Parameter-Free Hierarchical Co-clustering by n-Ary Splits

Dino Ienco, Ruggero G. Pensa, and Rosa Meo

University of Torino,
Department of Computer Science,
I-10149, Turin, Italy
{ienco,pensa,meo}@di.unito.it

Abstract. Clustering high-dimensional data is challenging. Classic metrics fail in identifying real similarities between objects. Moreover, the huge number of features makes the cluster interpretation hard. To tackle these problems, several co-clustering approaches have been proposed which try to compute a partition of objects and a partition of features simultaneously. Unfortunately, these approaches identify only a predefined number of flat co-clusters. Instead, it is useful if the clusters are arranged in a hierarchical fashion because the hierarchy provides insides on the clusters. In this paper we propose a novel hierarchical co-clustering, which builds two coupled hierarchies, one on the objects and one on features thus providing insights on both them. Our approach does not require a pre-specified number of clusters, and produces compact hierarchies because it makes $n-$ary splits, where n is automatically determined. We validate our approach on several high-dimensional datasets with state of the art competitors.

1 Introduction

Clustering is a popular data mining technique that enables to partition data into groups (clusters) in such a way that objects inside a group are similar to each other, and objects belonging to different groups are dissimilar [1]. When data are represented in a high-dimensional space, traditional clustering algorithms fail in finding an optimal partitioning because of the problem known as curse of dimensionality. Even though some distance metrics have been proposed to deal with high-dimensional data (e.g., cosine similarity) and feature selection tries to solve the problem by a reduction in the number of features, novel approaches emerged in the last years. One of the most appealing approach is co-clustering [2,3,4] whose solution provides contemporaneously a clustering of the objects and a clustering of the features. Furthermore, co-clustering algorithms are powerful because they exploit similarity measures on the clusters in one dimension of the problem in order to cluster the other dimension: that is, clusters of objects are evaluated by means of the clusters on the features and vice versa.

One of the classical aims of clustering is to provide a description of the data by means of an abstraction process. In many applications, the end-user is used to

W. Buntine et al. (Eds.): ECML PKDD 2009, Part I, LNAI 5781, pp. 580–595, 2009.
© Springer-Verlag Berlin Heidelberg 2009

study natural phenomena by the relative proximity relationships existing among the analyzed objects. For instance, he/she compares animals by means of the relative similarity in terms of the common features w.r.t. a same referential example. Many hierarchical algorithms have the advantage that are able to produce a dendrogram which stores the history of the merge operations (or split) between clusters. As a result they produce a hierarchy of clusters and the relative position of clusters in this hierarchy is meaningful because it implicitly tells the user about the relative similarity between the cluster elements. This hierarchy is often immediately understandable: it constitutes a helpful conceptual tool to understand the inner, existing relationships among objects in the domain; it provides a visual representation of the clustering result and explains it. Furthermore, it provides a ready to use tool to organize the conceptual domain, to browse and search objects, discover their common features or differences, etc. It is a conceptual tool especially advisable if one cluster hierarchy - built on one dimension of the problem, the objects - gives insights to study the other dimension of the problem - the features - and gives information to produce the feature hierarchy. In this paper we propose a co-clustering algorithm that simultaneously produces a hierarchical organization in both the problem dimensions: the objects and the features. In many applications both hierarchies are extremely useful and are searched for: in text mining, for instance, documents are organized in categories grouping related documents. The resulting object hierarchy is useful because it gives a meaningful structure to the collection of documents. On the other side, keywords are organized in groups of synonyms or words with related meaning and this hierarchy provides a semantic network with meaningful insights on the relationships between keywords. In bioinformatics and in other applications, a similar discussion applies: genes or proteins are grouped in groups sharing a similar behavior while biological experiments by their related, involved functionalities. The key contributions of this paper are the following. We present a novel co-clustering algorithm, *HiCC*, that simultaneously produces two hierarchies of clusters: one on the objects and the other one on the features. As we will see with the aid of the experimental section, these hierarchies are meaningful to the end-user and are valid also under the viewpoint of several objective measures. Our algorithm is able to produce compact hierarchies because it produces $n-$ary splits in the hierarchy instead of the usual binary splits. This compactness improves the readability of the hierarchy. The third contribution is the adoption of an unusual cluster association measure in co-clustering: Goodman-Kruskal τ. It has been considered as a good choice in a comparative study on several evaluation function for co-clustering [5,6]. The last novelty is that our algorithm is parameter-less and it does not require the user to set a number of clusters for the two dimensions, which usually require a pre-processing stage and is not easy to set.

2 An Introductory Example to Co-clustering

Goodman-Kruskal τ [7] has been originally proposed as a measure of association between two categorical variables: it is a measure of proportional reduction in

	F_1	F_2	F_3
O_1	d_{11}	d_{12}	d_{13}
O_2	d_{21}	d_{22}	d_{23}
O_3	d_{31}	d_{32}	d_{33}
O_4	d_{41}	d_{42}	d_{43}

(a)

	CF_1	CF_2	
CO_1	t_{11}	t_{12}	T_{O_1}
CO_2	t_{21}	t_{22}	T_{O_2}
	T_{F_1}	T_{F_2}	T

(b)

Fig. 1. An example dataset (a) and a related co-clustering (b)

the prediction error of a dependent variable given information on an independent variable. We intend to use τ as measure of validation of a co-clustering solution that produces an association between two clusterings: the former on the values of the independent variable, the latter on the values of the dependent one.

