segment members agree on all the variables that are characteristic for the segment). As a consequence, segments resulting from biclustering are very distinct, but small. This can be overcome by weakening the restriction that all members comply and permitting a small number of disagreements between segment members.

As shown in the empirical illustration, where 11 market segments were extracted from a survey data set based on common patterns of vacation activities, biclustering is particularly useful for market segmentation problems where the number of variables cannot be reduced. In the case of the empirical illustration presented, the variables were vacation activities. Although it is theoretically possible to merge sightseeing, visiting monuments, going to the theater, and going to museums and industrial attractions, a segmentation analysis based on such overarching variables would not provide the detail tourism destinations and tourism attractions need to identify their potential customers and develop customized vacation activity packages of communication messages for them. For instance, a tourism package aimed toward attracting tourists from Segment 1 would emphasize the cultural aspects of the destination, including any distinct industrial attractions, museums, and monuments. Package tours may be appealing to this segment if they include these types of attractions. A marketing mix highlighting a relaxing beach holiday, with the luxury of being able to eat at upmarket restaurants and frequent a pub would appeal to Segment 2. Segment 3, for instance, appears to value a more laid-back approach to eating, and prefers to partake in picnics and barbecues and gather with friends and relatives. An advertising campaign featuring nature reserves, and vacation spots near the beach with barbecue facilities would appeal to this segment's preference for outdoor dining. A comparison of each segment's distinct properties highlights the improvement in the marketing mix strategy when customizing based on a specific segment's activity preferences.

Moreover this chapter contains a comparison of the Bimax algorithm with traditional cluster algorithm k-means and Ward hierarchical clustering. Calculations on the tourism dataset and some generated artificial binary data show that Bimax outperform the two other by means of stability and segment detection.

Furthermore another application for binary bicluster algorithms, shopping basket data was introduced. Here biclustering is not used as a segmentation tool, but for retrieving useful information. In a second part of the chapter the use of biclustering on ordinal questionnaire data was demonstrated and features of the algorithms were tested on artificial data using the data generation method described in Chapter 4.

The last section of the chapter showed an example of applying bicluster on sports, especially baseball, data. The usage of biclustering was demonstrated and a outlook on future development in sports data analysis was given.

7. Summary

As high-dimensional data becomes increasingly available, suitable analysis methods are gaining in importance. Biclustering is one method specially designed to avoid the problems of traditional clustering with this type of data. By simultaneously clustering rows and columns, the size of the data matrix does not matter, and nuisance rows or columns are automatically eliminated.

This dissertation addressed methods, software, and applications of biclustering. Chapter 2 provided a theoretical background on biclustering and gave an overview of important algorithms. Since most of these algorithms are very sensitive to parameter changes or have random starting values, we developed an ensemble method, which we described in Chapter 3. By combining the results of multiple runs with one or more algorithms, we showed that it is possible to obtain more stable and reliable results. Another advantage of this method is that it can also detect an overlapping structure of bicluster. The multiple runs can be performed on the entire data set, a subsample, or a bootstrap sample. We suggested two combination methods, hierarchical and improved quality clustering, as well as two similarity measures. The simulation study showed that the improved quality approach, combined with sub sampling, obtains to the best results.

In order to test the performance of the Quest algorithm for ordinal data in Chapter 6, we introduced two methods for drawing ordinal random values with a given correlation structure. These methods perform a transformation of correlated multivariate normal distributed values to obtain the asked correlation structure. This transformation is based on a binary conversion in the first method and on the common means in the second. We also demonstrated how the methods work in brief examples. The binary method is faster, but has considerably more restrictions to the correlation matrix. The methods are available to the public through our R package `orddata`

The remaining chapters of this dissertation dealt with our software package `biclust` and with the application of biclustering to marketing data. In these sections, we demonstrated all the functions of the package and explained the theoretical background. We also included new visualization techniques such as the memberplot and the overlapping heatmap. These chapters also introduced

the newly developed algorithm Quest, which is applied to nominal, ordinal or metric data.

Chapter 6 showed how to use the package and the algorithm, and we calculated bicluster on various tourism questionnaires. We demonstrated the advantages of biclustering on this data and also showed that the Bimax method is superior to both k-means and hierarchical clustering on binary data.

Although many algorithms have already been developed for biclustering, we can expect to see the emergence of even more methods in the near future. These methods will address the special tasks of the various application fields. The next step for bicluster analysis will therefore be the development of a general validation framework.

Appendix

A. `biclust` Reference Manual

`BCBimax` *The Bimax Bicluster algorithm*

Description

Performs Bimax Biclustering based on the framework by Prelic et. al.(2006). It searches for submatrices of ones in a logical matrix. Uses the original C code of the authors.

Usage

```
## S4 method for signature 'matrix,BCBimax':
biclust(x, method=BCBimax(), minr=2, minc=2, number=100)
## S4 method for signature 'matrix,BCrepBimax':
biclust(x, method=BCrepBimax(), minr=2, minc=2, number=100,
        maxc=12)
```

Arguments

`x`	A logical matrix which represents the data.
`method`	Here BCBimax, to perform Bimax algorithm
`minr`	Minimum row size of resulting bicluster.
`minc`	Minimum column size of resulting bicluster.
`number`	Number of Bicluster to be found.
`maxc`	Maximum column size of resulting bicluster.

Value

Returns an object of class `Biclust`.

Author(s)

Sebastian Kaiser <sebastian.kaiser@stat.uni-muenchen.de>

References

Prelic, A.; Bleuler, S.; Zimmermann, P.; Wil, A.; Buhlmann, P.; Gruissem, W.; Hennig, L.; Thiele, L. & Zitzler, E. A Systematic Comparison and Evaluation of Biclustering Methods for Gene Expression Data Bioinformatics, Oxford Univ Press, 2006, 22, 1122-1129

See Also

biclust, Biclust

Examples

```
test <- matrix(rnorm(5000), 100, 50)
test[11:20,11:20] <- rnorm(100, 3, 0.1)
loma <- binarize(test,2)
res <- biclust(x=loma, method=BCBimax(), minr=4, minc=4, number=10)
res
```

BCCC *The CC Bicluster algorithm*

Description

Performs CC Biclustering based on the framework by Cheng and Church (2000). Searches for submatrices with a score lower than a specific treshold in a standardized data matrix.

Usage

```
## S4 method for signature 'matrix,BCCC':
biclust(x, method=BCCC(), delta = 1.0, alpha=1.5, number=100)
```

Arguments

`x`	Data matrix.
`method`	Here BCCC, to perform CC algorithm
`delta`	Maximum of accepted score.
`alpha`	Scaling factor.
`number`	Number of bicluster to be found.

Value

Returns an object of class `Biclust`.

Author(s)

Sebastian Kaiser `<sebastian.kaiser@stat.uni-muenchen.de>`

References

Cheng, Y. & Church, G.M. Biclustering of Expression Data Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology, 2000, 1, 93-103

See Also

`biclust`, `Biclust`

Examples

```
test <- matrix(rbinom(400, 50, 0.4), 20, 20)
res <- biclust(test, method=BCCC(), delta=1.5,  alpha=1, number=10)
res
```

BCPlaid *The Plaid Model Bicluster algorithm*

Description

Performs Plaid Model Biclustering as described in Turner et al., 2003. This is an improvement of original 'Plaid Models for Gene Expression Data' (Lazzeroni and Owen, 2002). This algorithm models data matrices to a sum of layers, the model is fitted to data through minimization of error.

Usage

```
## S4 method for signature 'matrix,BCPlaid':
biclust(x, method=BCPlaid(), cluster="b", fit.model = y ~ m + a + b,
      background = TRUE, background.layer = NA, background.df = 1,
      row.release = 0.7, col.release = 0.7, shuffle = 3,
      back.fit = 0, max.layers = 20, iter.startup = 5,
      iter.layer = 10, verbose = TRUE)
```

Arguments

`x`	The data matrix where biclusters have to be found
`method`	Here BCPlaid, to perform Plaid algorithm
`cluster`	'r', 'c' or 'b', to cluster rows, columns or both (default 'b')
`fit.model`	Model (formula) to fit each layer. Usually, a linear model is used, that stimates three parameters: m (constant for all elements in the bicluster), a(contant for all rows in the bicluster) and b (constant for all columns). Thus, default is: y ~ m + a + b.
`background`	If 'TRUE' the method will consider that a background layer (constant for all rows and columns) is present in the data matrix.
`background.layer`	If background='TRUE' a own background layer (Matrix with dimension of x) can be specified.
`background.df`	Degrees of Freedom of backround layer if background.layer is specified.
`shuffle`	Before a layer is added, it's statistical significance is compared against a number of layers obtained by random defined by this parameter. Default is 3, higher numbers could affect time performance.

`iter.startup`	Number of iterations to find starting values
`iter.layer`	Number of iterations to find each layer
`back.fit`	After a layer is added, additional iterations can be done to refine the fitting of the layer (default set to 0)
`row.release`	Scalar in [0,1](with interval recommended [0.5-0.7]) used as threshold to prune rows in the layers depending on row homogeneity
`col.release`	As above, with columns
`max.layers`	Maximum number of layer to include in the model
`verbose`	If 'TRUE' prints extra information on progress.

Value

Returns an Biclust object.

Author(s)

Adaptation of original code from Heather Turner from Rodrigo Santamaria (`<rodri@usal.es>`) and Sebastian Kaiser.

References

Heather Turner et al, "Improved biclustering of microarray data demonstrated through systematic performance tests",Computational Statistics and Data Analysis, 2003, vol. 48, pages 235-254.

Lazzeroni and Owen, "Plaid Models for Gene Expression Data", Standford University, 2002.

