A CLUE for CLUster Ensembles

Kurt Hornik

2006-02-09

Abstract

Cluster ensembles are collections of individual solutions to a given clustering problem which are useful or necessary to consider in a wide range of applications. The R package **clue** provides an extensible computational environment for creating and analyzing cluster ensembles, with basic data structures for representing partitions and hierarchies, and facilities for computing on these, including methods for measuring proximity and obtaining consensus and "secondary" clusterings.

1 Introduction

Cluster ensembles are collections of clusterings, which are all of the same "kind" (e.g., collections of partitions, or collections of hierarchies), of a set of objects. Such ensembles can be obtained, for example, by varying the (hyper)parameters of a "base" clustering algorithm, by resampling or reweighting the set of objects, or by employing several different base clusterers.

Questions of "agreement" in cluster ensembles, and obtaining "consensus" clusterings from it, have been studied in several scientific communities for quite some time now. A special issue of the Journal of Classification was devoted to "Comparison and Consensus of Classifications" (Day, 1986) almost two decades ago. The recent popularization of ensemble methods such as Bayesian model averaging (Hoeting, Madigan, Raftery, and Volinsky, 1999), bagging (Breiman, 1996) and boosting (Friedman, Hastie, and Tibshirani, 2000), typically in a supervised leaning context, has also furthered the research interest in using ensemble methods to improve the quality and robustness of cluster solutions. Cluster ensembles can also be utilized to aggregate base results over conditioning or grouping variables in multi-way data, to reuse existing knowledge, and to accommodate the needs of distributed computing, see e.g. Hornik (2005a) and Strehl and Ghosh (2003a) for more information.

Package **clue** is an extension package for R (R Development Core Team, 2005) providing a computational environment for creating and analyzing cluster ensembles. In Section 2, we describe the underlying data structures, and the functionality for measuring proximity, obtaining consensus clusterings, and "secondary" clusterings. Four examples are discussed in Section 3. Section 4 concludes the paper.

A previous version of this manuscript was published in the *Journal of Statistical Software* (Hornik, 2005b).

2 Data structures and algorithms

2.1 Partitions and hierarchies

Representations of clusterings of objects greatly vary across the multitude of methods available in R packages. For example, the class ids ("cluster labels") for the results of `kmeans()` in base package **stats**, `pam()` in recommended package **cluster** (Rousseeuw, Struyf, Hubert, and Maechler, 2005; Struyf, Hubert, and Rousseeuw, 1996), and `Mclust()` in package **mclust** (Fraley, Raftery, and Wehrens, 2005; Fraley and Raftery, 2003), are available as components named `cluster`, `clustering`, and `classification`, respectively, of the R objects returned by these functions. In many cases, the representations inherit from suitable classes. (We note that for versions of R prior to 2.1.0, `kmeans()` only returned a "raw" (unclassed) result, which was changed alongside the development of **clue**.)

We deal with this heterogeneity of representations by providing getters for the key underlying data, such as the number of objects from which a clustering was obtained, and predicates, e.g. for determining whether an R object represents a partition of objects or not. These getters, such as `n_of_objects()`, and predicates are implemented as S3 generics, so that there is a *conceptual*, but no formal class system underlying the predicates. Support for classed representations can easily be added by providing S3 methods.

2.1.1 Partitions

The partitions considered in **clue** are possibly soft ("fuzzy") partitions, where for each object i and class j there is a non-negative number μ_{ij} quantifying the "belongingness" or *membership* of object i to class j, with $\sum_j \mu_{ij} = 1$. For hard ("crisp") partitions, all μ_{ij} are in $\{0, 1\}$. We can gather the μ_{ij} into the *membership matrix* $M = [\mu_{ij}]$, where rows correspond to objects and columns to classes. The *number of classes* of a partition, computed by function `n_of_classes()`, is the number of j for which $\mu_{ij} > 0$ for at least one object i. This may be less than the number of "available" classes, corresponding to the number of columns in a membership matrix representing the partition.