We start our discussion by presenting a dataset example and one co-clustering on it. In Figure 1(a) we show a matrix whose rows represent the objects and the columns represent the features describing the objects themselves. d_{xy} is the frequency of the y-th feature in the x-th object.

Now consider table in Figure 1(b): it is the contingency table representing a co-clustering of the dataset of Figure 1(a). Symbol CO_i represents i-th cluster on objects while CF_j represents j-th cluster on features. T_{Oi} is the total counting for cluster CO_i, T_{Fj} is the total counting for cluster CF_j, and T is the global total. In this example we suppose CO_1 has aggregated objects O_1 and O_2 while CO_2 has aggregated objects O_3 and O_4. Similarly, CF_1 has aggregated features F_1 and F_2 while CF_2 contains only feature F_3. Co-cluster (CO_i, CF_j) is represented by the value t_{ij} stored in the cell at the intersection of the i-th object cluster and the j-th feature cluster. It has been computed by application of an aggregating function on features and on objects. The aggregation that produces co-cluster (CO_i, CF_j) is the following:

$$t_{ij} = \sum_{O_x \in CO_k} \sum_{F_y \in CF_l} d_{xy}$$

In order to review the meaning of τ for the evaluation of a co-clustering consider the table in Figure 1(b) whose cells at the intersection of the row CO_i with the column CF_j contain the frequencies of objects in cluster CO_i having the features in cluster CF_j.

$\tau_{CO|CF}$ determines the predictive power of the partitioning on features, CF, considered as an independent variable, for the prediction of the partitioning on objects, CO, considered as a dependent variable. The prediction power of CF is computed as a function of the error in the classification of CO.

The prediction error is first computed when we do not have any knowledge on the partitioning CF. E_{CO} denotes this error here. The reduction of this error allowed by CF is obtained by subtraction from E_{CO} of the error in the prediction of cluster in CO that we make when we have the knowledge of the cluster in CF (the independent variable) in any database example. $E_{CO}|CF$ denotes this latter error. The proportional reduction in prediction error of CO given CF, here called $\tau_{CO|CF}$, is computed by:

$$\tau_{CO|CF} = \frac{E_{CO} - E_{CO}|CF}{E_{CO}}$$

E_{CO} and $E_{CO}|CF$ are computed by a predictor which uses information from the cross-classification frequencies and tries to reduce as much as possible the prediction error. In the prediction, it also preserves the dependent variable distribution (relative frequencies of the predicted categories CO) in the following way: when no knowledge is given on CF, CO_i is predicted by the relative frequency T_{Oi}/T; otherwise, when CF_j is known for an example, CO_i is predicted with the relative frequency t_{ij}/T_{Fj}. Therefore, E_{CO} and $E_{CO}|CF$ are determined by:

$$E_{CO} = \sum_i \left(\frac{T - T_{Oi}}{T} \cdot T_{Oi} \right) ; E_{CO}|CF = \sum_i \sum_j \left(\frac{T_{Fj} - t_{ij}}{T_{Fj}} \cdot t_{ij} \right)$$

Analyzing the properties of τ, we can see that it satisfies many desirable properties for a measure of association between clusters. For instance, it is invariant by rows and columns permutation. Secondarily, it takes values between (0,1) (it is 0 iff there is independence between the object and feature clusters). Finally, it has an operational meaning: given an example, it is the relative reduction in the prediction error of the example's cluster in one of the problem dimensions, given the knowledge on the example's cluster in the other dimension, in a way that preserves the class distribution.

3 Original Algorithm

Before introducing our approach, we specify the notation used in this paper. Let $X = \{x_1, \ldots, x_m\}$ denote a set of m objects (rows) and $Y = \{y_1, \ldots, y_n\}$ denote a set of n features (columns). Let D denote a $m \times n$ data matrix built over X and Y, each element of D representing a frequency or a count. Let $R = \{r_1, \ldots, r_I\}$ denote a partition of rows of D and $C = \{c_1, \ldots, c_J\}$ denote a partition of columns of D. Given R and C, a contingency table is denoted by T, i.e., a $I \times J$ matrix such that each element $t_{ij} \in T$ ($i \in 1 \ldots I$ and $j \in 1 \ldots J$) is computed as specified in Section 2.

We build our algorithm on the basis of $\tau CoClust$ [6], whose goal is to find a partition of rows R and a partition of columns C such that Goodman-Kruskal's $\tau_{C|R}$ and $\tau_{R|C}$ are optimized [5,8]. $\tau CoClust$ is shown as algorithm 1.

In [8], the authors propose a heuristic to locally optimize the two coefficients. Given two co-clusterings matrices, T and T', the differential τ between T and T' is given by:

$$\Delta\tau_{C|R} = \tau_{C|R}(T) - \tau_{C|R}(T')$$

The local optimization strategy is sketched in Algorithm 2. For sake of brevity, we only present the optimization of the row partition. The optimization strategy of the column partition works in a similar way.