Examples

```
#Random matrix with embedded bicluster
test <- matrix(rnorm(5000),100,50)
test[11:20,11:20] <- rnorm(100,3,0.3)
res<-biclust(test, method=BCPlaid())
res

#microarray matrix
data(BicatYeast)
res<-biclust(BicatYeast, method=BCPlaid(), verbose=FALSE)
res
```

`BCQuest` *The Questmotif Bicluster algorithm*

Description

Performs Questmotif Biclustering a Bicluster algorithm for questionairs based on the framework by Murali and Kasif (2003). Searches subgroups of questionairs with same or similar answer to some questions.

Usage

```
## S4 method for signature 'matrix,BCQuest':
biclust(x, method=BCQuest(), ns=10, nd=10, sd=5, alpha=0.05,
        number=100)
## S4 method for signature 'matrix,BCQuestord':
biclust(x, method=BCQuestord(), d=1, ns=10, nd=10, sd=5,
        alpha=0.05, number=100)
## S4 method for signature 'matrix,BCQuestmet':
biclust(x, method=BCQuestmet(), quant=0.25, vari=1, ns=10, nd=10,
        sd=5, alpha=0.05, number=100)
```

Arguments

`x`	Data Matrix.
`method`	Here BCQuest, to perform Questmotif algorithm
`ns`	Number of questions choosen.
`nd`	Number of repetitions.
`sd`	Sample size in repetitions.
`alpha`	Scaling factor for column result.
`number`	Number of bicluster to be found.
`d`	Half margin of intervall question values should be in (Intervall is mean-d,mean+d).
`quant`	Which quantile to use on metric data
`vari`	Which varianz to use for metric data

Value

Returns an object of class `Biclust`.

Extends

Class "BiclustMethod", directly.

Author(s)

Sebastian Kaiser <sebastian.kaiser@stat.uni-muenchen.de>

References

Murali, T. & Kasif, S. Extracting Conserved Gene Expression Motifs from Gene Expression Data Pacific Symposium on Biocomputing, sullivan.bu.edu, 2003, 8, 77-88

See Also

biclust, Biclust

BCSpectral *The Spectral Bicluster algorithm*

Description

Performs Spectral Biclustering as described in Kluger et al., 2003. Spectral biclustering supposes that normalized microarray data matrices have a checkerboard structure that can be discovered by the use of svd decomposition in eigenvectors, applied to genes (rows) and conditions (columns).

Usage

```
## S4 method for signature 'matrix,BCSpectral':
biclust(x, method = BCSpectral(), normalization = "log",
      numberOfEigenvalues = 3, minr = 2, minc = 2, withinVar = 1)
```

Arguments

`x`	The data matrix where biclusters are to be found
`method`	Here BCSpectral, to perform Spectral algorithm
`normalization`	Normalization method to apply to mat. Three methods are allowed as described by Kluger et al.: "log" (Logarithmic normalization), "irrc" (Independent Rescaling of Rows and Columns) and "bistochastization". If "log" normalization is used, be sure you can apply logarithm to elements in data matrix, if there are values under 1, it automatically will sum to each element in mat (1+abs(min(mat))) Default is "log", as recommended by Kluger et al.
`numberOfEigenvalues`	the number of eigenValues considered to find biclusters. Each row (gene) eigenVector will be combined with all column (condition) eigenVectors for the first numberOfEigenValues eigenvalues. Note that a high number could increase dramatically time performance. Usually, only the very first eigenvectors are used. With "irrc" and "bistochastization" methods, first eigenvalue contains background (irrelevant) information, so it is ignored.
`minr`	minimum number of rows that biclusters must have. The algorithm will not consider smaller biclusters.
`minc`	minimum number of columns that biclusters must have. The algorithm will not consider smaller biclusters.
`withinVar`	maximum within variation allowed. Since spectral biclustering outputs a checkerboard structure despite of relevance of individual cells, a filtering of only relevant cells is necessary by means of this within variation threshold.

Value

Returns an object of class `Biclust`.

Author(s)

Rodrigo Santamaria `<rodri@usal.es>`

References

Kluger et al., "Spectral Biclustering of Microarray Data: Coclustering Genes and Conditions", Genome Research, 2003, vol. 13, pages 703-716

Examples

```
#Random matrix with embedded bicluster
test <- matrix(rnorm(5000),100,50)
test[11:20,11:20] <- rnorm(100,10,0.1)
res1 <- biclust(test, method=BCSpectral(),
                numberOfEigenvalues = 1)
res1
```

`BCXmotifs` *The Xmotifs Bicluster algorithm*

Description

Performs XMotifs Biclustering based on the framework by Murali and Kasif (2003). Searches for a submatrix where each row as a similar motif through all columns. The Algorihm needs a discret matrix to perform.

Usage

```
## S4 method for signature 'matrix,BCXmotifs':
biclust(x, method = BCXmotifs(), ns = 10, nd = 10, sd = 5,
        alpha = 0.05, number = 100)
```

Arguments

`x`	Data Matrix.
`method`	Here BCXmotifs, to perform Xmotifs algorithm
`ns`	Number of rows choosen.
`nd`	Number of repetitions.
`sd`	Sample size in repetitions.
`alpha`	Scaling factor for column result.
`number`	Number of bicluster to be found.

Value

Returns an object of class `Biclust`.

Extends

Class "BiclustMethod", directly.

Author(s)

Sebastian Kaiser <sebastian.kaiser@stat.uni-muenchen.de>

References

Murali, T. & Kasif, S. Extracting Conserved Gene Expression Motifs from Gene Expression Data Pacific Symposium on Biocomputing, sullivan.bu.edu, 2003, 8, 77-88

See Also

biclust, Biclust

Examples

```
data(BicatYeast)
x<-discretize(BicatYeast)
res <- biclust(x, method = BCXmotifs(), ns = 20, nd = 20, sd = 5,
               alpha = 0.01, number = 10)
res
```

BicatYeast *BicAT Yeast*

Description

Microarray data matrix for 80 experiments with Saccharomyces Cerevisiae organism extracted from BicAT example data set.

Usage

```
data(BicatYeast)
```

Format

Data structure with information about the expression levels of 419 probesets over 70 conditions Row names follow Affymetrix probeset notation

Source

BicAT datasets at http://www.tik.ee.ethz.ch/sop/bicat/

Biclust-class *The Biclust Class*

Description

Biclust is the class structure for results of a bicluster algorithm. It contains all information needed for further processing. The `show` Method gives the Name of the Algorithm used and the first Bicluster found. The `summary` Method gives sizes of all bicluster found.

Objects from the Class

Objects can be created by performing a bicluster algorithm via the `biclust()` function.

Slots

Objects of class `Biclust` have the following slots:

`Parameters:` Saves input Parameters in a list

`RowxNumber:` Logical Matrix which contains 1 in [i,j] if Row i is in Bicluster j

`NumberxCol:` Logical Matrix which contains 1 in [i,j] if Col j is in Bicluster i

`Number:` Number of Bicluster

`info:` Additional Outputs from the different bicluster algorithms

Details

`RowxNumber` and `NumberxCol` are named after the arrangement of the data they contain. The column results are transposed in order to ensure a easy processing.

Author(s)

Sebastian Kaiser <sebastian.kaiser@stat.uni-muenchen.de>

See Also

biclust, BiclustMethod-class

BiclustMethod-class

The BiclustMethod Virtual Class

Description

BiclustMethod is the virtual class structure for algorithms provided in the package. In order to use the biclust() function a algorithm has to have a class inherit from here.

Algorithms

Currently 6 classes inherit from BiclustMethod: BCCC, BCXmotifs, BCPlaid, BCSpectral, BCBimax, BCQuest

Author(s)

Sebastian Kaiser <sebastian.kaiser@stat.uni-muenchen.de>

See Also

biclust, Biclust-class, BCCC, BCXmotifs, BCPlaid, BCSpectral, BCBimax, BCQuest, BiclustMethod-class

`biclust` *The biclust Method*

Description

The function `biclust` is the main function of the package. It calculates the bicluster in a data matrix using the algorithm specified in the method-argument. Currently the package contains 5 different methods for the use in `biclust`. For each algorithm see the class help files for further details. For some algorithms preproccessing is necessary, e.g. `BCBimax` only runs with a logical matrix.

Usage

```
## S4 method for signature 'matrix,BiclustMethod':
biclust(x,method,...)

## S4 method for signature 'matrix,character':
biclust(x,method,...)
```

Arguments

`x`	Data matrix.
`method`	An object of class `"BiclustMethod"` or a character string with the name of a `"BiclustMethod"`-class.
`...`	Additional Parameters of the `"BiclustMethod"`

Value

Returns an object of class `Biclust`.

Author(s)

Sebastian Kaiser `<sebastian.kaiser@stat.uni-muenchen.de>`

See Also

`Biclust-class`, `BCCC`, `BCXmotifs`, `BCPlaid`, `BCSpectral`, `BCBimax`, `BCQuest`, `BiclustMethod-class`

Examples

```
test <- matrix(rbinom(400, 50, 0.4), 20, 20)
res <- biclust(test, method=BCCC(), delta=1.5,  alpha=1, number=10)
```

`biclustbarchart` *Bicluster Barchart*

Description

Draws a barchart for a Bicluster result representing the columns

Usage

```
biclustbarchart(x, Bicres, which=NULL, ...)
```

Arguments

`x`	The data matrix
`Bicres`	BiclustResult object with a bicluster result set. If this value is set to NULL, the data matrix is drawn as a heatmap, without any reordering. Default NULL.
`which`	If specified gives the ploting order of the columns from bottom to top
`...`	Additional plot options passed to barchart

Author(s)

Sebastian Kaiser `<sebastian.kaiser@stat.uni-muenchen.de>`

See Also

`bubbleplot` for simultaneous representation of biclusters, `parallelCoordinates` for single representation of biclusters as lines of gene or condition profiles, `drawHeatmap` for Heatmap representation of biclusters and `biclustmember` for a membership graph.