The predicate functions `is.cl_partition()`, `is.cl_hard_partition()`, and `is.cl_soft_partition()` are used to indicate whether R objects represent partitions of objects of the respective kind, with hard partitions as characterized above (all memberships in $\{0, 1\}$). (Hence, "fuzzy clustering" algorithms can in principle also give a hard partition.) `is.cl_partition()` and `is.cl_hard_partition()` are generic functions; `is.cl_soft_partition()` gives true iff `is.cl_partition()` is true and `is.cl_hard_partition()` is false.

For R objects representing partitions, function `cl_membership()` computes an R object with the membership values, currently always as a dense mem-

bership matrix with additional attributes. This is obviously rather inefficient for computations on hard partitions; we are planning to add "canned" sparse representations (using the vector of class ids) in future versions. Function `as.cl_membership()` can be used for coercing "raw" class ids (given as atomic vectors) or membership values (given as numeric matrices) to membership objects.

Function `cl_class_ids()` determines the class ids of a partition. For soft partitions, the class ids returned are those of the "nearest" hard partition obtained by taking the class ids of the (first) maximal membership values. Note that the cardinality of the set of the class ids may be less than the number of classes in the (soft) partition.

Many partitioning methods are based on *prototypes* ("centers"). In typical cases, these are points p_j in the same feature space the measurements x_i on the objects i to be partitioned are in, so that one can measure distance between objects and prototypes, and e.g. classify objects to their closest prototype. Such partitioning methods can also induce partitions of the entire feature space (rather than "just" the set of objects to be partitioned). Currently, package **clue** has only minimal support for this "additional" structure, providing a `cl_prototypes()` generic for extracting the prototypes, and is mostly focused on computations on partitions which are based on their memberships.

Many algorithms resulting in partitions of a given set of objects can be taken to induce a partition of the underlying feature space for the measurements on the objects, so that class memberships for "new" objects can be obtained from the induced partition. Examples include partitions based on assigning objects to their "closest" prototypes, or providing mixture models for the distribution of objects in feature space. Package **clue** provides a `cl_predict()` generic for predicting the class memberships of new objects (if possible).

Function `cl_fuzziness()` computes softness (fuzziness) measures for (ensembles) of partitions. Built-in measures are the partition coefficient and partition entropy (e.g., Bezdek, 1981), with an option to normalize in a way that hard partitions and the "fuzziest" possible partition (where all memberships are the same) get fuzziness values of zero and one, respectively. Note that this normalization differs from "standard" ones in the literature.

In the sequel, we shall also use the concept of the *co-membership matrix* $C(M) = MM'$, where $'$ denotes matrix transposition, of a partition. For hard partitions, an entry c_{ij} of $C(M)$ is 1 iff the corresponding objects i and j are in the same class, and 0 otherwise.

2.1.2 Hierarchies

The hierarchies considered in **clue** are *total indexed hierarchies*, also known as *n-valued trees*, and hence correspond in a one-to-one manner to *ultrametrics* (distances u_{ij} between pairs of objects i and j which satisfy the ultrametric constraint $u_{ij} = \max(u_{ik}, u_{jk})$ for all triples i, j, and k). See e.g. Gordon (1999, Page 69–71).

Function `cl_ultrametric(x)` computes the associated ultrametric from an

R object `x` representing a hierarchy of objects. If `x` is not an ultrametric, function `cophenetic()` in base package **stats** is used to obtain the ultrametric (also known as cophenetic) distances from the hierarchy, which in turn by default calls the S3 generic `as.hclust()` (also in **stats**) on the hierarchy. Support for classes which represent hierarchies can thus be added by providing `as.hclust()` methods for this class. In R 2.1.0 or better (again as part of the work on **clue**), `cophenetic` is an S3 generic as well, and one can also more directly provide methods for this if necessary.

In addition, there is a generic function `as.cl_ultrametric()` which can be used for coercing *raw* (non-classed) ultrametrics, represented as numeric vectors (of the lower-half entries) or numeric matrices, to ultrametric objects. Finally, the generic predicate function `is.cl_hierarchy()` is used to determine whether an R object represents a hierarchy or not.

Ultrametric objects can also be coerced to classes `"dendrogram"` and `"hclust"` (from base package **stats**), and hence in particular use the `plot()` methods for these classes. By default, plotting an ultrametric object uses the plot method for dendrograms.