This algorithm is a stochastic optimization technique and it allows to obtain, starting from a given set of row clusters, another set of row clusters that

optimizes the objective function. As first step, the algorithm chooses a cluster r_b randomly from the set of the initial row clusters R. Then it randomly picks an object x belonging to r_b. The original features of x are projected onto the current column clusters assignment. Then, the function tries to move x from the original cluster r_b to another cluster r_e s.t. $r_e \neq r_b$ and $r_e \in \{R \cup \emptyset\}$. Using function $\Delta_{\tau_{R|C}}(x, r_b, r_e, C)$, it verifies if there is an increment in the objective function as a result of this change. Then, it chooses cluster $r_{e_{min}}$ s.t. $r_{e_{min}} = \arg\min_{r_e} \Delta_{\tau_{R|C}}(x, r_b, r_e, C)$. Finally, the function updates the contingency table T and the row cluster partition (line 11). To perform this step, it first moves the object x from the cluster r_b to cluster $r_{e_{min}}$. Then it performs one or more of the following three actions:

1. if cluster r_b is empty after removing x, it deletes cluster r_b and updates the contingency table T consequently;
2. if cluster $r_{e_{min}}$ is the empty cluster, it adds a new cluster to the row partition and updates the contingency table T consequently;
3. if the two above mentioned cases do not apply, it simply updates the content of T in a consequent way.

Thanks to this strategy, the number of clusters may grow or decrease at each step, and their number only depends on the effective optimization of $\tau_{R|C}$. This makes this approach substantially different from the one described in [4] and in other state-of-the-art co-clustering approaches, where the number of co-clusters is fixed as a user-defined parameter. For a deep view on the efficient version of this algorithm see [8].

Algorithm 1. $\tau CoClust(D, N_{iter})$

```
1: Initialize R, C
2: T ← ContingencyTable(R, C, D)
3: while (t ≤ N_iter) do
4:    optimizeRowCluster(R, C, T)
5:    optimizeColumnCluster(R, C, T)
6:    t ← t + 1
7: end while
8: return (R, C)
```

4 Hierarchical Co-clustering

In this section we present our method, named HiCC (Hierarchical Co-Clustering by n-ary split). Let us introduce the notation.

4.1 Notations

Given the above described matrix D defined on the set of objects $X=\{x_1, \ldots, x_m\}$ and on the set of features $Y = \{y_1, \ldots, y_n\}$, the goal of our hierarchical

Algorithm 2. *optimizeRowCluster*(R, C, T)

$min_{\Delta\tau_{R|C}} = 0$
Randomly choose a cluster $r_b \in R$
Randomly choose an object $x \in r_b$
for all $r_e \in \{R \cup \emptyset\} s.t. r_b \neq r_e$ **do**
  if $(\Delta_{\tau_{R|C}}(x, r_b, r_e, C) < min_{\Delta\tau_{R|C}})$ **then**
    $e_{min} = e$
    $min_{\Delta\tau_{R|C}} = \Delta_{\tau_{R|C}}(x, r_b, r_e, C)$
  end if
end for
if $min_{\Delta\tau_{R|C}} \neq 0$ **then**
  Update R using (x,r_b,$r_{e_{min}}$) and modify T consequently
end if

Algorithm 3. $HiCC(D, N_{iter})$

$k \leftarrow 0$, $l \leftarrow 0$, $\mathcal{R} \leftarrow \emptyset$, $\mathcal{C} \leftarrow \emptyset$
$(R_k, C_l) \leftarrow \tau CoClust(D, N_{iter})$
$\mathcal{R} \leftarrow \mathcal{R} \cup R_k$, $\mathcal{C} \leftarrow \mathcal{C} \cup C_l$
while $(TERMINATION)$ **do**
  $R_{k+1} \leftarrow \emptyset$
  for all $r_{ki} \in R_k$ **do**
    $R'_i \leftarrow RandomSplit(r_{ki})$
    $t \leftarrow 0$
    $T_{ki} \leftarrow ContingencyTable(R'_i, C_l, D)$
    while $(t \leq N_{iter})$ **do**
      $optimizeRowCluster(R'_i, C_l, T_{ki})$
      $t \leftarrow t + 1$
    end while
    $R_{k+1} \leftarrow R_{k+1} \cup R'_i$
  end for
  $\mathcal{R} \leftarrow \mathcal{R} \cup R_{k+1}$
  $k \leftarrow k + 1$
  $C_{l+1} \leftarrow \emptyset$
  for all $c_{lj} \in C_l$ **do**
    $C'_j \leftarrow RandomSplit(c_{lj})$
    $t \leftarrow 0$
    $T_{lj} \leftarrow ContingencyTable(R_k, C'_j, D)$
    while $(t \leq N_{iter})$ **do**
      $optimizeColumnCluster(R_k, C'_j, T_{lj})$
      $t \leftarrow t + 1$
    end while
    $C_{l+1} \leftarrow C_{l+1} \cup C'_j$
  end for
  $\mathcal{C} \leftarrow \mathcal{C} \cup C_{l+1}$
  $l \leftarrow l + 1$
end while
return $(\mathcal{R}, \mathcal{C})$

co-clustering algorithm is to find a hierarchy of row clusters $\mathcal{R}$ over X, and a hierarchy of column clusters $\mathcal{C}$ over Y. Supposing that $\mathcal{R}$ has K levels, and that $\mathcal{C}$ has L levels, $\mathcal{R}$ and $\mathcal{C}$ are defined as $\mathcal{R} = \{R_1, \ldots, R_K\}$ and $\mathcal{C} = \{C_1, \ldots, C_L\}$ (where R_1 and C_1 are the clustering at the roots of the respective hierarchy). Each $R_k \in \mathcal{R}$ is a set of clusters denoted by $R_k = \{r_{k1}, \ldots, r_{k|R_k|}\}$ where $|R_k|$ is the total number of clusters in R_k, $r_{ki} \subseteq X$, $\bigcup_i r_{ki} = X$ and $\forall i, j$ s.t. $i \neq j$ $r_{ki} \cap r_{kj} = \emptyset$. Similarly, $C_l = \{c_{l1}, \ldots, c_{l|C_l|}\}$, and the other conditions hold for C_l too. Since $\mathcal{R}$ defines a hierarchy, each R_k must also satisfy the following conditions:

1. $\forall R_{k_1}, R_{k_2}$ s.t. $k_1 < k_2$, $|R_{k_1}| \leq |R_{k_2}|$;
2. $\forall R_k$ $(k > 1)$, $\forall r_{ki} \in R_k$, $\forall R_{k_0}$ $(k_0 < k)$, $\exists! r_{k_0 j \in R_{k_0}}$ s.t. $r_{ki} \subseteq r_{k_0 j}$;

Two similar conditions must hold for $\mathcal{C}$ too.