Examples

```
set.seed(1)
x=matrix(rnorm(900),30,30)
x[1:5,1:5]=rnorm(25,3,0.3)
x[11:15,11:15]=rnorm(25,-3,0.3)
x[21:25,21:25]=rnorm(25,6,0.3)
colnames(x)<-paste("Var.",1:30)
bics <- biclust(x,BCPlaid(), back.fit = 2, shuffle = 3,
fit.model = ~m + a + b, iter.startup = 5, iter.layer = 30,
verbose = TRUE)
biclustbarchart(x,bics, col="#A3E0D8")
ord<-bicorder(bics, cols=TRUE, rev=TRUE)
biclustbarchart(x,bics,which=ord)
```

---

bicluster *Extract Bilcuster*

Description

Function to extract the bicluster or the row and column numbers from a given bicluster result

Usage

```
bicluster(x, BicRes, number= 1:BicRes@Number)
biclusternumber(BicRes, number= 1:BicRes@Number)
```

Arguments

`x`	The data matrix
`BicRes`	BiclustResult object
`number`	Which bicluster to be extracted

Value

Returns a list containing all extracted bicluster

Author(s)

Sebastian Kaiser <sebastian.kaiser@stat.uni-muenchen.de>

See Also

writeclust,writeBiclusterResults

Examples

```
s2=matrix(rnorm(400),20,20)
s2[12:16,12:16]=rnorm(25,3,0.3)
set.seed(1)
bics <- biclust(s2,BCPlaid(), back.fit = 2, shuffle = 3,
                fit.model = ~m + a + b, iter.startup = 5,
                iter.layer = 30,  verbose = TRUE)
bicluster(s2, bics)
biclusternumber(bics)
```

---

bimax.grid *Parameter Grid for BCBimax Biclustering*

Description

Generates a list containing parameter settings for the ensemble algorithm.

Usage

```
bimax.grid(method = "BCBimax", minr = c(10, 11), minc = c(10, 11),
           number = 10)
```

Arguments

method	Here BCBimax, to perform Bimax algorithm
minr	Minimum row size of resulting bicluster.
minc	Minimum column size of resulting bicluster.
number	Number of Bicluster to be found.

Value

A list containing parameter settings

Author(s)

Sebastian Kaiser <sebastian.kaiser@stat.uni-muenchen.de>

See Also

ensemble, BCBimax

Examples

```
bimax.grid()
```

---

binarize *Binarize*

Description

Methods to convert a real matrix to a binary matrix.

Usage

```
binarize(x, threshold=NA)
binarizeByPercentage(x,percentage, error=0.2, gap=0.1)
densityOnes(x)
```

Arguments

x	The data matrix to be binarized.
threshold	Threshold used to binarize. Values over threshold will be set to 1, the rest to 0. If threshold is NA, median is used as threshold. Default NA.
percentage	Percentage of ones against zeros desired in the binary matrix.
error	Percentage of ones against zeros in the final matrix will be in [percentage-error, percentage+error]. Default 0.2
gap	Value used for incremental search of threshold. Default 0.1

Details

The `binarize` function returns a matrix binarized by input threshold, or by the median if no threshold is given.

The `binarizeByPercentage` function returns a matrix binarize by input percentage, given as desired density of ones against zeros.

The `densityOnes` function returns the percentage of ones against zeros in a logical matrix

Author(s)

Rodrigo Santamaria `<rodri@usal.es>` and Sebastian Kaiser

Examples

```
data(BicatYeast)
m1=binarize(BicatYeast)
m2=binarize(BicatYeast, 0.2)
m3=binarizeByPercentage(BicatYeast, 5)
densityOnes(m3)
densityOnes(m2)
densityOnes(m1)
drawHeatmap(BicatYeast)
drawHeatmap(m1)
drawHeatmap(m2)
drawHeatmap(m3)
```

`bubbleplot` *Bubbleplot*

Description

Draws a bubble plot where each bicluster is represented as a circle (bubble). Color represents the bicluster set to which bicluster pertains (up to three bicluster sets can be represented simultaneously). Brightness representes the bicluster homogeneity (darker, less homogeneous). Size represents the size of the bicluster, as (number of genes)x(number of conditions). Location is a 2D-projection of gene and condition profiles.

Usage

```
bubbleplot(x, bicResult1, bicResult2 = NULL, bicResult3 = NULL,
           projection = "mean", showLabels = FALSE)
```

Arguments

`x`	The data matrix from which biclusters were identified.
`bicResult1`	BiclustResult object with a bicluster result set whose biclusters will be drawn in green.
`bicResult2`	BiclustResult object with an optional second bicluster result set. Will be drawn in red (default NULL)
`bicResult3`	BiclustResult object with an optional third bicluster result set. Will be drawn in blue (default NULL)
`projection`	Projection algorithm used to position bubbles. Allowed projections are 'mean', 'isomds' and 'cmdscale' (default 'mean'). See details section for a broader explanation.
`showLabels`	If 'TRUE', puts a label over each bubble that tells the number within the corresponding bicluster result (default 'FALSE').

Details

Position of circles depend on a 2D projection of the multidimensional point formed by rows and columns present in the bicluster. For example, if we have a 3x3 matrix to analyze and we find a bicluster with rows 1 and 3 and columns 2 and 3, the corresponding multidimensional point will be p=(1,0,1,0,1,1). For this example, 'mean' projection will map the bicluster with the point x=(1+3)/2=2 and y=(2+3)/2=2,5. Other projections will take the point p and project it following the corresponding algorithms (see the corresponding help pages for details)

Note

Bubbleplot 2D-projection, as any multidimensional scaling, loses information, trying to take the main relationships and trends of n-dimensional data. Thus, locations and intersections between bubbles-biclusters are only an estimate of its similarity. This visualization should be used just as a help to understand overall behavior of biclustering methods, detect trends and outliers, etc.

Author(s)

Rodrigo Santamaria <rodri@usal.es>

See Also

`drawHeatmap` for single representation of biclusters inside data matrix, `parallelCoordinates` for single representation of biclusters as lines of gene or condition profiles, `cmdscale, isomds` for multidimensional scaling and `plot` for other point representations.

Examples

```
#Simplified yeast microarray data
## Not run:
data(BicatYeast)
set.seed(1)
bics1 <- biclust(BicatYeast,BCPlaid(), back.fit = 2,
            shuffle = 3, fit.model = ~m + a + b,
            row.release = 0.7, col.release = 0.7,
            verbose = FALSE, max.layers = 10,
            iter.startup = 5, iter.layer = 30)
bubbleplot(BicatYeast,bics1, showLabels=TRUE)

loma=binarize(BicatYeast,2)
bics2=biclust(loma,BCBimax(), minr=4, minc=4, number=10)
bubbleplot(BicatYeast,bics1,bics2)
## End(Not run)
```

ChiaKaruturi *Chia and Karuturi Function*

Description

Function computing scores as described in the paper of Chia and Karuturi (2010)

Usage

```
ChiaKaruturi(x, bicResult, number)
```

Arguments

`x`	Data Matrix
`bicResult`	`Biclust` object from `biclust` package
`number`	Number of bicluster in the output for computing the scores

Details

The function computes row (T) and column (B) effects for a chosen bicluster. The scores for columns within bicluster have index 1, the scores for columns outside the bicluster have index 2. Ranking score is SB, stratification score is TS.

Value

Data.Frame with 6 slots: T, B scores for within and outside bicluster, SB and TS scores

Author(s)

Tatsiana Khamiakova `<tatsiana.khamiakova@uhasselt.be>` and Sebastian Kaiser

References

Chia, B. K. H. and Karuturi, R. K. M. (2010) Differential co-expression framework to quantify goodness of biclusters and compare biclustering algorithms. Algorithms for Molecular Biology, 5, 23.

See Also

`diagnosticPlot`, `computeObservedFstat`, `diagnoseColRow`

Examples

```
#---simulate dataset with 1 bicluster ---#
xmat<-matrix(rnorm(20*50,0,0.25),50,50) # background noise only
rowSize <- 20 #number of rows in a bicluster
colSize <- 10 #number of columns in a bicluster
a1<-rnorm(rowSize,1,0.1) #sample row effect from N(0,0.1)
#adding a coherent values bicluster:
b1<-rnorm((colSize),2,0.25)  #sample column effect from N(0,0.05)
mu<-0.01 #constant value signal
 for ( i in 1 : rowSize){
  for(j in 1: (colSize)){
  xmat[i,j] <- xmat[i,j] + mu + a1[i] + b1[j]
  }
 }
 #--obtain a bicluster by running an algorithm---#
plaidmab <- biclust(x=xmat, method=BCPlaid(), cluster="b",
```

```
                    fit.model = y ~ m + a+ b, background = TRUE,
                    row.release = 0.6, col.release = 0.7,
                    shuffle = 50, back.fit = 5, max.layers = 1,
                    iter.startup = 100, iter.layer = 100)

#Get Chia and Karuturi scores:
ChiaKaruturi(x=xmat, bicResult = plaidmab, number = 1)
```

`biclustmember` *Bicluster Membership Graph*

Description

Draws a membership graph cluster x columns

Usage

```
biclustmember(bicResult, x, mid = T, cl_label = "", which = NA,
            main = "BiCluster Membership Graph", xlab = "Cluster",
            color = diverge_hcl(101, h = c(0, 130)), ...)

clustmember(res, x, mid = T, cl_label = "", which = NA,
          main = "Cluster Membership Graph", xlab="Cluster",
          color = diverge_hcl(101, h = c(0, 130)), ...)

bicorder(bicResult, cols=TRUE, rev=FALSE)
```

Arguments

`x`	The data matrix
`bicResult`	BiclustResult object with a bicluster result set.
`res`	Cluster Result (is converted into a kcca object)
`mid`	If TRUE, shows the value of the remaining objects inside the cluster value, else shows both aside each other.
`cl_label`	Ticks of x-axis
`which`	If specified gives the ploting order of the columns from bottom to top
`main`	Gives the title of the plot
`xlab`	Label of x-axis

color	Range of colors for the plot
...	Additional plot options or if neccessary option for as.kcca
cols	If TRUE orders the column by appearance in the bicluster, else orders the rows.
rev	If TRUE reverses the order

Author(s)

Sebastian Kaiser <sebastian.kaiser@stat.uni-muenchen.de>

See Also

bubbleplot for simultaneous representation of biclusters, parallelCoordinates for single representation of biclusters as lines of gene or condition profiles, drawHeatmap for Heatmap representation of biclusters and biclustbarchart for a barchart.

