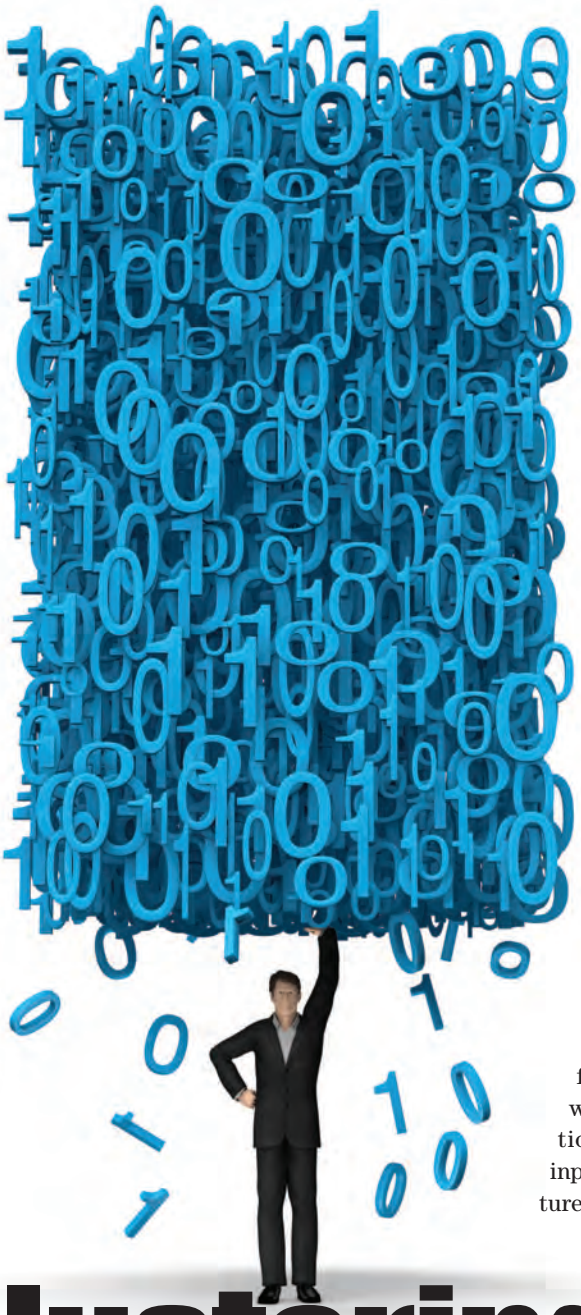
Coclustering of Multidimensional Big Data

A Useful Tool for Genomic, Financial, and Other Data Analysis

by Hong Yan

The analysis of a multidimensional data array is necessary in many applications. Although a data set can be very large, it is possible that meaningful and coherent patterns embedded in the data array are much smaller in size. For example, in genomic data, we may want to find a subset of genes that coexpress under a subset of conditions. In this article, I will explain coclustering algorithms for solving the coherent pattern-detection problem. In these methods, a coherent pattern corresponds to a low-rank matrix or tensor and can be represented as an intersection of hyperplanes in a high-dimensional space. We can then extract coherent patterns from the large data array by detecting hyperplanes. Examples will be provided to demonstrate the effectiveness of the coclustering algorithms for solving unsupervised pattern classification problems.

Digital Object Identifier 10.1109/MSMC.2017.2664218
Date of publication: 18 April 2017



©ISTOCKPHOTO/GRANDEDUC

Coclustering of 2-D and 3-D Data

Coclustering is often called *biclustering* for two-dimensional (2-D) data and *triclustering* for three-dimensional (3-D) data [1]–[3]. Let us take the data in Figure 1 as an example. Our input is a large matrix in Figure 1(a) that appears to be just random data. In fact, a subset of rows and a subset of columns contain a much smaller coherent pattern. If we rearrange the elements in the matrix so that the rows and columns in the pattern are placed in a contiguous region, then we will see the coherent pattern in Figure 1(b). Our task of coclustering here is to find the locations or indexes of these rows and columns [Figure 1(c)].

This concept of coclustering developed for 2-D data can be extended to 3-D and higher-dimensional data, which are represented as higher-order tensors. It is useful to explain the terminology we use here. In pattern recognition and machine learning, an input sample represented by a feature vector is often considered as a

point in a multidimensional space. That is, the variable for each element in the vector represents an axis in the multidimensional space. The dimension of the vector is the number of elements in the vector. However, in image processing and computer programming, we use the terms like 2-D and 3-D *images*, and 2-D and 3-D *data arrays*. Here, *dimension* means the number of directions or indices in an image or data array. In tensor algebra, each direction is called a *mode*, and the term *multiway data array* is often used. For simplicity, we will use the word *multidimensional* for both vector spaces and data arrays when the context is clear. That is, for real numbers, a multidimensional space corresponds to the set $\mathbf{R}^M$, where M is the number of elements or dimensionality of the vectors in the space. An N-dimensional data array (N-way data array, or Nth order, or N-mode tensor) belongs to the set $\mathbf{R}^{M_1 \times M_2 \times \cdots \times M_N}$, where M_1, M_1, …, M_N are the numbers of elements in corresponding directions.

In the previous discussion, we used the term *coherent pattern*, which means that elements in the rows or columns of the pattern are correlated and display certain common properties. We will explain later that a coherent pattern can be represented as a low-rank matrix or tensor and that our tasks of coclustering are to extract one or more coherent patterns embedded in the matrix or tensor.

Although a data set can be very large, it is possible that meaningful and coherent patterns embedded in the data array are much smaller in size.

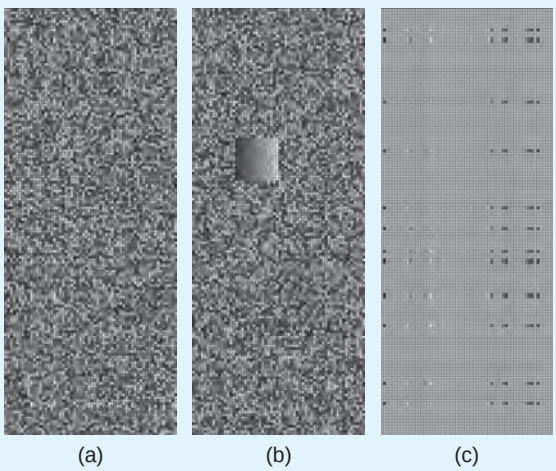


Figure 1. An example of a cocluster. (a) A large matrix that appears to contain just random data. (b) A cocluster exists in the matrix. The elements of the cocluster can be put together to show a coherent pattern by rearranging rows and columns in (a). (c) The locations of the elements in the cocluster in the original matrix in (a), with all elements outside the cocluster shown as a uniform gray background.