Our approach first computes the first level of the hierarchy (R_1 and C_1) using Algorithm 1. Then, it builds R_2 by fixing C_1 by optimization of $\tau_{R_2|C_1}$. In general, given a generic hierarchy level h, Algorithm 3 alternates the optimization of $\tau_{R_h|C_{h-1}}$ and $\tau_{C_h|R_h}$, finding the appropriate row cluster R_h and column cluster C_h, constrained by the two above mentioned conditions. We present now the details of our algorithm.

4.2 Algorithm Description

The whole procedure is presented in Algorithm 3. HiCC adopts a divisive strategy. At line 1 it initializes all our indices and structures. Using function $\tau CoClust$ (see Algorithm 1) it builds the first level of both hierarchies.

From line 4 to line 31 it builds the levels of the two hierarchies. From line 6 to line 13, for each row cluster $r_{ki} \in R_k$, the algorithm splits it into a new set of row clusters R'_i using *RandomSplit* function. This function first sets the cardinality R'_i randomly; then it randomly assigns each row in r_{ki} to a cluster of R'_i. Subsequently, it initializes a new contingency table T_{ki} related to the set R'_i of row clusters and to the set C_l of column clusters (we consider all the column clusters found at previous level). Without changing columns partition, the algorithm tries to optimize $\tau_{R'_i|C_l}$ using the *optimizeRowCluster* function (see Algorithm 2). It returns a new and optimized R'_i. After all r_{ki} have been processed, the algorithm adds the new level of the row hierarchy to $\mathcal{R}$ (line 16). Column clusters are then processed in the same way, using the row cluster assignment returned at the end of previous steps. In this way the two hierarchies grow until a TERMINATION condition is reached. The TERMINATION condition is satisfied when all leaves of the two hierarchies contain only one element. Obviously, some cluster may be split into singletons at higher levels than others. At the end, our algorithm returns both the hierarchies over rows ($\mathcal{R}$) and columns ($\mathcal{C}$).

As shown in [8,6], the local search strategy employed to update partitions, sometimes leads to some degradation of $\tau_{R|C}$ or $\tau_{C|R}$. This is due to the fact that an improvement on one partition may decrease the quality of the other one. The authors, however, showed that there is always a compensation effect such that, after an adequate number of iterations, the two coefficients become stable.

In our approach, a single cluster is partitioned while keeping the other partition fixed. This ensure that the corresponding coefficient increases at each iteration.

4.3 Complexity Discussion

HiCC complexity is influenced by three factors. The first one is the number of iterations. It influences the convergence of the algorithm. A deep study of the number of iterations is presented in [6]. The second factor is the number of row/column clusters in *optimizeRowCluster* and in *optimizeColCluster* functions. This number of clusters influences the swap step which tries to optimize the internal objective function moving one object from a cluster b to another cluster e. The number of possible different values that e can assume influences the speed of the algorithm, but this number varies during the optimization. The third factor is the depth of the two hierarchies, in particular the deeper one. This value influences the number of times the main loop from lines 4-31 of algorithm 3 is repeated. If we denote by N the number of iterations, by $\widetilde{c}$ the mean number of row/column clusters inside the optimization procedure, and by v the mean branching factor, we observe that the complexity of a split of r_{ki} and c_{lj} is equal as average to $O(N \times \widetilde{c})$. Each split is performed for each node of the two hierarchies except for the bottom level. The number of nodes in a tree with branching factor v is $\sum_{i=0}^{levels} v^i$ where $levels$ is the number of levels of the tree. We can expand this summation and we obtain $\frac{1-v^{1+levels}}{1-v}$. From the previous consideration we estimate the complexity of our approach $O\left(N \times \widetilde{c} \times \frac{1-v^{levels}}{1-v}\right)$. The worst case is verified when any split is binary and at each level $\widetilde{c} = n$, where n is the number of rows/columns. In this case, the complexity is $O(N \times (n-1) \times n)$, $(n-1)$ being the number of internal nodes in the hierarchy. In conclusion, in the worst case, the overall complexity is $O(Nn^2)$. In the experimental section, we show that our algorithm often splits into more than two clusters, thus reducing the number of internal nodes. Moreover, assumption $\widetilde{c} = n$ is very pessimistic. In general our algorithm runs in linear time with the number of iterations, and in subquadratic time with the number of objects/features. Notice also that in this work, the number of iterations is constant, but it could be adapted to the size of the set of objects, and then reduced at each split.