Examples

```
set.seed(1)
x=matrix(rnorm(900),30,30)
x[1:5,1:5]=rnorm(25,3,0.3)
x[11:15,11:15]=rnorm(25,-3,0.3)
x[21:25,21:25]=rnorm(25,6,0.3)
colnames(x)<-paste("Var.",1:30)
bics <- biclust(x,BCPlaid(), back.fit = 2, shuffle = 3,
                fit.model = ~m + a + b, iter.startup = 5,
                iter.layer = 30,  verbose = TRUE)
biclustmember(bics,x)
ord<-bicorder(bics, cols=TRUE, rev=TRUE)
biclustmember(bics,x,which=ord)
```

---

coherence *Coherence measures*

Description

Different prelilminary measures of how much constant or (additive, multiplicative, sign) coherent a bicluster is, following Madeira and Oliveira classification of biclusters.

Usage

```
constantVariance(x, resultSet, number, dimension="both")
additiveVariance(x, resultSet, number, dimension="both")
multiplicativeVariance(x, resultSet, number, dimension="both")
signVariance(x, resultSet, number, dimension="both")
```

Arguments

`x`	The data matrix from which biclusters were identified
`resultSet`	BiclustResult object with a bicluster result set where is the bicluster to measure
`number`	Number of the bicluster withing the result set
`dimension`	"both" for determining overall variance, "row" for gene variance and "col" for column variance. Default "both"

Details

Returns the corresponding variance of genes or conditions as the average of the sum of euclidean distances between all rows and/or columns of the bicluster. For additive, multiplicative and sign variance first a transformation of the bicluster is done, so variance is computed on a matrix that reflects difference, rest or change of sign between rows, columns or both.

The lower the value returned, the more constant or coherent the bicluster is. If the value returned is 0, the bicluster is ideally constant or coherent. Usually, a value above 1-1.5 is enough to determine the bicluster is not constant or coherent.

Note

There are preliminary measures for coherence. Since transformations are different, measures are not normalized and comparison between, for example, additive and multiplicative variance is not meaningful. Only comparisons between different measures of the same kind of variance are reliable by now.

Author(s)

Rodrigo Santamaria `<rodri@usal.es>`

Examples

```
#Simplified yeast microarray data
data(BicatYeast)
```

```
set.seed(1)
bics1 <- biclust(BicatYeast,BCPlaid(), back.fit = 2,
            shuffle = 3, fit.model = ~m + a + b,
            row.release = 0.7, col.release = 0.7,
            verbose = FALSE, max.layers = 10,
            iter.startup = 5, iter.layer = 30)

constantVariance(BicatYeast, bics1,1,"row")
constantVariance(BicatYeast, bics1,1,"col")
constantVariance(BicatYeast, bics1,1,"both")
additiveVariance(BicatYeast, bics1,1,"both")
multiplicativeVariance(BicatYeast, bics1,1,"both")
signVariance(BicatYeast, bics1,1,"both")
```

`computeObservedFstat`

Diagnostic F Statistic Calculation

Description

Functions for obtaining F statistics within bicluster and the significance levels. The main effects considered are row, column and interaction effect.

Usage

```
computeObservedFstat(x, bicResult, number)
```

Arguments

`x`	Data Matrix
`bicResult`	`Biclust` object from `biclust` package
`number`	Number of bicluster in the output for computing observed statistics

Details

F-statistics are calculated from the two-way ANOVA mode with row anc column effect. The full model with interaction is undentifiable, thus, Tukey's test for non-additivity is used to detect an interaction within a bicluster. p-values are obtained from assymptotic F distributions.

Value

Data frame with three rows ("Row Effect", "Column Effect", "Tukey test") and 2 columns for corresponding statistics (Fstat) and their p-values (PValue). 2

Author(s)

Tatsiana Khamiakova <tatsiana.khamiakova@uhasselt.be> and Sebastian Kaiser

See Also

ChiaKaruturi, diagnoseColRow

Examples

```
#---simulate dataset with 1 bicluster ---#
xmat<-matrix(rnorm(20*50,0,0.25),50,50) # background noise only
rowSize <- 20 #number of rows in a bicluster
colSize <- 10 #number of columns in a bicluster
a1<-rnorm(rowSize,1,0.1) #sample row effect from N(0,0.1)
#adding a coherent values bicluster:
b1<-rnorm((colSize),2,0.25)  #sample column effect from N(0,0.05)
mu<-0.01 #constant value signal
 for ( i in 1 : rowSize){
  for(j in 1: (colSize)){
  xmat[i,j] <- xmat[i,j] + mu + a1[i] + b1[j]
  }
 }
 #--obtain a bicluster by running an algorithm---#
plaidmab <- biclust(x = xmat, method = BCPlaid(), cluster = "b",
               fit.model = y ~ m + a+ b, background = TRUE,
               row.release = 0.6, col.release = 0.7,
               shuffle = 50, back.fit = 5, max.layers = 1,
               iter.startup = 100, iter.layer = 100)

#Calculate statistics and their p-values to infer about
#the structure within bicluster:
Structure <- computeObservedFstat(x=xmat, bicResult = plaidmab,
                number = 1)
```

`diagnoseColRow` *Bootstrap Procedure for Bicluster Diagnostics*

Description

Calculate the signifiance of the discovered patter in the data based on the bootstrapping procedure.

Usage

```
diagnoseColRow(x, bicResult, number, nResamplings, replace = TRUE)
```

Arguments

`x`	data matrix, which `biclust` function was applied to
`bicResult`	object of class `biclust`, containing result of a biclustering algorithm
`number`	number of bicluster from the output for the diagnostics
`nResamplings`	number of bootstrap replicates
`replace`	logical flag for bootstrap (TRUE), or sampling without replacement (FALSE)

Details

The function computes observed F statistics for row and column effect based on two-way ANOVA model. Bootstrap procedure is used to evaluate the significance of discovered bicluster. Based on `nResamplings` replicates, the disribution of F statistics for row and column effects are obtained. The p-value is computed as

$$P(A) = \frac{\#\left\{F^*(A)_b > F(A)^{obs}\right\}}{nResamplings + 1}$$

Low p-values denote non-random selection of columns for a given bicluster. Large p-values show that in other columns for a given set of genes in the bicluster structure is similar. Hence, bicluster columns were just randomly picked by an algorithm for a set of co-regulated genes.

Value

`bootstrapFstats`
: matrix with two columns, containing values of bootstrap F-statistics. The first column corresponds to row, the second column corresponds to column.

`observedFstatRow`
: observed F-statistics for the row effect

`observedFstatCol`
: observed F-statistics for the column effect

`bootstrapPvalueRow`
: bootstrap p value for row effect

`bootstrapPvalueCol`
: bootstrap p value for column effect

Author(s)

Tatsiana Khamiakova `<tatsiana.khamiakova@uhasselt.be>` and Sebastian Kaiser

See Also

`diagnosticPlot`, `computeObservedFstat`, `ChiaKaruturi`

Examples

```
#---simulate dataset with 1 bicluster ---#
xmat<-matrix(rnorm(20*50,0,0.25),50,50) # background noise only
rowSize <- 20 #number of rows in a bicluster
colSize <- 10 #number of columns in a bicluster
a1<-rnorm(rowSize,1,0.1) #sample row effect from N(0,0.1)
#adding a coherent values bicluster:
b1<-rnorm((colSize),2,0.25)  #sample column effect from N(0,0.05)
mu<-0.01 #constant value signal
 for ( i in 1 : rowSize){
  for(j in 1: (colSize)){
  xmat[i,j] <- xmat[i,j] + mu + a1[i] + b1[j]
  }
 }
 #--obtain a bicluster by running an algorithm---#
plaidmab <- biclust(x = xmat, method = BCPlaid(), cluster = "b",
               fit.model = y ~ m + a+ b, background = TRUE,
               row.release = 0.6, col.release = 0.7,
               shuffle = 50, back.fit = 5, max.layers = 1,
               iter.startup = 100, iter.layer = 100)
```

```
#Run boosotrap procedure:
Bootstrap <- diagnoseColRow(x=xmat, bicResult = plaidmab,
                number = 1, nResamplings = 999, replace = TRUE)
#plotting distribution of bootstrap replicates
diagnosticPlot(bootstrapOutput = Bootstrap)
```

`diagnosticPlot` *Diagnostic F Statistics Visualization*

Description

Plots distributions of bootstrap replicates of F-statistics for row and column effect and highlights the observed statistics

Usage

```
diagnosticPlot(bootstrapOutput)
```

Arguments

`bootstrapOutput`
: output of `diagnoseColRow` function, containing bootstrap replicates and observed F-statistics

Value

No value is returned. The plot is constructed in a current device.