Differences Between Clustering and Coclustering

Clustering is a commonly used method in unsupervised learning. Given the input data as a matrix, the task of clustering is to partition the matrix into clusters along either the row or the column direction, but not both at the same time [Figure 2(a)–(b)]. The rows and columns correspond to samples and features, respectively, in pattern recognition applications. Similar to clustering, coclustering deals with unlabeled data and performs unsupervised pattern classification. However, there is a fundamental difference between clustering and coclustering. In coclustering, the data are partitioned in both row and column directions. Therefore, for 2-D data, a cluster consists of a subset of rows (or columns) with all columns (rows) present in each row (column), while a cocluster consists of a subset of rows and a subset of columns [Figures 2(c)].

There are several other differences between clustering and coclustering [3], [16]. In hard clustering, the clusters do not have any overlap [Figure 2(a)–(b)]. That is, a sample is assigned to one and only one cluster unless it is considered as an outlier and is excluded in classification. In soft clustering, a sample is assigned to each cluster with a membership value in fuzzy logic modeling. In coclustering, several coclusters can partially overlap. For example, in Figure 2(c), cocluster 3 overlaps with all the other three coclusters. In addition, an element in the data matrix may belong to none, one, or more coclusters. As shown in Figure 1(b), a large matrix may contain much smaller coclusters, and most elements in the matrix are irrelevant. We need to identify the row and column indices of relevant elements to detect the coclusters.

In hard clustering [Figure 2(a)–(b)], we can always exchange the rows or columns so that each cluster occupies a contiguous region. However, in coclustering, we may not be able to display all coclusters as contiguous regions at the same time. This is illustrated in Figure 2(c). If we exchange rows so that three regions of cocluster 3 become contiguous, then cocluster 2 or cocluster 4 would be split into more than one region. Therefore, it is more difficult to visualize all coclusters than clusters.

The concept of coclustering was introduced in the 1970s, but it has gained much attention only recently due to its useful applications to genomic data analysis [1]–[26]. In a genome, genes usually work together to perform a biological function. In gene expression data, the samples represent genes and features conditions. A condition can be a time point, an organ, a species, or a different experiment setup. A cocluster represents a subset of genes that coexpress (or are coregulated) under a subset of conditions. Coclustering analysis is useful for understanding diseases caused by aberrant coexpressed genes.

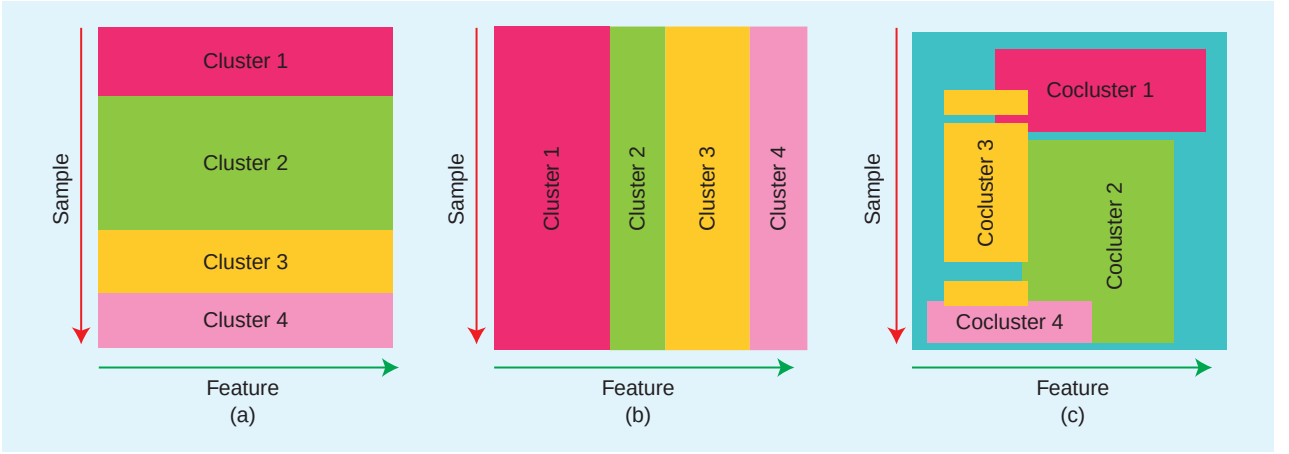


Figure 2. An illustration of the differences between clustering and coclustering. (a) Clustering involving data classification along the sample direction only. (b) Clustering with data classification along the feature direction only. (c) Coclustering involving data classification along both sample and feature directions.

Types of Coclusters

Let us consider 2-D data first. Assume that our input is a matrix $\mathbf{A} = (a_{ij}) \in \mathbf{R}^{M_1 \times M_2}$ and that a 2-D cocluster $\mathbf{A}_{\mathbf{IJ}}$ contains elements of $\mathbf{A}$ in $N_{\mathbf{I}}$ rows $\mathbf{I} = \{i_1, i_2, \cdots, i_{N_1}\}$ and $N_{\mathbf{J}}$ columns $\mathbf{J} = \{j_1, j_2, \cdots, j_{N_2}\}$. We will define several types of 2-D coclusters [3], [16], [25]. These coclusters have been studied extensively in gene expression analysis, and their biological meanings will be explained.