5 Experimental Validation

In this section we report on several experiments performed on real, high-dimensional, multi-class datasets. We compare our approach with Information-Theoretic Co-Clustering (ITCC) [4], a well-known co-clustering algorithm which minimizes the loss in mutual information. To evaluate our results, we use several objective performance parameters that measure the quality of clustering. Besides the precision of the hierarchical co-clustering we analyze also the hierarchies returned by our approach for both rows and columns. This analysis allows to emphasize the utility of the hierarchical structure w.r.t. standard flat

Table 1. Datasets characteristics

Dataset	n. instances	n. attributes	n. of classes
oh0	1003	3182	10
oh15	913	3100	10
tr11	414	6429	9
tr21	336	7902	6
re0	1504	2886	13
re1	1657	3758	25

co-clustering approaches. All experiments are performed on PC with a 2.6GHz Opteron processor, 4GB RAM, running Linux.

5.1 Datasets for Evaluation

To evaluate our results we use some of the datasets described in [9]. In particular we use:

- **oh0**, **oh15**: two samples from OHSUMED dataset. OHSUMED is a clinically-oriented MEDLINE subset of abstracts or titles from 270 medical journals over five-year period (1987-1991).
- **tr11**, **tr21**: two samples from TREC dataset. These data come from the Text REtrieval Conference archive.
- **re0**, **re1**: two samples from Reuters-21578 dataset. This dataset is widely used as test collection for text categorization research.

All datasets have more than five classes, which usually is a hard context for text categorization. The characteristics of datasets are shown in Table 1.

5.2 External Evaluation Measures

We evaluate the algorithm performance using three external validation indices. We denote by $\mathbf{C} = \{C_1 \dots C_J\}$ the partition built by the clustering algorithm on objects at a particular level, and by $\mathbf{P} = \{P_1 \dots P_I\}$ the partition inferred by the original classification. J and I are respectively the number of clusters $|\mathbf{C}|$ and the number of classes $|\mathbf{P}|$. We denote by n the total number of objects.

The first index is *Normalized Mutual Information* (NMI). NMI provides an information that is impartial with respect to the number of clusters [10]. It measures how clustering results share the information with the true class assignment. NMI is computed as the average mutual information between every pair of clusters and classes:

$$\mathbf{NMI} = \frac{\sum_{i=1}^{I}\sum_{j=1}^{J} x_{ij} \log \frac{n x_{ij}}{x_i x_j}}{\sqrt{\sum_{i=1}^{I} x_i \log \frac{x_i}{n} \sum_{j=1}^{J} x_j \log \frac{x_j}{n}}}$$

where x_{ij} is the cardinality of the set of objects that occur both in cluster C_j and in class P_i; x_j is the number of objects in cluster C_j; x_i is the number of objects in class P_i. Its values range between 0 and 1.

The second measure is *purity*. In order to compute purity each cluster is assigned to the majority class of the objects in the cluster. Then, the accuracy of this assignment is measured by counting the number of correctly assigned objects divided by the total number of objects n.

$$\mathbf{Purity}(\mathbf{C}, \mathbf{P}) = \frac{1}{n} \sum_j max_i |C_j \cap P_i|$$

The third measure is the adjusted Rand index [11]. Let a be the number of object pairs belonging to the same cluster in $\mathbf{C}$ and to the same class in $\mathbf{P}$. This metric captures the deviation of a from its expected value corresponding to the hypothetic value of a obtained when $\mathbf{C}$ and $\mathbf{P}$ are two random, independent partitions. The expected value of a denoted by $E[a]$ is computed as follows:

$$E[a] = \frac{\pi(C) \cdot \pi(P)}{n(n-1)/2}$$

where $\pi(C)$ and $\pi(P)$ denote respectively the number of object pairs from the same clusters in $\mathbf{C}$ and from the same class in $\mathbf{P}$. The maximum value for a is defined as:

$$max(a) = \frac{1}{2}(\pi(C) + \pi(P))$$

The agreement between $\mathbf{C}$ and $\mathbf{P}$ can be estimated by the adjusted rand index as follows:

$$AR(\mathbf{C}, \mathbf{P}) = \frac{a - E[a]}{max(a) - E[a]}$$

Notice that this index can take negative values, and when $AR(\mathbf{C}, \mathbf{P}) = 1$, we have identical partitions.

5.3 Comparison Results

In this section we evaluate HiCC performance w.r.t ITCC [4]. HiCC is not deterministic like other well-known clustering algorithms such as K-means or ITCC itself. At each run we can obtain similar, but not equal, hierarchies. For this reason we run HiCC 30 times over each dataset (setting the number of iterations to 50,000). From these row/column co-hierarchies we choose the run that better optimizes an internal evaluation function.

To obtain a measure of quality of HiCC, for each level i of the hierarchy on the rows we select the corresponding level of the hierarchy on the columns. These levels define a pair of partitions: the first partition comes from the row cluster hierarchy, and the second one from the columns cluster hierarchy. On the pair of partitions from level i, an evaluation function EF_i is computed on the basis of Goodman-Kruskal $\tau_{\mathcal{S}}$ [12], which is a symmetrical version of τ [7]. In order to compute an overall measure for the co-clustering we compute the following weighted mean:

$$Goodness = \frac{\sum_{i=1} \alpha_i * EF_i}{\sum_{i=1} \alpha_i} \tag{1}$$

Table 2. Average values of External Indices

Dataset	NMI		Purity		Adjusted Rand Index	
	Avg.	std. dev.	Avg.	std. dev.	Avg.	std. dev.
oh0	0.5176	±0.0134	0.7768	±0.0250	0.1150	±0.0129
oh15	0.4223	±0.0103	0.6653	±0.0207	0.0790	±0.0080
tr11	0.4606	±0.0087	0.7510	±0.0145	0.1091	±0.0119
tr21	0.2387	±0.0120	0.8103	±0.0122	0.0323	±0.0045
re0	0.3588	±0.0273	0.7317	±0.0235	0.0381	±0.0140
re1	0.5005	±0.0267	0.7616	±0.0397	0.0714	±0.017

where α_i is the weight associated to i-th level of the hierarchy, and allows to specify the significance assigned to the i-th level w.r.t. to the other levels of the hierarchy.