Author(s)

Tatsiana Kahamiakova `<tatsiana.khamiakova@uhasselt.be>` and Sebastian Kaiser

See Also

`diagnoseColRow`, `computeObservedFstat`

Examples

```
#---simulate dataset with 1 bicluster ---#
xmat<-matrix(rnorm(20*50,0,0.25),50,50) # background noise only
rowSize <- 20 #number of rows in a bicluster
colSize <- 10 #number of columns in a bicluster
a1<-rnorm(rowSize,1,0.1) #sample row effect from N(0,0.1)
#adding a coherent values bicluster:
b1<-rnorm((colSize),2,0.25)  #sample column effect from N(0,0.05)
mu<-0.01 #constant value signal
 for ( i in 1 : rowSize){
  for(j in 1: (colSize)){
  xmat[i,j] <- xmat[i,j] + mu + a1[i] + b1[j]
  }
 }
 #--obtain a bicluster by running an algorithm---#
plaidmab <- biclust(x = xmat, method = BCPlaid(), cluster = "b",
               fit.model = y ~ m + a+ b, background = TRUE,
               row.release = 0.6, col.release = 0.7,
               shuffle = 50, back.fit = 5, max.layers = 1,
               iter.startup = 100, iter.layer = 100)

#Run bootsrap procedure:
Bootstrap <- diagnoseColRow(x=xmat, bicResult = plaidmab,
                number = 1, nResamplings = 999, replace = TRUE)

# plotting distribution of bootstrap replicates
diagnosticPlot(bootstrapOutput = Bootstrap)
```

---

discretize *Create a discret matrix*

Description

Some biclusteralgorithms need a discret matrix to perform well. This function delivers a discret matrix with either a given number of levels of equally spaced intervals from minimum to maximum, or levels of same size using the quantiles.

Usage

```
discretize(x,nof=10,quant=FALSE)
```

Arguments

x	The data matrix from which should be dicretized
nof	Number of levels
quant	If TRUE using the quantiles, else using equally spaced levels

Author(s)

Sebastian Kaiser <sebastian.kaiser@stat.uni-muenchen.de>

Examples

```
#Discretize yeast microarray data
data(BicatYeast)
discretize(BicatYeast[1:10,1:10])
```

---

drawHeatmap *Draw Heatmap*

Description

Draws a microarray data matrix as a heatmap, with rows and colums reordered so the rows and columns of the input bicluster will be at top-left of the matrix.

Usage

```
drawHeatmap(x, bicResult = NULL, number = NA, local = TRUE,
            beamercolor = FALSE, paleta, ...)
drawHeatmap2(x, bicResult = NULL, number = NA, plotAll = FALSE)
```

Arguments

x	The data matrix where the bicluster is to be drawn.
bicResult	BiclustResult object with a bicluster result set. If this value is set to NULL, the data matrix is drawn as a heatmap, without any reordering. Default NULL.

number	Bicluster to be drawn from the result set 'bicResult'. If bicResult is set to NULL, this value is ignored. Default NA
local	If TRUE, only rows and columns of the bicluster were drawn.
plotAll	If TRUE, all Bicluster of result set 'bicResult' were drawn.
beamercolor	If TRUE, palete colors are used.
paleta	Colors
...	Additional plot options

Details

'plotAll' only works if there is a exclusive rows and column Result!

Author(s)

Rodrigo Santamaria <rodri@usal.es>, Sebastian Kaiser

See Also

bubbleplot for simultaneous representation of biclusters and parallelCoordinates for single representation of biclusters as lines of gene or condition profiles.

Examples

```
#Random 100x50 matrix with a single, up-regulated 10x10 bicluster
s2=matrix(rnorm(5000),100,50)
s2[11:20,11:20]=rnorm(100,3,0.3)
set.seed(1)
bics <- biclust(s2,BCPlaid(), back.fit = 2, shuffle = 3,
                fit.model = ~m + a + b, iter.startup = 5,
                iter.layer = 30,  verbose = TRUE)
drawHeatmap(s2,bics,1)
```

`EisenYeast` *Eisen Yeast*

Description

Microarray data matrix for 80 experiments with Saccharomyces Cerevisiae organism by Eisen Lab.

Usage

```
data(EisenYeast)
```

Format

Data frame with information about the expression levels of 6221 genes over 80 conditions. Missing values have been imputed unsin k-nearest neighbor averaging implemented in impute.knn() from library 'impute' (using default k=10). Gene names follow ORF (Open Reading Format) notation.

Source

Eisen Lab at http://rana.lbl.gov/EisenData.htm

`ensemble` *Ensemble Methods for Bicluster Algorithms*

Description

Calculates an ensemble of biclusters from different parameter setting of possible different bicluster algorithms.

Usage

```
ensemble(x, confs, rep = 1, maxNum = 5, similar = jaccard2,
         thr = 0.8, simthr =0.7, subs = c(1, 1),
         bootstrap = FALSE, support = 0, combine=firstcome, ...)
```

Arguments

x	Data Matrix
confs	Matrix containing parameter sets
rep	Number of repetitions for each parameter set
maxNum	Maximum number of biclusters taken from each run
similar	Function to produce a similarity matrix of bicluster
thr	Threshold for similarity
simthr	Proportion of row column combinations in bicluster
subs	Vector of proportion of rows and columns for subsampling. Default c(1,1) means no subsampling.
bootstrap	Should bootstrap sampling be used (logical: replace=bootstrap).
support	Wich proportion of the runs must contain the bicluster to have enough support to report it (between 0 and 1).
combine	Function to combine the single bicluster only firstcome and hcl for hierarchical clustering are possible at the moment.
...	Arguments past to the combine function.

Details

Two different kinds (or both combined) of ensebmbling is possible. Ensemble of repeated runs or ensemble of runs on subsamples.

Value

Return an object of class Biclust

Author(s)

Sebastian Kaiser <sebastian.kaiser@stat.uni-muenchen.de>

See Also

Biclust-class, plaid.grid, bimax.grid

Examples

```
data(BicatYeast)
ensemble.plaid <- ensemble(BicatYeast, plaid.grid()[1:5], rep = 1,
                           maxNum = 2, thr = 0.5, subs = c(1,1))
ensemble.plaid
x <- binarize(BicatYeast)
ensemble.bimax <- ensemble(x, bimax.grid(), rep = 10, maxNum = 2,
                           thr = 0.5, subs = c(0.8,0.8))
ensemble.bimax
```

---

heatmapBC *Overlapping Heatmap*

Description

Other than `drawHeatmap` this function plots all or a chosen number of bicluster in one plot even if they were overlapping.

Usage

```
heatmapBC(x, bicResult, number = 0, local = FALSE, order = FALSE,
          axes = FALSE, outside = FALSE,
          zlim = c(min(x), max(x)), ...)
```

Arguments

`x`	The data matrix where the bicluster is to be drawn.
`bicResult`	BiclustResult object with a bicluster result set.
`number`	Number of bicluster to be drawn from the result set 'bicResult'. If the default 0 is chosen all bicluster of the bicResult are drawn.
`local`	If TRUE, only rows and columns of the bicluster were drawn.
`order`	If TRUE, rows and columns are ordered by there values.
`axes`	Argument passed to `image()`
`outside`	Boxes were drawn for overlapping
`zlim`	Argument passed to `image()`
`...`	Additional plot options

Details

Overlap plotting only works for two neighbor bicluster defined by the order in the number slot.

Author(s)

Sebastian Kaiser

See Also

`drawHeatmap`,`parallelCoordinates`

Examples

```
set.seed(1234)
data(BicatYeast)
resplaid <- biclust(BicatYeast, BCPlaid(), verbose = FALSE)
heatmapBC(x = BicatYeast, bicResult = resplaid)
```

---

isoverlapp *Is Bicresult overlapping?*

Description

Checks if Biclusterresult includes overlapping rows or columns

Usage

```
isoverlapp(bicResult)
```

Arguments

`bicResult`	Result of biclust function

Value

`Overlapping`	Is there overlapping
`Max.bicluster.Rows`	Maximal number of bicluster a single row is in
`Max.bicluster.Cols`	Maximal number of bicluster a single col is in

Author(s)

Sebastian Kaiser <sebastian.kaiser@stat.uni-muenchen.de>

See Also

drawHeatmap

---

jaccardind *Jaccardind*

Description

An adaption of the Jaccard Index for clustering is calculated.

Usage

```
jaccardind(bicres1,bicres2)
jaccard2(Rows, Cols)
```

Arguments

bicres1	A object of class Biclust
bicres2	A object of class Biclust
Rows	Matrix containing rows of biclusters
Cols	Matrix containing cols of biclusters

Details

The function calculates the percentage of datapoints in the same bicluster structure from all datapoints at least included in one bicluster.

Value

jaccardind calculates the Jaccard index jaccard2 returns a similarity matrix containing the Jaccard index between all biclusters (upper triangle matrix)

Author(s)

Sebastian Kaiser <sebastian.kaiser@stat.uni-muenchen.de>

Examples

```
## Not run:
data(BicatYeast)
res1<-biclust(BicatYeast, method=BCPlaid(), back.fit = 2,
         shuffle = 3, fit.model = ~m + a + b,iter.startup = 5,
         iter.layer = 30, verbose = TRUE)
res2<-biclust(BicatYeast, method=BCCC())
jaccardind(res1,res2)
## End(Not run)
```

---

parallelCoordinates

Parallel Coordinates

Description

Represents expression levels through gene and/or condition profiles in a bicluster as lines.