1) *Constant cocluster:* $\mathbf{A}_{\mathbf{IJ}} = \{a_{ij} = \mu \mid i \in \mathbf{I}, j \in \mathbf{J}\}$, where μ is a constant. In a constant cocluster, all elements take the same value. In gene expression data analysis, a constant cocluster means all genes in $\mathbf{I}$ have the same expression level under all experiment conditions in $\mathbf{J}$.
2) *Constant-row cocluster:* $\mathbf{A}_{\mathbf{IJ}} = \{\mathbf{a}_{ij}^T = \mu_i \mathbf{1}_{\mathbf{J}} \mid i \in \mathbf{I}\}$, where $\mathbf{1}_{\mathbf{J}}$ is a vector of $N_{\mathbf{J}}$ ones. Similarly, we can define a constant-column cocluster as $\mathbf{A}_{\mathbf{IJ}} = \{\mathbf{a}_{ij} = \mu_j \mathbf{1}_{\mathbf{I}} \mid j \in \mathbf{J}\}$, where $\mathbf{1}_{\mathbf{I}}$ is a vector of $N_{\mathbf{I}}$ ones. In a constant-row cocluster, elements in each row have the same value, while in a constant-column cocluster, elements in each column have the same value. In gene expression analysis, a constant-row cocluster means each gene in $\mathbf{I}$ has the same expression level under all conditions in $\mathbf{J}$, but different genes have different expression levels. A constant-column cocluster means all genes in $\mathbf{I}$ have the same expression level for each condition in $\mathbf{J}$, but the expression level is different for a different condition.
3) *Additive cocluster:* $\mathbf{A}_{\mathbf{IJ}} = \{\mathbf{a}_{ij}^T = \mu_i \mathbf{1}_{\mathbf{J}} + \mathbf{a}_{qj}^T \mid i, q \in \mathbf{I}\}$. In an additive cocluster, a column can be obtained by adding a constant to another column. In gene expression analysis, an additive cocluster means that the expression level of each gene in $\mathbf{I}$ under a condition in $\mathbf{J}$ is always increased or decreased by a constant under another condition. Here, we have defined the additivity in terms of columns. It is also possible to define the additivity in terms of rows, but such a pattern has not been used in gene expression analysis, to our knowledge.
4) *Multiplicative cocluster:* $\mathbf{A}_{\mathbf{IJ}} = \{\mathbf{a}_{ij}^T = \mu_i \mathbf{a}_{qj}^T \mid i, q \in \mathbf{I}\}$. In a multiplicative cocluster, a column can be obtained by multiplying another column by a constant. In gene expression analysis, a multiplicative cocluster means that the expression level of each gene in $\mathbf{I}$ under a condition in $\mathbf{J}$ is always scaled up or down by a constant under another condition. Again, here we have defined the scaling in terms of columns. It is also possible to define the scaling in terms of rows, but such a pattern has not been used in gene expression analysis, to our knowledge.

Several numerical examples of these coclusters are shown in Figure 3.

8.2	8.2	8.2	8.2
8.2	8.2	8.2	8.2
8.2	8.2	8.2	8.2
8.2	8.2	8.2	8.2

(a)

1.5	1.5	1.5	1.5
5.2	5.2	5.2	5.2
2.8	2.8	2.8	2.8
9.3	9.3	9.3	9.3

(b)

1.5	5.2	2.8	9.3
1.5	5.2	2.8	9.3
1.5	5.2	2.8	9.3
1.5	5.2	2.8	9.3

(c)

6.8	5.7	7.7	4.8
5.6	4.5	6.5	3.6
3.3	2.2	4.2	1.3
8.5	7.4	9.4	6.5

(d)

2.2	3.3	1.1	4.4
3.4	5.1	1.7	6.8
4.6	6.9	2.3	9.6
5.6	8.4	2.8	11.2

(e)

Figure 3. Examples of coclusters: (a) a constant cocluster, (b) a constant-row cocluster, (c) a constant-column cocluster, (d) an additive cocluster, and (e) a multiplicative cocluster.

The linear relations (coherence among rows or columns) in 2-D coclusters can be extended to higher-dimensional data. We provide definitions for 3-D data only [25], and these definitions can be further extended to higher dimensions in a similar way. Assume that our input is a third-order tensor, $\mathcal{A} = (a_{ijk}) \in \mathbf{R}^{M_1 \times M_2 \times M_3}$, and that a 3-D cocluster, $\mathbf{A_{IJK}}$, contains elements in locations $\mathbf{I} \times \mathbf{J} \times \mathbf{K} = \{i_1, i_2, \cdots, i_{N_1}\} \times \{j_1, j_2, \cdots, j_{N_2}\} \times \{k_1, k_2, \cdots, k_{N_K}\}$. We define the following 3-D coclusters:

1) *Constant cocluster*: $\mathbf{A_{IJK}} = \{a_{ijk} = \mu \mid i \in \mathbf{I}, j \in \mathbf{J}, k \in \mathbf{K}\}$
2) *Constant-column-fiber cocluster*: $\mathbf{A_{IJK}} = \{a_{Ijk} = \mu_{jk}\mathbf{1}_I \mid j \in \mathbf{J}, k \in \mathbf{K}\}$
3) *Additive-fiber cocluster*: $\mathbf{A_{IJK}} = \{a_{Ijk} = \mu_{jk}\mathbf{1}_I + \mathbf{a}_{I(1)} \mid j \in \mathbf{J}, k \in \mathbf{K}\}$, where $\mathbf{a}_{I(1)}$ is the first fiber of the co-cluster
4) *Multiplicative-fiber cocluster*: $\mathbf{A_{IJK}} = \{a_{Ijk} = \mu_{jk}\mathbf{a}_{I(1)} \mid j \in \mathbf{J}, k \in \mathbf{K}\}$.

In practical applications, a cocluster pattern often contains noise, and the linear relations defined above only hold approximately. Also, there can be more than one cocluster in the data, and we need to determine the index sets **I** and **J** for 2-D data or **I**, **J**, and **K** for 3-D data for each cocluster.