(1) is a general formula for the evaluation the goodness of a run of our method. In this work we set $\alpha_i = 1/i$. Indeed, we give a heavy weight to the top level and the lowest weight to the last level. This choice is motivated by observation that in a hierarchical solution the clusters on a level depend on the clusters at previous level. If we start with a good clustering, then next levels are more likely to produce good results too.

To compare each level of our hierarchies with ITCC results, we need to fix a number of row/column clusters to set ITCC parameters. We recall that ITCC is flat and does not produce hierarchies. For this reason we plan our experiments in the following way. Since HiCC is not deterministic, each run may produce partitions of different cardinality at each level. For this reason, we need to select one specific run of HiCC. Using *Goodness* function with $\tau_{\mathcal{S}}$ as evaluation function, we choose the best hierarchies from HiCC for each dataset. From these hierarchies, we obtain a set of pairs (#numberRowCluster,#numberColCluster), where each pair specifies the number of clusters in a level of the hierarchies. For each of these combinations, we run ITCC 50 times with (#numberRowCluster,#numberColCluster) as parameters, and average for each level the obtained results.

In Table 3 we show the experimental results. To obtain a single index value for each dataset we compute the previously proposed *Goodness* function for each of the three external validation indices. We can see that our approach is competitive w.r.t. ITCC. Notice however that, for a given number of desired co-clusters, ITCC tries to optimize globally its objective function, and each time it starts from scratch. On the contrary, our algorithm is constrained by previous level which helps to speed-up its execution.

We can notice that ITCC is not more accurate than our algorithm. To clarify this point we report the complete behavior of the two algorithms in an example. In Table 4 we report the value of the three indices for each level of the hierarchy obtained by HiCC for **re1**. In the same table we show the values obtained by ITCC using the same number of clusters (#numberRowCluster,#numberColCluster) discovered by HiCC. We also report the standard deviation for ITCC, since for each pair of cluster numbers, we run it 50 times. We can see that HiCC outperforms ITCC, especially at the higher levels (first, second and third) of the row hierarchy.

Table 3. Comparison between ITCC and HiCC with the same number of cluster for each level

Dataset	ITCC			HiCC		
	NMI	Purity	Adj. Rand Index	NMI	Purity	Adj. Rand Index
oh0	0.4311	0.5705	0.1890	0.4607	0.5748	0.1558
oh15	0.3238	0.4628	0.1397	0.3337	0.4710	0.1297
tr11	0.3861	0.5959	0.1526	0.3949	0.6028	0.1325
tr21	0.1526	0.7291	0.0245	0.1277	0.7332	0.0426
re0	0.2820	0.5695	0.0913	0.2175	0.5388	0.0472
re1	0.3252	0.4957	0.0832	0.3849	0.5513	0.1261

Table 4. Complete view of performance parameters for **re1**

		ITCC			HiCC		
RowClust	ColClust	NMI	Purity	Adj. Rand	NMI	Purity	Adj. Rand
3	3	0.1807±0.0519	0.3282±0.0448	0.1028±0.0466	0.3290	0.4255	0.2055
6	6	0.2790±0.0462	0.3991±0.0393	0.1723±0.0530	0.2914	0.4255	0.1828
15	257	0.2969±0.0178	0.4357±0.0226	0.1358±0.0183	0.3098	0.4472	0.1555
330	1291	0.3499±0.0028	0.4013±0.0059	0.0021±0.0003	0.3950	0.51	0.0291
812	2455	0.4857±0.0019	0.5930±0.0056	0.0013±0.0002	0.4810	0.6530	0.0031
1293	3270	0.5525±0.0013	0.8284±0.0041	0.0008±0.0001	0.5517	0.8461	0.0009
1575	3629	0.5864±0.0006	0.9602±0.0020	0.0005±0	0.5854	0.9638	0.0001
1646	3745	0.5944±0.0002	0.9926±0.0008	0.0004±0	0.5940	0.9952	0
1657	3757	0.5951±0	1±0	0±0	0.5952	1	0
1657	3758	0.5951±0	1±0	0±0	0.5952	1	0

We notice also that *purity* index in HiCC always increases monotonically. This experiment shows that, when we explore deeper levels of the hierarchy, the confusion inside each cluster decreases. For sake of brevity we can only show one example, but in all the other experiments we observe the same trend.

In Table 5 we report the average Goodness of τ_S for each dataset, and the related standard deviation. We observe that standard deviation is very low w.r.t. the average Goodness. From this empirical evaluation we can conclude that the algorithm is quite stable.

In Table 2 we show the average values of the three external indices for each dataset. To perform this analysis we first compute the average of NMI, Purity and Adjusted Rand Index between all levels of the row hierarchy and for each run of the algorithm. Then we compute the average and the standard deviation over all 30 runs.

Finally, we report in Table 5 the average time needed to generate the two hierarchies for each dataset. The largest dataset is **re1**, and for this dataset HiCC takes about 4 hours to complete a run. In the same table we show the mean depth of the row hierarchy and of the column hierarchy for each dataset. We observe that the standard deviation is low. This points out that our algorithm is stable also from this point of view. Notice that HiCC has the ability to generate hierarchies which are not deep. Shorter hierarchies are preferable to hierarchies obtained only by binary splits, since they allow to identify the natural partitioning of the data, and improve the exploration of the results.