Usage

```
parallelCoordinates(x, bicResult, number, plotBoth = FALSE,
                    plotcol = TRUE, compare = TRUE, info = F,
                    bothlab = c("Rows", "Columns"), order = FALSE,
                    order2 = 0, ylab = "Value" , col=1, ...)
```

Arguments

`x`	The data matrix of the bicluster to be drawn
`bicResult`	BiclustResult object with a bicluster result set
`number`	Bicluster to be drawn from the result set 'bicResult'
`plotBoth`	If 'TRUE', Parallel Coordinates of rows (Genes) and columns (Conditions) were drawn one below the other.

plotcol	If 'TRUE', columns profiles are drawn, so each line represents one of the columns in the bicluster. Otherwise, row profiles are drawn. Default 'TRUE'
compare	If 'TRUE', values of the complete data matrix are considered and drawn as shaded lines. Default 'TRUE'
info	If 'TRUE', a prepared Title is drawn
bothlab	Names of the x Axis if PlotBoth
order	Rows and/or Columns are in increasing order.
order2	Which ordering.
ylab	ylab
col	col
...	Plot Parameters

Author(s)

Rodrigo Santamaria, Martin Sill and Sebastian Kaiser <sebastian.kaiser@stat.uni-muenchen.de>

See Also

drawHeatmap for alternative representation of biclusters and bubbleplot for simultaneous representation of biclusters.

Examples

```
#Random 100x50 matrix with a single, up-regulated 10x10 bicluster
s2=matrix(rnorm(5000),100,50)
s2[11:20,11:20]=rnorm(100,3,0.3)
set.seed(1)
bics <- biclust(s2,BCPlaid(), back.fit = 2, shuffle = 3,
                fit.model = ~m + a + b, iter.startup = 5,
                iter.layer = 30,  verbose = TRUE)
parallelCoordinates(x = s2,bicResult = bics,number = 1,
        plotBoth = TRUE, plotcol = TRUE, compare = TRUE,
        info = TRUE, bothlab=c("Genes Bicluster 1",
        "Conditions Bicluster 1"), order =TRUE)

parallelCoordinates(x = s2, bicResult = bics, number = 1,
        plotBoth = FALSE, plotcol = TRUE, compare = FALSE,
        info = TRUE)
```

`plaid.grid` *Parameter Grid for BCPlaid Biclustering*

Description

Generates a list containing parameter settings for the ensemble algorithm.

Usage

```
plaid.grid(method = "BCPlaid", cluster = "b",
           fit.model = y ~ m + a + b, background = TRUE,
           background.layer = NA, background.df = 1,
           row.release = c(0.5, 0.6, 0.7),
           col.release = c(0.5, 0.6, 0.7), shuffle = 3,
           back.fit = 0, max.layers = 20, iter.startup = 5,
           iter.layer = 10, verbose = FALSE)
```

Arguments

`method`	Here BCPlaid, to perform Plaid algorithm
`cluster`	'r', 'c' or 'b', to cluster rows, columns or both (default 'b')
`fit.model`	Model (formula) to fit each layer. Usually, a linear model is used, that stimates three parameters: m (constant for all elements in the bicluster), a(contant for all rows in the bicluster) and b (constant for all columns). Thus, default is: y ~ m + a + b.
`background`	If 'TRUE' the method will consider that a background layer (constant for all rows and columns) is present in the data matrix.
`background.layer`	If background='TRUE' a own background layer (Matrix with dimension of x) can be specified.
`background.df`	Degrees of Freedom of backround layer if background.layer is specified.
`shuffle`	Before a layer is added, it's statistical significance is compared against a number of layers obtained by random defined by this parameter. Default is 3, higher numbers could affect time performance.
`iter.startup`	Number of iterations to find starting values
`iter.layer`	Number of iterations to find each layer

back.fit	After a layer is added, additional iterations can be done to refine the fitting of the layer (default set to 0)
row.release	Scalar in [0,1](with interval recommended [0.5-0.7]) used as threshold to prune rows in the layers depending on row homogeneity
col.release	As above, with columns
max.layers	Maximum number of layer to include in the model
verbose	If 'TRUE' prints extra information on progress.

Value

A list containing parameter settings

Author(s)

Sebastian Kaiser <sebastian.kaiser@stat.uni-muenchen.de>

See Also

ensemble, BCPlaid

Examples

```
plaid.grid()
```

---

plotclust *Barplot of Bicluster*

Description

Draws a graph to compare the values inside the diffrent biclusters with the values outside the bicluster

Usage

```
plotclust(res, x, bicluster = TRUE, legende = FALSE, noC = 5,
          wyld = 3, Titel = "Plotclust", ...)
```

Arguments

x	The data matrix
res	BiclustResult object if bicluster=TRUE else a normal kcca object.
bicluster	If TRUE,res is treated as a BiclustResult object
legende	Draws a legend.
noC	Number of Clusters drawn
wyld	Gives the distance between plot and axis.
Titel	Gives the title of the plot.
...	Additional plot options

Author(s)

Sebastian Kaiser <sebastian.kaiser@stat.uni-muenchen.de>

See Also

bubbleplot for simultaneous representation of biclusters. parallelCoordinatesfor single representation of biclusters as lines of gene or condition profiles. drawHeatmapfor Heatmap representation of biclusters.

Examples

```
s2=matrix(rnorm(400),20,20)
s2[12:16,12:16]=rnorm(25,3,0.3)
set.seed(1)
bics <- biclust(s2, BCPlaid(), back.fit = 2, shuffle = 3,
                fit.model = ~m + a + b, iter.startup = 5,
                iter.layer = 30,  verbose = TRUE)
plotclust(bics,s2)
```

---

predictBimax *Predict from a BCrepBimax Result*

Description

Predicts cluster membership for new data rows given a BCrepBimax Result

Usage

```
predictBimax(BCrepBimax, x)
```

Arguments

`BCrepBimax`	Result of biclust function with method BCrepBimax
`x`	The data matrix which clustermembership should be predicted

Value

Returns a vector with clustermembership of data x of class.

Author(s)

Sebastian Kaiser `<sebastian.kaiser@stat.uni-muenchen.de>`

See Also

`BCrepBimax`

`SyntrenEcoli` *SynTReN E. coli*

Description

Synthetic microarray data matrix generated by Syntren for 20 experiments using 200 genes from Transcription Regulatory Network of Shen-Orr et al. (2002).

Usage

```
data(SyntrenEcoli)
```

Format

Data structure with information about the expression levels of 200 genes over 20 conditions. Conditions are named as C1... C20

Source

SynTReN software can be downloaded at
http://homes.esat.kuleuven.be/~kmarchal/SynTReN/index.html

References

Shen-Orr et al., "Network motifs in the transcriptional regulation network of Escherichia coli", Nature Genetics 2002, volume 31, pages 64-68.

Tim Van den Bulcke et al., "SynTReN: a generator of synthetic gene expression data for design and analysis of structure learning algorithms", BMC Bioinformatics, 2006, volume 7, number 43.

`writeBiclusterResults`
writeBiclusterResults

Description

Write bicluster results to a file

Usage

```
writeBiclusterResults(fileName, bicResult, bicName, geneNames,
                      arrayNames, append=FALSE, delimiter=" ")
```

Arguments

`fileName`	Path to the file were biclusters are written.
`bicResult`	Biclusters results as a Biclust class.
`bicName`	Brief description for the biclustering algorithm used.
`geneNames`	Array of strings with gene (row) names in the analyzed data matrix
`arrayNames`	Array of strings with condition (column) names in the analyzed data matrix
`append`	If true, adds the bicluster results to previous information in the text file, if it exists. Default false.
`delimiter`	delimiter string between gene and condition names. Default " ".

Author(s)

Rodrigo Santamaria <rodri@usal.es>

Examples

```
data(BicatYeast)
res <- biclust(BicatYeast, method = BCCC(), delta = 1.5,  alpha = 1,
               number=10)
writeBiclusterResults("results.txt", res, "CC with delta 1.5",
                      dimnames(BicatYeast)[1][[1]],
                      dimnames(BicatYeast)[2][[1]])
```

---

writeclust *Write a Bicluster as a Cluster Result*

Description

Draws a graph to compare the values inside the diffrent biclusters with the values outside the bicluster

Usage

```
writeclust(Biclusterresult,row=TRUE,noC=10)
```

Arguments

Biclusterresult	BiclustResult object
row	If TRUE, cluster of rows were written.
noC	Number of Clusters written

Author(s)

Sebastian Kaiser <sebastian.kaiser@stat.uni-muenchen.de>

Examples

```
s2=matrix(rnorm(400),20,20)
s2[12:16,12:16]=rnorm(25,3,0.3)
set.seed(1)
bics <- biclust(s2, BCPlaid(), back.fit = 2, shuffle = 3,
                fit.model = ~m + a + b, iter.startup = 5,
                iter.layer = 30,  verbose = TRUE)
writeclust(bics)
```

B. orddata Reference Manual

We provide an implementation of all the methods used in Chapter 4 as add on package orddata (Kaiser and Leisch, 2010) for R (R Development Core Team, 2011). It extends package bindata (Leisch et al., 2009) which contains the method of Leisch et al. (1998) for drawing correlated binary data, and will eventually replace it. In this appendix we give a small manual how to use the package performing one of the simulation study in Chapter 4.