Coclustering Algorithms

Cheng and Church [1] were the first who introduced coclustering to gene expression data analysis. In their method, the following cost function is minimized to find a 2-D cocluster defined on $\mathbf{I} \times \mathbf{J}$:

$$C(\mathbf{I},\mathbf{J}) = \frac{1}{|\mathbf{I}||\mathbf{J}|}\sum_{i\in\mathbf{I}}\sum_{j\in\mathbf{J}}(a_{ij} - \bar{a}_{i\mathbf{J}} - \bar{a}_{\mathbf{I}j} + \bar{a}_{ij})^2, \tag{1}$$

where $\bar{a}_{i\mathbf{J}} = (1/|\mathbf{J}|)\sum_{j\in\mathbf{J}} a_{ij}$, $\bar{a}_{\mathbf{I}j} = (1/|\mathbf{I}|)\sum_{i\in\mathbf{I}} a_{ij}$, and $\bar{a}_{\mathbf{IJ}} = (1/|\mathbf{I}||\mathbf{J}|)\sum_{i\in\mathbf{I}}\sum_{j\in\mathbf{J}} a_{ij}$ represent row average, column average, and pattern average, respectively, of the cocluster. The method can be used to detect constant coclusters. It is interesting to note that in (1), if we remove the row and column averages and change the sign of the pattern average, then the cost function looks similar to that used for the *k*-means clustering algorithm.

A number of other coclustering methods have also been developed recently [2]–[26]. Most of these methods can be applied to 2-D data and deal with a specific type of coclusters only. Several review papers provide detailed summaries and comparisons of these methods [3], [16], [20].

Our research group has proposed geometric models for coclusters [6], [7], [9], [15], [19], [25]. According to the definitions of 2-D coclusters discussed previously, columns of a cocluster can be related by linear equations. Geometrically, these equations correspond to lines, planes, or hyperplanes in 2-D, 3-D, or higher-dimensional spaces, respectively. If we consider all columns at the same time, then we need to find all these hyperplanes and determine their intersections. The hyperplanes can be detected based on the Hough transform (HT), which is widely used in image processing and pattern recognition [27], [28].

In coclustering, the data is partitioned in both row and column directions.

If the input data contain a large number of columns, then the HT-based method that is applied to all columns at the same time becomes ineffective because the accumulator array for the quantized hyperplane parameters is too large. The problem can be overcome if we only consider two columns at a time. Subcoclusters detected from column-pair spaces can then be merged to form larger ones. For 3-D and higher-dimensional data, there will be a large number of column pairs. This problem can be solved if we transform the input data first using singular-value decomposition (SVD) [25], which will be discussed in the next section.

Cocluster Detection in Singular-Vector Spaces

In this section, we describe a coclustering method based on hyperplane detection in singular-vector spaces (HDSVS), which we have recently developed [25]. This method can be used to attend to all of the types of coclusters previously discussed and low-rank matrices or tensors in general that are embedded in high-dimensional data arrays.

Let us consider 2-D coclusters first. According to their definition given previously, an additive cocluster as a matrix has rank two, and constant, constant-row, constant-column, and multiplicative coclusters have rank one. Applying SVD to $\mathbf{A_{IJ}}$, we obtain

$$\begin{aligned}\mathbf{A_{IJ}} = \mathbf{U}\Sigma\mathbf{V}^T &= [\mathbf{u}_1 \ \mathbf{u}_2]\begin{bmatrix}\sigma_1 & 0\\ 0 & \sigma_2\end{bmatrix}\begin{bmatrix}(\mathbf{v}_1)^T\\ (\mathbf{v}_2)^T\end{bmatrix}\\ &= [\mathbf{u}_1 \ \mathbf{u}_2]\begin{bmatrix}\sigma_1 & 0\\ 0 & \sigma_2\end{bmatrix}\begin{bmatrix}v_{11} & v_{12} & \cdots & v_{1M_2}\\ v_{21} & v_{22} & \cdots & v_{2M_2}\end{bmatrix}\\ &= [\sigma_1 v_{11}\mathbf{u}_1 + \sigma_2 v_{21}\mathbf{u}_2 \ \ \sigma_1 v_{12}\mathbf{u}_1\\ &\quad + \sigma_2 v_{22}\mathbf{u}_2 \cdots \ \sigma_1 v_{1M_2}\mathbf{u}_1 + \sigma_2 v_{2M_2}\mathbf{u}_2],\end{aligned} \tag{2}$$

where **U**, $\mathbf{\Sigma}$, and **V** are matrices containing left singular vectors, singular values, and right singular vectors, respectively. Here, we only keep two leading singular values because the rank of $\mathbf{A_{IJ}}$ is at most two.

Equation (2) shows that each column $\mathbf{a}_{\mathbf{I}j}$ of $\mathbf{A_{IJ}}$ can be expressed as a linear combination of singular vectors $\mathbf{u}_1$ and $\mathbf{u}_2$:

$$\sigma_1 v_{1j}\mathbf{u}_1 + \sigma_2 v_{2j}\mathbf{u}_2 = \mathbf{a}_{\mathbf{I}j}. \tag{3}$$

This set of linear equations represents a line in the $\mathbf{u}_1$ and $\mathbf{u}_2$ space, while elements of $\mathbf{u}_1$ and $\mathbf{u}_2$ are points on the line. Similarly, we can show that each row of $\mathbf{A_{IJ}}$ can be expressed as a linear combination of singular vectors, $\mathbf{v}_1$ and $\mathbf{v}_2$, and elements of $\mathbf{v}_1$ and $\mathbf{v}_2$ form a line.

Note that (3) holds as long as $\mathbf{A}_{IJ}$ has rank two or less. For example, if a column of $\mathbf{A}_{IJ}$ is obtained by scaling another column and adding a constant, as a combination of additive and multiplicative cocluster patterns, $\mathbf{A}_{IJ}$ has rank two, and (3) is also valid. In general, HDSVS can be used to extract low-rank patterns embedded in a large matrix or tensor.

If the data have noise, we need to filter out small singular values. If there is more than one cocluster in the data, then the rank of the matrix can be larger than two, and we need to retain more singular vectors. In this case, each cocluster will correspond to a hyperplane in general in the singular-vector spaces. When coclusters are embedded in a large matrix, rows and columns in the coclusters will be found from the detected hyperplanes, while irrelevant elements will be distributed randomly off the hyperplanes. Different scenarios have been discussed in [25].

The concept of coclustering was introduced in the 1970s, but it has only gained much attention recently due to its useful applications to genomic data analysis.