Table 5. Average τ_S, mean depth of hierarchies and mean time

Dataset	Goodness	Row Hier. Depth	Col. Hier. Depth	Avg. Time
oh0	0.2680±0.0063	9.37±0.71	9.83±0.64	6647.99 sec
oh15	0.2783±0.0012	10.63±0.71	11.23±0.62	5866.92 sec
tr11	0.2294±0.0032	10.27±0.63	15.63±1.99	5472.37 sec
tr21	0.3003±0.0012	10.87±0.67	14.4±0.99	5493.98 sec
re0	0.2350±0.0166	10.7±0.74	10.6±0.88	9359.04 sec
re1	0.1697±0.0112	8.8±0.65	9.2±0.65	14887.18 sec

Table 6. Row hierarchy of oh15

Enzyme-Activation	Enzyme-Activation	Enzyme-Activation
		Enzyme-Activation
	Cell-Movement	Cell-Movement
		Adenosine-Diphosphate
Staphylococcal-Infections	Uremia	Uremia
		Staphylococcal-Infections
	Staphylococcal-Infections	Staphylococcal-Infections
		Memory

Table 7. Column hierarchy of re1

oil, compani, opec, gold, ga, barrel, strike, mine, lt, explor		tonne, wheate, sugar, corn, mln, crop, grain, agricultur, usda, soybean		coffee, buffer, cocoa, deleg, consum, ico, stock, quota, icco, produc	
oil, opec, tax, price, dlr, crude, bank, industri, energi, saudi	compani, gold, mine, barrel, strike, ga, lt, ounce, ship, explor	tonne, wheate, sugar, corn, grain, crop, agricultur, usda, soybean, soviet	mln, export, farm, ec, import, market, total, sale, trader, trade	quota, stock, produc, meet, intern, talk, bag, agreem, negoti, brazil	coffee, deleg, buffer, cocoa, consum, ico, icco, pact, council, rubber

5.4 Inspecting Hierarchies

In this section we analyze in a qualitative way two hierarchies built by our algorithm. In particular we analyze the row hierarchy for **oh0** data and the column hierarchy from **re1** data. In Table 6 we report the first three levels of the row hierarchy produced by HiCC. To obtain this hierarchy we assigned at each cluster a label. We chose the label of the majority class in the cluster. Each column represents a level of the hierarchy and each cell of the table represents a single cluster. Cluster *Enzyme-Activation* at first level is split in two clusters: the first one has the same label, the second is *Cell-movement*, a topic more related to *Enzyme-Activation* than *Staphylococcal-Infections*. Then *Cell-movement* is split into two clusters. One of these is *Adenosine-Diphosphate*, which is known to be correlated with the topic *Cell-movement*. In the other branch of the hierarchy we notice that *Uremia* cluster is a child of *Staphylococcal-Infections*(a renal failure with bacterial infection as one of its causes). In Table 7 we report the first two levels of the column hierarchy produced by HiCC. For each cluster, we computed the mutual information for each word, and ranked the set of words, as described in [13]. Then we selected the 10-top words w.r.t. the mutual information for each

cluster. At the first level each of the three clusters is about a well distinct topic: the first one is about *mineral*, the second one is on *agriculture* and the third one is on *coffee and cocoa*. From *mineral* cluster, two clusters are produced: *oil*, and *gold mineral*. Cluster on *agriculture* is split into two clusters: *agricultural products* and *buying and selling in agriculture*. Finally, we observe that *coffee and cocoa* cluster is split in a similar way.

6 Related Work

One of the earliest co-clustering formulations was introduced by Hartigan [2]. This algorithm begins with the entire data in a single block and then at each stage finds the row or column split of every block into two pieces, choosing the one that produces largest reduction in the total within block variance. The splitting is continued till the reduction of within block variance due to further splitting is less than a given threshold. This approach is clearly hierarchical, but it does not build any cluster hierarchy. Moreover, it does not optimize any global objective function. Kluger et al. [3] propose a spectral co-clustering method. First, they perform an adequate normalization of the data set to accentuate co-clusters if they exist. Then, they consider that the correlation between two columns is better estimated by the expression level mean of each column w.r.t. a partition of the rows. The bipartition is computed by the algebraic eigenvalue decomposition of the normalized matrix. Their algorithm critically depends on the normalization procedure. Dhillon et al. [4] and Robardet et al. [8] have considered the two searched partitions as discrete random variables whose association must be maximized. Different measures can be used. Whereas Cocluster [4] uses the loss in mutual information, Bi-Clust [8] uses Goodman-Kruskal's τ coefficient to evaluate the link strength between the two variables. In both algorithms, a local optimization method is used to optimize the measure by alternatively changing a partition when the other one is fixed. The main difference between these two approaches is that the τ measure is independent of the number of co-clusters and thus Bi-Clust can automatically determine the number of co-clusters. Another co-clustering formulation was presented in [14]. Authors propose two different residue measure, and introduce their co-clustering algorithm which optimizes the sum-squared residues function. Recently, Banerjee et al. have proposed in [15] a co-clustering setting based on matrix approximation. The approximation error is measured using a large class of loss functions called Bregman divergences. They introduce a meta-algorithm whose special cases include the algorithms from [4] and [14]. Another recent and significant theoretical result has been presented in [16]. The authors show that the co-clustering problem is NP-hard, and propose a constant-factor approximation algorithm for any norm-based objective functions.