The package can be downloaded from R-Forge and loaded into R using

```
> install.packages("orddata", repos = "http://R-Forge.R-project.org")
> library("ordata")
```

The main function of the package is rmvord(), which returns n observations with given marginal probabilities probs and correlation structure Cor using the mean mapping algorithm. probs is a list of probabilities for the variables where length of list equals number of variables and the length of the probabilities equals the number of items. The probs list for the example in section 4.4.3 looks like this

```
> probs1 <- list(c(5, 25, 55, 15)/100, c(10, 10, 10, 70)/100,
+     c(20, 15, 25, 40)/100)
```

The first correlation matrix of Table 4.1 can be specified by

```
> Cor1 <- matrix(c(1, 0.4, 0.3, 0.4, 1, 0.4, 0.3, 0.4,
+     1), 3, 3)
```

To draw n =100 observation one then has to call

```
> rmvord(n = 100, probs = probs1, Cor = Cor1)
```

If a faster production of correlated ordinal values is needed and the restrictions to the correlation matrix do not apply the function

```
> rmvord_b(n = 100, probs = probs1, Cor = Cor1)
```

does the same using the faster binary conversion method. For further details and examples, please see the package reference manual:

`minmax.ordcorr` *Minimum and Maximum of Possible Correlations*

Description

This program `minmax.ordcorr` checks the first condition of the feasibility of a correlation matrix of ordinal random numbers. A mean vector (as list) needs to be specified. It returns yes/no if also a correlation matrix was given and in either case the Min-Max Correlation Matrix, which has the minimum correlation in the lower triangular matrix and the maximum correlation in the upper triangular matrix.

Usage

```
minmax.ordcorr(probs, Cor = 0, n = 1e+06, showX = FALSE)
```

Arguments

`probs`	List of probabilities for the variables, length of probability equals number of items, length of list equals number of variables
`Cor`	Correlation matrix
`n`	Number of Observations
`showX`	If TRUE resulting correlation matrix is shown

Value

It returns yes/no if also a correlation matrix was given and in either case the Min-Max Correlation Matrix, which has the minimum correlation in the lower triangular matrix and the maximum correlation in the upper triangular matrix.

Author(s)

Dominik Traeger and Sebastian Kaiser

References

Sebastian Kaiser, Dominik Traeger and Friedrich Leisch (2011). Generating correlated ordinal data.

See Also

`check.ordcorr`

Examples

```
minmax.ordcorr(list(c(1,1,1,1)/4,c(1,1,1,1)/4), cbind(c(0.5, 0.4),
               c(0.4, 0.8)), n = 1000, showX = TRUE)
```

---

rmvord *Multivariate Ordinal Random Variates*

Description

Creates correlated multivariate ordinal random variables by thresholding a normal distribution.

Usage

```
rmvord(n = 1, probs, Cor, showCor_norm = TRUE)
```

Arguments

`n`	Number of Observations
`probs`	List of probabilities for the variables, length of probability equals number of items, length of list equals number of variables
`Cor`	Correlation matrix
`showCor_norm`	If TRUE analytical correlation matrix is printed.

Details

This function implements the mean mapping method described in Kaiser et al., 2010.

Value

Returns `n` ordinal observations with given marginal probabilities `probs` and correlation structure `Cor`.

Author(s)

Sebastian Kaiser and Dominik Traeger

References

Sebastian Kaiser, Dominik Traeger and Friedrich Leisch (2011). Generating correlated ordinal data.

See Also

`rmvord_b`,`rmvord_mc`

Examples

```
rmvord(n = 20, probs = list(c(1,1,1,1)/4,c(1,1,1,1)/4),
       Cor = cbind(c(1, 0.4), c(0.4, 1)))
```

`rmvord_b` *Multivariate Ordinal Random Variates via Binary Conversion*

Description

Creates correlated multivariate ordinal random variables by converting them into binary variables.

Usage

```
rmvord_b(n = 1, probs, Cor, showCor_b = FALSE)
```

Arguments

`n`	Number of Observations
`probs`	List of probabilities for the variables, length of probability equals number of items, length of list equals number of variables
`Cor`	Correlation matrix
`showCor_b`	If TRUE binary correlation matrix is printed.

Details

Binary conversion is used to transform the correlation matrix. For Details see Kaiser et al., 2011.

Value

Returns `n` observations with given marginal probabilities `probs` and correlation structure `Cor`.

Author(s)

Sebastian Kaiser and Dominik Traeger

References

Sebastian Kaiser, Dominik Traeger and Friedrich Leisch (2011). Generating correlated ordinal data.

See Also

`rmvord`,`rmvord_mc`

Examples

```
rmvord_b(n = 20, probs = list(c(1,1,1,1)/4,c(1,1,1,1)/4),
         Cor = cbind(c(1, 0.4), c(0.4, 1)))
```

---

rmvord_mc *Multivariate Ordinal Random Variates by Monte Carlo Simulation*

Description

Creates correlated multivariate ordinal random variables by a Monte Carlo simulation.

Usage

```
rmvord_mc(n = 1, probs, Cor)
```

Arguments

`n`	Number of Observations
`probs`	List of probabilities for the variables, length of probability equals number of items, length of list equals number of variables
`Cor`	Correlation matrix

Details

Ordinal values are produced by shifting the variables until correlation structure is reached.

Value

Returns `n` observations with given marginal probabilities `probs` and correlation structure `Cor`.

Author(s)

Dominik Traeger and Sebastian Kaiser

See Also

`rmvord_b`,`rmvord`

Examples

```
rmvord_mc(n = 20, probs = list(c(1,1,1,1)/4,c(1,1,1,1)/4),
          Cor = cbind(c(1, 0.4), c(0.4, 1)))
```

C. Mathematical nomenclature

For describing bicluster analysis we use the following notation:

A	Data matrix
a_{ij}	Value of A (row i, column j)
a_{iJ}	Mean value of row i
a_{Ij}	Mean value of column j
a_{IJ}	Overall mean
X	Objects or data rows
Y	Variables or data columns
n	Number of objects
m	Number of Variables
$I = \{i_1, ..., i_k\}, I \subset X$	Subset of objects
$J = \{j_1, ..., j_l\}, J \subset Y$	Subset of variables
$BC_z = (I_z, J_z)$	Bicluster z
$Bicres_h = \{BC_1(Bicres_h), ..., BC_g(Bicres_h)\}$	Bicluster result set
jac	Jaccard Index

Bibliography

Barkow, S., S. Bleuler, A. Prelic, P. Zimmermann, and E. Zitzler (2006). Bicat: a biclustering analysis toolbox. *Bioinformatics 22*, 1282–1283.

Ben-Dor, A., B. Chor, R. Karp, and Z. Yakhini (2003). Discovering local structure in gene expression data: The order-preserving submatrix problem. *Journal of Computational Biology 10*, 373–384.

Bergmann, S., J. Ihmels, and N. Barkai (2003). Iterative signature algorithm for the analysis of large-scale gene expression data. *Physical Review E E 67 031902*, 1–18.

Biswas, A. (2004, October). Generating correlated ordinal categorical random samples. *Statistics & Probability Letters 70*(1), 25–35.

Bühlmann, P. (2010). *Handbook of Computational Statistics: Concepts and Methods, 2nd Edition*, Chapter Bagging, boosting and ensemble methods. Springer.

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

Chia, B. and R. K. M. Karuturi (2010). Differential co-expression framework to quantify goodness of biclusters and compare biclustering algorithms. *Algorithms for Molecular Biology 5*(1), 23.

Claycamp, H. J. and W. F. Massy (1968). A theory of market segmentation. *Journal of Marketing Research 5*(4), pp. 388–394.

Couper, M. P. (2000). Web surveys: A review of issues and approaches. *Public Opinion Quarterly 64*(4), 464–94.

Csardi, G. (2009). *isa2: The Iterative Signature Algorithm.* R package version 0.2.1.

Csardi, G. (2010). *eisa: Expression data analysis via the Iterative Signature Algorithm.* R package version 1.2.0.

Demirtas, H. (2006). A method for multivariate ordinal data generation given marginal distributions and correlations. *Journal of Statistical Computation and Simulation 76(11)*, 1017–1025.

Deutskens, E., K. D. Ruyter, M. Wetzels, and P. Oosterveld (2004). Response rate and response quality of internet-based surveys: An experimental study. *Marketing Letters 15*, 21–36.

Dolnicar, S. (2002). A review of data-driven market segmentation in tourism. *Journal of Travel and Tourism Marketing 12*(1), 1 – 22.

Dolnicar, S. and B. Grün (2008). Challenging 'factor - cluster segmentation'. *Journal of Travel Research 47*(1), 63–71.

Dolnicar, S., S. Kaiser, K. Lazarevski, and F. Leisch (2011). Biclustering overcoming data dimensionality problems in market segmentation. *Journal of Travel Research*.

Dolnicar, S. and F. Leisch (2010). Evaluation of structure and reproducibility of cluster solutions using the bootstrap. *Marketing Letters 21*, 83–101.

Erhardt, V. and C. Czado (2010). *Sampling Count Variables with specified Pearson Correlation - a Comparison between a naive and a C-vine Sampling Approach*. World Scientific Publishing Company.

Eugster, M. J. A., J. Gertheiss, and S. Kaiser (2010). Having the second leg at home – advantage in the UEFA Champions League knockout phase? *Journal of Quantitative Analysis in Sports*. Accepted for publication on 2010-11-23.

Everitt, B. S., S. Landau, and M. Leese (2009). *Cluster Analysis*. London: Wiley.

Formann, A. K. (1984). *Die Latent-Class-Analyse: Einfï¿½hrung in die Theorie und Anwendung*. Weinheim: Beltz.

Fox, J. (2010). *Rcmdr: R Commander*. R package version 1.6-0.

Gange, S. J. (1995). Generating multivariate categorical variates using the iterative proportional fitting algorithm. *The American Statistician 49*(2), 134–138.

Gentleman, R. C., V. J. Carey, D. M. Bates, et al. (2004). Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology 5*, R80.

Getz, G., E. Levine, and E. Domany (2000). Coupled two-way clustering analysis of gene microarray data. *Proceedings of the National Academy of Sciences of the United States of America 97*(22), 12079–12084.