Let us assume that we have found row indices sets $\mathbf{I}_1$ and $\mathbf{I}_2$ from the hyperplanes detected from left singular vectors and column indices sets $\mathbf{J}_1$ and $\mathbf{J}_2$ from the hyperplanes detected from right singular vectors. There are four combinations of these index sets: $\mathbf{I}_1 \times \mathbf{J}_1$, $\mathbf{I}_1 \times \mathbf{J}_2$,$\mathbf{I}_2 \times \mathbf{J}_1$, and $\mathbf{I}_2 \times \mathbf{J}_2$. It is possible that some of these pairs do not correspond to coclusters. We can use the following scoring function to select coclusters [25]:

$$S(\mathbf{I},\mathbf{J}) = \min_{i\in\mathbf{I}, j\in\mathbf{J}}(S_{Ij}, S_{iJ}) = \min_{i\in\mathbf{I}, j\in\mathbf{J}}\left[1-\frac{1}{|\mathbf{J}|-1}\sum_{q\neq j, q\in\mathbf{J}}\rho(\mathbf{a}_{Ij},\mathbf{a}_{Iq}), 1-\frac{1}{|\mathbf{I}|-1}\sum_{q\neq i, q\in\mathbf{I}}\rho(\mathbf{a}_{iJ},\mathbf{a}_{qJ})\right], \quad (4)$$

where ρ represents Pearson's correlation coefficient between two vectors. We consider that a cocluster is identified if $S(\mathbf{I}, \mathbf{J})$ is less or equal to a prespecified value.

The HDSVS method can be extended to higher-order tensors [25]. The higher-order SVD (HOSVD) [29]–[32] of an N-mode tensor $\mathcal{A} \in \mathbf{R}^{M_1\times M_2\times\cdots\times M_N}$ is defined as

$$\mathcal{A} = \mathcal{B}\times_1\mathbf{U}^{(1)}\times_2\mathbf{U}^{(2)}\times_3\cdots\times_N\mathbf{U}^{(N)}, \quad (5)$$

where $\mathcal{B}$ is the core tensor, $\mathbf{U}^{(n)}$ (n = 1, 2, …, N) contains singular vectors, and $\times_n$ stands for tensor and matrix multiplication along mode n. Then, $\mathbf{U}^{(n)}$ can be computed from the mode-n unfolded matrix $\mathbf{A}^{(n)}$ of the tensor. In general, $\mathbf{A}^{(n)}$ has the size of $M_n \times M_1\cdots M_{n-1}M_{n+1}\cdots M_N$ and is obtained by keeping mode n as one dimension and flattening all other modes to another dimension. The SVD of $\mathbf{A}^{(n)}$ has the following form:

$$\mathbf{A}^{(n)} = \mathbf{U}^{(n)}\Sigma^{(n)}(\mathbf{V}^{(n)})^T, \quad (6)$$

where $\mathbf{U}^{(n)}$ and $\mathbf{V}^{(n)}$ are matrices for the left and right singular vectors, respectively, and $\mathbf{\Sigma}^{(n)}$ is a diagonal matrix containing singular values. For n = 1, 2, …, N, we only need to compute $\mathbf{U}^{(n)}$ and not $\mathbf{V}^{(n)}$ to obtain (5). The decomposition of a three-mode tensor is illustrated in Figure 4.

By detecting hyperplanes in the singular-vector space with $\mathbf{U}^{(n)}$, we can then find indices sets along mode n for cocluster identification. An example is shown in Figure 5. The index sets along three modes are **I**, **J**, and **K** for 3-D data, and they should be understood as $\mathbf{I}^{(1)}$, $\mathbf{I}^{(2)}$, and $\mathbf{I}^{(3)}$ in a general notation. Assume that an index set along mode n is $\mathbf{I}^{(n)}$; then, the scoring function in (4) can be extended to higher dimensional coclusters:

$$S(\mathbf{I}^{(1)},\mathbf{I}^{(2)},\cdots,\mathbf{I}^{(N)}) = \min_{i_1\in\mathbf{I}^{(1)}, i_2\in\mathbf{I}^{(2)},\cdots,i_N\in\mathbf{I}^{(N)}}(S_{i_1\mathbf{I}^{(2)}\cdots\mathbf{I}^{(N)}}, S_{\mathbf{I}^{(1)}i_2\mathbf{I}^{(3)}\cdots\mathbf{I}^{(N)}},\cdots, S_{\mathbf{I}^{(1)}\mathbf{I}^{(2)}\cdots i_N}). \quad (7)$$

Examples

As an example, we show an eight-by-five matrix in Figure 6(a). The highlighted elements in the matrix form a general linear cocluster, which is a combination of additive and multiplicative

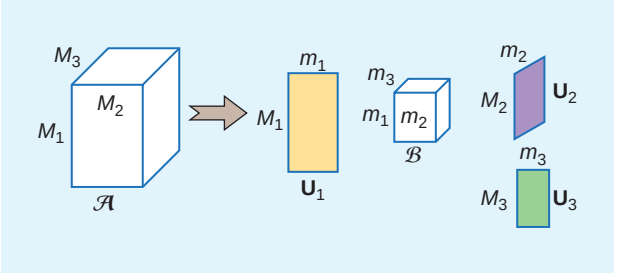


Figure 4. The decomposition of a three-mode tensor using HOSVD. $\mathbf{U}_1$ can be found by unfolding $\mathcal{A}$ to an $M_1 \times M_2M_3$ matrix and computing the left singular vectors of this matrix. $\mathbf{U}_2$ and $\mathbf{U}_3$ can be computed in similar ways.

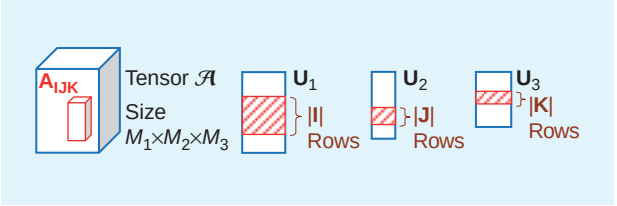


Figure 5. The detection of coclusters in singular vector spaces. In this diagram, $\mathbf{A}_{IJK}$ is a cocluster embedded in a large tensor $\mathcal{A}$. The index sets **I**, **J**, and **K** can be determined by searching for hyperplanes based on $\mathbf{U}_1$, $\mathbf{U}_2$, and $\mathbf{U}_3$, respectively.

patterns. Column 4 is obtained by scaling column 1 by 0.9 and adding a constant 10, and column 5 is obtained by scaling column 1 by 1.2 and adding a constant 23. These additive and multiplicative relations are approximate because the cocluster is corrupted by noise. Moving columns 1, 4, and 5 together [Figure 6(b)], we can see a ramp pattern with increasing value from the upper left to the lower right. Values of elements in the cocluster are distributed in the range 19–89, while those in the background (irrelevant elements) are in the range 19–99 plus an outlier with value 5. So there is no easy way to determine which elements belong to the cocluster from just the values.