To the best of our knowledge, our approach is the first one that performs a simultaneous hierarchical co-clustering on both dimensions, and that returns two coupled hierarchies. However, in recent literature, several approaches have been proposed that could be related to our work, even though they do not

produce the same type of results. In [17] a hierarchical co-clustering for queries and URLs of a search engine log is introduced. This method first constructs a bipartite graph for queries and visited URLs, and then all queries and related URLs are projected in a reduced dimensional space by applying singular value decomposition. Finally, all connected components are iteratively clustered using *k-means* for constructing hierarchical categorization. In [18], the authors propose a hierarchical, model-based co-clustering framework that views a binary dataset as a joint probability distribution over row and column variables. Their approach starts by clustering tuples in a dataset, where each cluster is characterized by a different probability distribution. Then, the conditional distribution of attributes over tuples is exploited to discover natural co-clusters in the data. This method does not construct any coupled hierarchy, moreover, co-cluster are identified in a separate step, only after the set of tuple has been partitioned. In [19], a method is proposed that construct two hierarchies on gene expression data, but they are not generated simultaneously. In our approach, levels of the two hierarchies are alternately generated, so that each level of both hierarchies identifies a strongly related set of co-clusters of the matrix.

7 Conclusion

Quality of flat clustering solutions in high-dimensional data results often degraded. In this paper we have proposed a co-clustering approach. HiCC is a novel hierarchical algorithm, which builds two coupled hierarchies, one on the objects and one on features thus providing insights on both them. Hierarchies are high quality. We have validated them by objectives functions like NMI, Purity and Adjusted Rand Index on many high-dimensional datasets. In addition, HiCC has other benefits: it is parameter-less; it does not require a pre-specified number of clusters, produces compact hierarchies because it makes $n-$ary splits, with n automatically determined.

References

1. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, San Francisco (2000)
2. Hartigan, J.A.: Direct clustering of a data matrix. Journal of the American Statistical Association 67(337), 123–129 (1972)
3. Kluger, Y., Basri, R., Chang, J., Gerstein, M.: Spectral biclustering of microarray data: coclustering genes and conditions. Genome Research 13, 703–716 (2003)
4. Dhillon, I.S., Mallela, S., Modha, D.S.: Information-theoretic co-clustering. In: Proc. ACM SIGKDD 2003, Washington, USA, pp. 89–98. ACM, New York (2003)
5. Robardet, C., Feschet, F.: Comparison of three objective functions for conceptual clustering. In: Siebes, A., De Raedt, L. (eds.) PKDD 2001. LNCS (LNAI), vol. 2168, pp. 399–410. Springer, Heidelberg (2001)
6. Robardet, C.: Contribution à la classification non supervisée: proposition d'une methode de bi-partitionnement. PhD thesis, Université Claude Bernard - Lyon 1 (Juliet 2002)

7. Goodman, L.A., Kruskal, W.H.: Measures of association for cross classification. Journal of the American Statistical Association 49, 732–764 (1954)
8. Robardet, C., Feschet, F.: Efficient local search in conceptual clustering. In: Jantke, K.P., Shinohara, A. (eds.) DS 2001. LNCS (LNAI), vol. 2226, pp. 323–335. Springer, Heidelberg (2001)
9. Forman, G.: An extensive empirical study of feature selection metrics for text classification. Journal of Machine Learning Research 3, 1289–1305 (2003)
10. Strehl, A., Ghosh, J.: Cluster ensembles — a knowledge reuse framework for combining multiple partitions. Journal of Machine Learning Research 3, 583–617 (2002)
11. Hubert, L., Arabie, P.: Comparing partitions. Journal of Classification 2, 193–218 (1985)
12. Goodman, L.A., Kruskal, W.H.: Measure of association for cross classification ii: further discussion and references. Journal of the American Statistical Association 54, 123–163 (1959)
13. Slonim, N., Tishby, N.: Document clustering using word clusters via the information bottleneck method. In: Proc. SIGIR 2000, New York, NY, USA, pp. 208–215 (2000)
14. Cho, H., Dhillon, I.S., Guan, Y., Sra, S.: Minimum sum-squared residue co-clustering of gene expression data. In: Proc. SIAM SDM 2004, Lake Buena Vista, USA (2004)
15. Banerjee, A., Dhillon, I., Ghosh, J., Merugu, S., Modha, D.S.: A generalized maximum entropy approach to bregman co-clustering and matrix approximation. JMLR 8, 1919–1986 (2007)
16. Anagnostopoulos, A., Dasgupta, A., Kumar, R.: Approximation algorithms for co-clustering. In: Proc. PODS 2008, Vancouver, BC, Canada, pp. 201–210 (2008)
17. Hosseini, M., Abolhassani, H.: Hierarchical co-clustering for web queries and selected urls. In: Benatallah, B., Casati, F., Georgakopoulos, D., Bartolini, C., Sadiq, W., Godart, C. (eds.) WISE 2007. LNCS, vol. 4831, pp. 653–662. Springer, Heidelberg (2007)
18. Costa, G., Manco, G., Ortale, R.: A hierarchical model-based approach to co-clustering high-dimensional data. In: Proc. of ACM SAC 2008, Fortaleza, Ceara, Brazil, pp. 886–890 (2008)
19. Heard, N.A., Holmes, C.C., Stephens, D.A., Hand, D.J., Dimopoulos, G.: Bayesian coclustering of anopheles gene expression time series: Study of immune defense response to multiple experimental challenges. Proc. Natl. Acad. Sci. (102), 16939–16944