Hanczar, B. and M. Nadif (2010). Bagging for biclustering: Application to microarray data. In J. Balcï¿½zar, F. Bonchi, A. Gionis, and M. Sebag (Eds.), *Machine Learning and Knowledge Discovery in Databases*, Volume 6321 of *Lecture Notes in Computer Science*, pp. 490–505. Springer Berlin / Heidelberg.

Hartigan, J. A. (1972). Direct clustering of a data matrix. *Journal of the American Statistical Association 67*(337), 123–129.

Hastie, T., R. Tibshirani, and J. H. Friedman (2003, July). *The Elements of Statistical Learning* (Corrected ed.). Springer.

Heyer, L. J., S. Kruglyak, and S. Yooseph (1999). Exploring Expression Data: Identification and Analysis of Coexpressed Genes. *Genome Research 9*(11), 1106–1115.

Hochreiter, S., U. Bodenhofer, M. Heusel, A. Mayr, A. Mitterecker, A. Kasim, T. Khamiakova, S. V. Sanden, D. Lin, W. Talloen, L. Bijnens, H. W. H. G"ohlmann, Z. Shkedy, and D.-A. Clevert (2010). Fabia: Factor analysis for bicluster acquisition. *Bioinformatics 26*(12), 1520–1527. doi:10.1093/bioinformatics/btq227.

Jaccard, P. (1901). Distribution de la flore alpine dans le bassin des dranses et dans quelques règions voisines. *Bulletin de la Sociate Vaudoise des Sciences Naturelles 37*, 241–272.

Ji, L., K. W.-L. Mock, and K.-L. Tan (2006). Quick hierarchical biclustering on microarray gene expression data. *Proceedings of the 6th IEEE Symposium on Bioinformatics and Bioengineering 1*, 110–120.

Kaiser, S. and F. Leisch (2008). A toolbox for bicluster analysis in R. In P. Brito (Ed.), *Compstat 2008—Proceedings in Computational Statistics*, pp. 201–208. Physica Verlag, Heidelberg, Germany.

Kaiser, S. and F. Leisch (2010). *orddata: Generation of Artificial Ordinal and Binary Data*. R package version 0.1.

Kaiser, S., R. Santamaria, M. Sill, R. Theron, L. Quintales, and F. Leisch (2011). *biclust: BiCluster Algorithms*. R package version 1.0.

Kaiser, S., D. Träger, and F. Leisch (2011). Generating correlated ordinal random values. Submitted.

Khamiakova, T., S. Kaiser, and Z. Shkedy (2011). Goodness-to-fit and diagnostic tools within the differential co-expression and biclusters setting.

Kluger, Y., R. Basri, J. T. Chang, and M. Gerstein (2003). Spectral biclustering of microarray data: Coclustering genes and conditions. *Genome Research 13*, 703–716.

Kostka, D. and R. Spang (2004). Finding disease specific alterations in the co-expression of genes. *Bioinformatics 20*(Suppl 1), i194–i199.

Kotler, P. and G. Armstrong (2006). *Principles of marketing* (11th edition ed.). Upper Saddle River: Prentice Hall.

Kriegel, H.-P., P. Kröger, and A. Zimek (2009, March). Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Trans. Knowl. Discov. Data 3*, 1:1–1:58.

Lawrence, H. and P. Arabie (1985). Comparing partitions. *Journal of Classification 2*(1), 193–21.

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

Lee, M., H. Shen, J. Z. Huang, and J. S. Marron (2010, Feb). Biclustering via sparse singular value decomposition. *Biometrics*.

Leisch, F. and B. Grün (2006). Extending standard cluster algorithms to allow for group constraints. In A. Rizzi and M. Vichi (Eds.), *Compstat 2006 – Proceedings in Computational Statistics*, pp. 885–892. Physica Verlag, Heidelberg, Germany.

Leisch, F., A. Weingessel, and K. Hornik (1998). On the generation of correlated artificial binary data. Technical Report 13, SFB Adaptive Information Systems and Modelling in Economics and Management Science, Wirtschaftsuniversität Wien, Augasse 2-6, A-1090 Wien, Austria.

Leisch, F., A. Weingessel, and K. Hornik (2009). *bindata: Generation of Artificial Binary Data*. R package version 0.9-17.

Lilien, G. and A. Rangaswamy (2002). *Marketing Engineering* (2nd Edition ed.). Upper Saddle River: Pearson Education.

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

McLendon, R., A. Friedman, D. Bigner, E. Van Meir, D. Brat, G. Mastrogianakis, J. Olson, T. Mikkelsen, N. Lehmann, K. Aldape, W. Yung, O. Bogler, J. Weinstein, S. VandenBerg, M. Berger, and Prados (2008). Comprehensive genomic characterization defines human glioblastoma genes and core pathways. *Nature 455*, 1061–1068.

Mechelen, I. V., H.-H. Bock, and P. D. Boeck (2004). Two-mode clustering methods: a structured overview. *StatisticalMethods in Medical Research 13*, 363–394.

Murali, T. and S. Kasif (2003). Extracting conserved gene expression motifs from gene expression data. *Pacific Symposium on Biocomputing 8*, 77–88.

Papastefanou, G. (2001). The zuma scientific use file of the gfk consumerscan household panel 1995. In G. Papastefanou, P. Schmidt, A. Börsch-Supan, H. Lüdtke, and U. Oltersdorf (Eds.), *Social and Economic Analyses with Consumer Panel Data*, pp. 206–212. ZUMA Mannheim.

Patel, J. K. and C. B. Read (1982). *Handbook of the Normal Distribution*, Volume 40 of *Statistics: Textbooks and Monographs*. New York and Basel: Marcel Dekker, Inc.

Pfundstein, G. (2010). Ensemble methods for plaid bicluster algorithm. Bachelor Thesis, Institut für Statistik, LMU München.

Philip, K., S. Adam, L. Brown, and G. Armstrong (2001). *Principles of marketing*. Frenchs Forest: Pearson Education Australia.

Pramana, S. and S. Kaiser (2010). *BiclustGUI: R commander Plug In for bicluster analysis*. R package version 1.0.1.

Prelic, A., S. Bleuler, P. Zimmermann, A. Wil, P. Bühlmann, W. Gruissem, L. Hennig, L. Thiele, and E. Zitzler (2006). A systematic comparison and evaluation of biclustering methods for gene expression data. *Bioinformatics 22*(9), 1122–1129.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. ISBN 3-900051-07-0.

Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association 66*(336), 846–850.

Reiss, D., N. Baliga, and R. Bonneau (2006). Integrated biclustering of heterogeneous genome-wide datasets for the inference of global regulatory networks. *BMC Bioinformatics 7*(1), 280.

Reiss, D. J. (2011). *cMonkey: Intgrated Biclustering Algorithm*. R package version 4.8.0.

Santamaría, R., L. Quintales, and R. Therón (2007). Methods to bicluster validation and comparison in microarray data. In H. Yin, P. Tino, E. Corchado, W. Byrne, and X. Yao (Eds.), *Intelligent Data Engineering and Automated*

Learning - IDEAL 2007, Volume 4881 of *Lecture Notes in Computer Science*, pp. 780–789. Springer Berlin / Heidelberg.

Santamaría, R., R. Therón, and L. Quintales (2008, May). Bicoverlapper: A tool for bicluster visualization. *Bioinformatics 24*(9), 1212–1213.

Santamaría, R., R. Therón, and L. Quintales (2008). A visual analytics approach for understanding biclustering results from microarray data. *BMC Bioinformatics 9*(247).

Scharl, T. and F. Leisch (2006). The stochastic qt–clust algorithm: Evaluation of stability and variance on time–course microarray data. In A. Rizzi and M. Vichi (Eds.), *Compstat 2006—Proceedings in Computational Statistics*, pp. 1015–1022. Physica Verlag, Heidelberg, Germany.

Sheng, Q., Y. Moreau, and B. D. Moor (2003). Biclustering microarray data by Gibbs sampling. *Bioinformatics 19*.

Sill, M. and S. Kaiser (2011). *s4vd: Biclustering via sparse singular value decomposition incorporating stability selection*. R package version 1.0/r42.

Sill, M., S. Kaiser, A. Benner, and A. Kopp-Schneider (2011). Robust biclustering by sparse singular value decomposition incorporating stability selection. *Bioinformatics*.

Smith, W. R. (1956). Product differentiation and market segmentation as alternative marketing strategies. *The Journal of Marketing 21*(1), pp. 3–8.

Tanay, A., R. Sharan, and R. Shamir (2002). Discovering statistically significant biclusters in gene expression data. *Bioinformatics 18*(1), 136–144.

Tanay, A., R. Sharan, and R. Shamir (2005). Biclustering Algorithms: A Survey. In *Handbook of Computational Molecular Biology / CRC Computer and Information Science Series*.

Theußl, S. and A. Zeileis (2009, May). Collaborative Software Development Using R-Forge. *The R Journal 1*(1), 9–14.

Träger, D. (2009). Generation of correlated ordinal random numbers. Diploma Thesis, Institut für Statistik, LMU München.

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

Wilkerson, M. D. and D. N. Hayes (2010). Consensusclusterplus: a class discovery tool with confidence assessments and item tracking. *Bioinformatics 26*(12), 1572–1573.

Williams, G. J. (2009, December). Rattle: A data mining gui for r. *The R Journal 1*(2), 45–55.

Yu, K. F. and W. Yuan (2004). Regression models for unbalanced longitudinal ordinal data: computer software and a simulation study. *Computer Methods and Programs in Biomedicine 75*(3), 195 – 200.

Zins, A. H. (2008). *Change Management in Tourism: From Old to New*, Chapter Market Segmentation in tourism: A critical review of 20 years, pp. 289 – 301. Erich Schmidt, Verlag.