In HDSVS, the removal of small singular values can reduce the number of variables for hyperplane detection.

The linear relations among the columns in the cocluster in Figure 6(a) correspond to hyperplanes in a five-dimensional space. In practical applications, the number of columns would be much larger, and it would be difficult to detect the hyperplanes and find intersections of them with all column variables at the same time. Alternatively, we can consider each column pair at a time. Column pair 1–2 contains only scattered points as shown in blue in Figure 6(c). One may argue that points in rows 2, 3, and 7, i.e., (76, 78), (26, 43), and (92, 90), are approximately collinear. We can specify the minimum number of rows required for a cocluster, four in this case, to filter out noisy components. Further filtering can be done by specifying the minimum number of columns required. For column pair 1–4, we detect a long line with five points in rows, {1, 3, 4, 6, 8}, which corresponds to row locations of the cocluster. The line detection can be done using the HT in general. We will obtain the same results for column pairs 1–5 and 4–5.

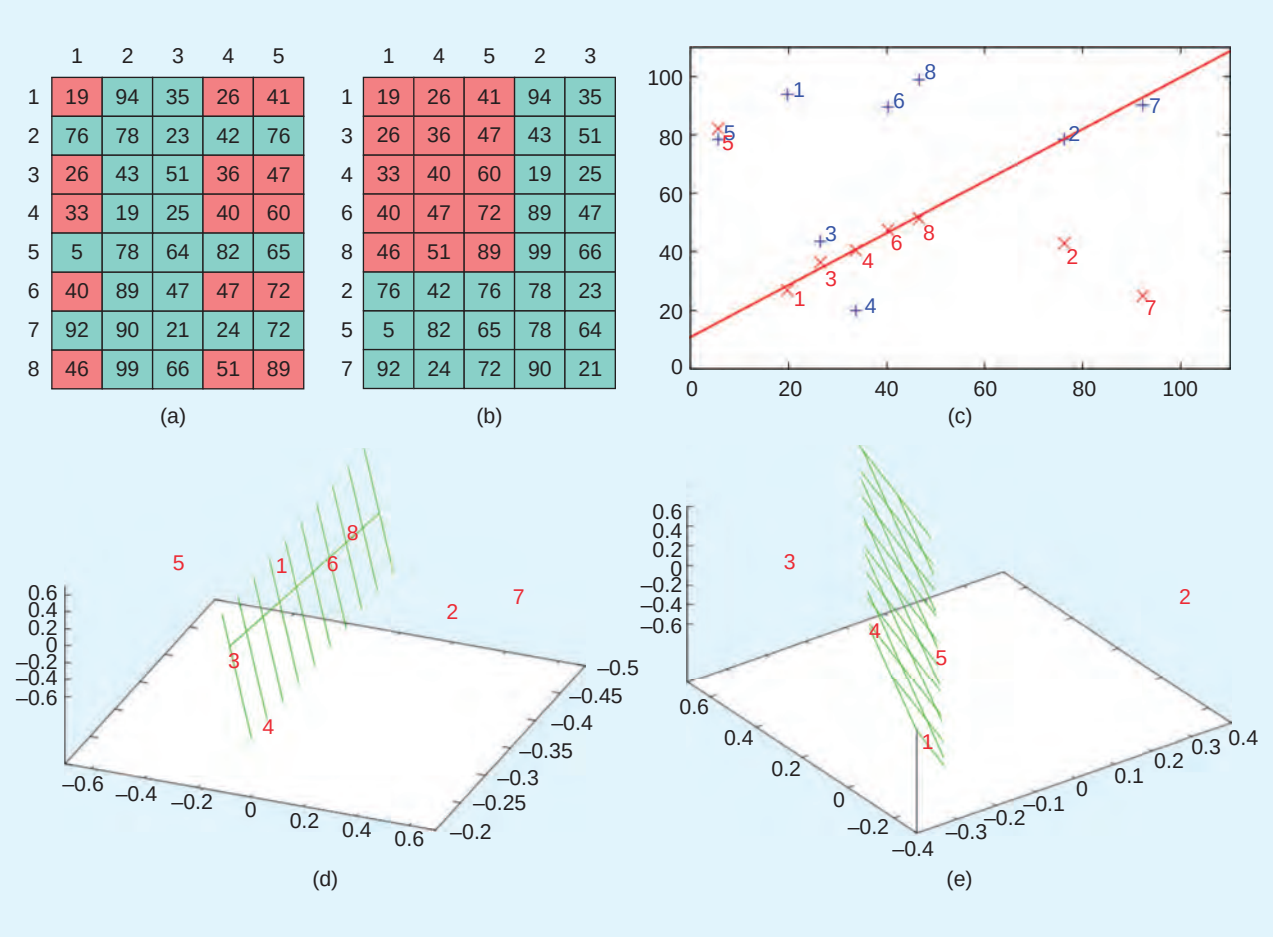


Figure 6. An illustration of 2-D cocluster detection. (a) A noisy cocluster with the combination of additive and multiplicative patterns embedded in a matrix with large variations in the background. (b) The column pairs 1–2 (blue) and 1–4 (red). (c) A long line close to five points is detected in the space of column pair 1–4. (d) The points shown in the u_1-u_2-u_3 space. A hyperplane close to six points is detected. (e) The points shown in the v_1-v_2-v_3 space. A hyperplane close to three points is detected.

Therefore, we find that a cocluster exists in $\{1, 3, 4, 6, 8\} \times \{1, 4, 5\}$.

In this example, the column-pair-based method works well. For a matrix with M_2 columns, there are $M_2(M_2 - 1)$ column pairs, and to combine the results from all these column pairs can be complicated. One may think many column pairs are redundant. In the example here, we can obtain the column indices {1, 4, 5} from column pairs 1–4 and 1–5 without considering 4–5. However, in more complex cases, due to noisy and multiple and overlapping coclusters, the column pairs may not produce the same results; therefore, it is difficult to determine which pairs are redundant.

Coclustering provides classifications simultaneously in all directions of a multidimensional data array.

We can consider all columns at the same time using HDSVS. First, we compute the SVD of the input matrix. The singular values of the example in Figure 6(a) are 358.56, 89.39, 54.81, 23.69, and 12.84. It is reasonable to retain the first three large singular values and remove the two smallest ones to filter out noise. Although the cocluster has a rank of two, we usually obtain better results by retaining slightly more singular values. In Figure 6(d), we consider singular vectors $\mathbf{u}_1$, $\mathbf{u}_2$, and $\mathbf{u}_3$ corresponding to the three retained singular values. In the u_1-u_2-u_3 space, we detect a hyperplane close to (u_{11}, u_{12}, u_{13}), (u_{31}, u_{32}, u_{33}), (u_{41}, u_{42}, u_{43}), (u_{61}, u_{62}, u_{63}), and (u_{81}, u_{82}, u_{83}). Similarly, in the v_1-v_2-v_3 space, we detect a hyperplane close to (v_{11}, v_{12}, v_{13}), (v_{41}, v_{42}, v_{43}), and (v_{51}, v_{52}, v_{53}) [Figure 6(e)]. Therefore, a cocluster exists in $\{1, 3, 4, 6, 8\} \times \{1, 4, 5\}$.

In HDSVS, the removal of small singular values can reduce the number of variables for hyperplane detection. If the matrix is large and the coclusters are small in size, the reduction will be significant. Hyperplane detection is a key step in HDSVS. This can be done using the HT or the linear grouping algorithm [25].

Although the matrix in this example only has the size of eight by five for the purpose of illustration, the computational procedure for a larger data size is the same. In our work on gene expression data analysis, many experiments were carried out with large data sets, and more examples can be found in [6], [7], [9], [14], [15], [19], and [25]. To improve computational speed, graphics processing units and specially designed hardware, such as field programmable gate arrays, can be employed [22], [23].

Discussions and Conclusions

Coclustering is useful for genomic data analysis. In addition, it has many other applications [3], [16], some of which are

- analysis of consumer spending patterns to discover a subset of consumers who like to purchase products from a subset of companies
- analysis of financial data to find a subset of sectors that have similar economic growth patterns in a subset of regions
- analysis of chemical components in food to determine a subset of components (nutrients or toxicants) in a subset of food types
- analysis of web documents to search for a subset of documents that contain a subset of keywords in a subset of locations
- analysis of training data in machine learning to extract a subset of features that are relevant to a subset of features.

In [8], we analyzed foreign exchange data over 120 months and found currencies with correlations in some but not all time periods. Some of the coclusters reveal that several currencies had similar rapid movement patterns in the early stage of the Asian financial crisis in late 1990s. In [26], we used coclustering to select Gabor wavelet features of images for facial expression classification. There are thousands and even tens of thousands of features to compute for a full set of Gabor wavelets with different scales and orientations in different locations of a face image. With the type of facial expression in one direction and the Gabor wavelet feature in another direction, we performed cocluster analysis of the feature data. From the coclusters, we can retain relevant features for a facial expression and discard all other features. With only 20% of the features selected this way, we have obtained an even higher classification rate than that from the entire set of features. The reason for the improved performance is that, through coclustering, we can remove irrelevant and noisy features and enhance the feature quality. It is interesting that a reasonable accuracy can still be achieved even when we retain only 0.9% of the original features.

This example of facial expression feature selection is especially encouraging. In traditional machine-learning and pattern-recognition methods, all features are used to build a classifier. In practical applications, if our input data contain a large number of features, a subset of features may be useful for only a subset of classes, and another subset of features may be useful for just another subset of classes. In these cases, feature selection is a combinatorial problem. Coclustering provides a systematic and robust method to deal with big data sets and select relevant and important features. This procedure can be used to filter out noise, improve classification performance, and reduce the computing time significantly.

In summary, coclustering provides classifications simultaneously in all directions of a multidimensional data array. It may be used to detect coherent patterns that are embedded in a large matrix or tensor and contain a subset of elements in each direction. It can be performed effectively in singular-vector spaces based

on hyperplane detection. We expect that coclustering will gain more and more applications in machine learning, image processing, bioinformatics, and big data analytics.

Acknowledgments

This work is supported by the Hong Kong Research Grants Council (project CityU 11214814) and City University of Hong Kong (project 9610308).

About the Author

Hong Yan (h.yan@cityu.edu.hk) earned his Ph.D. degree from Yale University, New Haven, Connecticut. He was a professor of imaging science at the University of Sydney, Australia, and he is currently a professor of computer engineering at City University of Hong Kong. His research interests include image processing, pattern recognition, and bioinformatics. He was elected an International Association for Pattern Recognition fellow for contributions to document image analysis and an IEEE Fellow for contributions to image recognition techniques and applications. He received the 2016 Norbert Wiener Award from the IEEE Systems, Man, and Cybernetics Society for contributions to image and biomolecular pattern recognition techniques.

References

[1] Y. Cheng and G. M. Church, "Biclustering of expression data," in *Proc. 8th Int. Conf. Intelligent Syst. Molecular Biology*, 2000, pp. 93–103.

[2] Y. Kluger, R. Basri, J. T. Chang, and M. Gerstein, "Spectral biclustering of microarray data: Coclustering genes and conditions," *Genome Res.*, vol. 13, no. 4, pp. 703–716, 2003.

[3] S. C. Madeira and A. L. Oliveira, "Biclustering algorithms for biological data analysis: A survey," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 1, no. 1, pp. 24–45, 2004.

[4] B. Long, Z. Zhang, and P. S. Yu, "Coclustering by block value decomposition," in *Proc. Assoc. Computing Machinery Special Interest Group Conf. Knowledge Discovery and Data Mining (ACM SIGKDD)* , 2005, pp. 635–640.

[5] P. Carmona-Saez, R. D. Pascual-Marqui, F. Tirado, J. M. Carazo, and A. Pascual-Montano, "Biclustering of gene expression data by non-smooth non-negative matrix factorization," *BMC Bioinformatics*, vol. 7, no. 78, pp. 1–18, 2006.

[6] X. Gan, A. W. C. Liew, and H. Yan, "Discovering biclusters in gene expression data based on high-dimensional linear geometries," *BMC Bioinformatics*, vol. 9, no. 209, pp. 1–15, 2008.

[7] H. Zhao, A. W. C. Liew, X. Xie, and H. Yan, "A new geometric biclustering algorithm based on the Hough transform for analysis of large-scale microarray data," *J. Theoretical Biol.*, vol. 251, no. 3, pp. 264–274, 2008.

[8] H. Li and H. Yan, "Bicluster analysis of currency exchange rates," in *Soft Computing Applications in Business*, B. Prasad, Ed. New York: Springer-Verlag, 2008, pp. 19–34.

[9] H. Zhao, K. L. Chan, L. M. Cheng, and H. Yan, "A probabilistic relaxation labeling framework for reducing the noise effect in geometric biclustering of gene expression data," *Pattern Recognit.*, vol. 42, no. 11, pp. 2578–2588, 2009.

[10] J. van Rosmalen, P. J. F. Groenen, J. Trejos, and W. Castillo, "Optimization strategies for two-mode partitioning," *J. Classification*, vol. 26, no. 2, pp. 155–181, 2009.

[11] A. Li and D. Tuck, "An effective tri-clustering algorithm combining expression data with gene regulation information," *Gene Regulation Syst. Biol.*, vol. 2009, no. 3, pp. 49–64, 2009.

[12] H. Wang, F. Nie, H. Huang, and C. Ding, "Nonnegative matrix tri-factorization based high-order co-clustering and its fast implementation," in *Proc. IEEE 11th Int. Conf. Data Mining*, 2011, pp. 774–783.

[13] Z. Li and X. Wu, "Weighted nonnegative matrix tri-factorization for co-clustering," in *Proc. IEEE Int. Conf. Tools Artificial Intelligence*, 2011, pp. 811–816.

[14] W. H. Yang, D. Q. Dai, and H. Yan, "Finding correlated biclusters from gene expression data," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 4, pp. 568–584, 2011.

[15] Z. Wang, C. W. Yu, R. C. C. Cheung, and H. Yan, "Hypergraph based geometric biclustering algorithm," *Pattern Recognit. Lett.*, vol. 33, no. 12, pp. 1656–1665, 2012.

[16] H. Zhao, A. W. C. Liew, D. Z. Wang, and H. Yan, "Biclustering analysis for pattern discovery: Current techniques, comparative studies, and applications," *Current Bioinformatics*, vol. 7, no. 1, pp. 43–55, 2012.

[17] W. Ayadi, M. Elloumi, and J. K. Hao, "Pattern-driven neighborhood search for biclustering of microarray data," *BMC Bioinformatics*, vol. 13, no. 7, pp. S11, 2012.

[18] F. Shang, L. C. Jiao, and F. Wang, "Graph dual regularization non-negative matrix factorization for co-clustering," *Pattern Recognit.*, vol. 45, no. 6, pp. 2237–2250, 2012.

[19] D. Z. Wang and H. Yan, "A graph spectrum based geometric biclustering algorithm," *J. Theoretical Biol.*, vol. 317, pp. 200–211, Jan. 2013.

[20] K. Eren, M. Deveci, O. Küçüktunç, and Ü. V. Çatalyürek, "A comparative analysis of biclustering algorithms for gene expression data," *Briefings Bioinformatics*, vol. 14, no. 3, pp. 279–292, 2013.

[21] E. E. Papalexakis, N. D. Sidiropoulos, and R. Bro, "From k-means to higher-way co-clustering multilinear decomposition with sparse latent factors," *IEEE Trans. Signal Process.*, vol. 61, no. 2, pp. 493–506, 2013.

[22] B. Liu, Y. Xin, R. C. C. Cheung, and H. Yan, "GPU-based biclustering for microarray data analysis in neurocomputing," *Neurocomputing*, vol. 134, pp. 239–246, June 2014.

[23] B. Liu, C. Wai, D. Wang, R. Cheung, and H. Yan, "Design exploration of geometric biclustering for microarray data analysis in data mining," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 10, pp. 2540–2550, 2014.

[24] A. Oghabian, S. Kilpinen, S. Hautaniemi, and E. Czeizler, "Biclustering methods: Biological relevance and application in gene expression analysis," *PLoS ONE*, vol. 9, no. 3, p. e90801, 2014.

[25] H. Zhao, D. D. Wang, L. Chen, X. Liu, and H. Yan, "Identifying multidimensional co-clusters in tensors based on hyperplane detection in singular vector spaces," *PLoS ONE*, vol. 11, no. 9, p. e0162293, 2016.

[26] S. Khan, L. Chen, X. Zhe, and H. Yan, "Feature selection based on co-clustering for effective facial expression recognition," in *Proc. 16th IEEE Int. Conf. Machine Learning and Cybernetics*, 2016, pp. 48–53.

[27] H. Li, M. A. Lavin, and R. J. Le Master, "Fast Hough transform: a hierarchical approach," *Comput. Vis. Graph. Image Process.*, vol. 36, no. 2–3, pp. 139–161, 1986.

[28] A. Goldenshluger and A. Zeevi, "The Hough transform estimator," *Ann. Statist.*, vol. 32, no. 5, pp. 1908–1932, 2004.

[29] L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM J. Matrix Anal. Appl.*, vol. 21, no. 4, pp. 1253–1278, 2000.

[30] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Rev.*, vol. 51, no. 3, pp. 455–500, 2009.

[31] G. Bergqvist and E. G. Larsson, "The higher-order singular value decomposition: Theory and an application," *IEEE Signal Process. Mag.*, vol. 27, no. 3, pp. 151–154, 2010.

[32] S. P. Ponnapalli, M. A. Saunders, C. F. Van Loan, and O. Alter, "A higher-order generalized singular value decomposition for comparison of global mRNA expression from multiple organisms," *PloS ONE*, vol. 6, no. 12, pp. e28072, 2012.

SMC