Obtaining a hierarchy on a given set of objects can be thought of as transforming the pairwise dissimilarities between the objects (which typically do not yet satisfy the ultrametric constraints) into an ultrametric. Ideally, this ultrametric should be as close as possible to the dissimilarities. In some important cases, explicit solutions are possible (e.g., "standard" hierarchical clustering with single or complete linkage gives the optimal ultrametric dominated by or dominating the dissimilarities, respectively). On the other hand, the problem of finding the closest ultrametric in the least squares sense is known to be NP-hard (Krivanek and Moravek, 1986; Krivanek, 1986). One important class of heuristics for finding least squares fits is based on iterative projection on convex sets of constraints (Hubert and Arabie, 1995).

Function `ls_fit_ultrametric()` follows de Soete (1986) to use an SUMT (Sequential Unconstrained Minimization Technique; Fiacco and McCormick, 1968) approach in turn simplifying the suggestions in Carroll and Pruzansky (1980). Let $L(u)$ be the function to be minimized over all u in some constrained set $\mathcal{U}$—in our case, $L(u) = \sum(d_{ij} - u_{ij})^2$ is the least squares criterion, and $\mathcal{U}$ is the set of all ultrametrics u. One iteratively minimizes $L(u) + \rho_k P(u)$, where $P(u)$ is a non-negative function penalizing violations of the constraints such that $P(u)$ is zero iff $u \in \mathcal{U}$. The ρ values are increased according to the rule $\rho_{k+1} = q\rho_k$ for some constant $q > 1$, until convergence is obtained in the sense that e.g. the Euclidean distance between successive solutions u_k and u_{k+1} is small enough. Optionally, the final u_k is then suitably projected onto $\mathcal{U}$.

For `ls_fit_ultrametric()`, we obtain the starting value u_0 by "random shaking" of the given dissimilarity object, and use the penalty function $P(u) = \sum_\Omega (u_{ij} - u_{jk})^2$, were Ω contains all triples i, j, k for which $u_{ij} \le \min(u_{ik}, u_{jk})$ and $u_{ik} \neq u_{jk}$, i.e., for which u violates the ultrametric constraints. The unconstrained minimizations are carried out using either `optim()` or `nlm()` in base package **stats**, with analytic gradients given in Carroll and Pruzansky (1980). This "works", even though we note however that P is not even a continuous

function, which seems to have gone unnoticed in the literature! (Consider an ultrametric u for which $u_{ij} = u_{ik} < u_{jk}$ for some i, j, k and define $u(\delta)$ by changing the u_{ij} to $u_{ij} + \delta$. For u, both (i, j, k) and (j, i, k) are in the violation set Ω, whereas for all δ sufficiently small, only (j, i, k) is the violation set for $u(\delta)$. Hence, $\lim_{\delta \to 0} P(u(\delta)) = P(u) + (u_{ij} - u_{ik})^2$. This shows that P is discontinuous at all non-constant u with duplicated entries. On the other hand, it is continuously differentiable at all u with unique entries.) Hence, we need to turn off checking analytical gradients when using `nlm()` for minimization.

The default optimization using conjugate gradients should work reasonably well for medium to large size problems. For "small" ones, using `nlm()` is usually faster. Note that the number of ultrametric constraints is of the order n^3, suggesting to use the SUMT approach in favor of `constrOptim()` in **stats**. It should be noted that the SUMT approach is a heuristic which can not be guaranteed to find the global minimum. Standard practice would recommend to use the best solution found in "sufficiently many" replications of the base algorithm.

2.1.3 Extensibility

The methods provided in package **clue** handle the partitions and hierarchies obtained from clustering functions in the base R distribution, as well as packages **RWeka** (Hornik, Hothorn, and Karatzoglou, 2006), **cba** (Buchta and Hahsler, 2005), **cclust** (Dimitriadou, 2005), **cluster**, **e1071** (Dimitriadou, Hornik, Leisch, Meyer, and Weingessel, 2005), **flexclust** (Leisch, 2006), **flexmix** (Leisch, 2004), **kernlab** (Karatzoglou, Smola, Hornik, and Zeileis, 2004), and **mclust** (and of course, **clue** itself).

Extending support to other packages is straightforward, provided that clusterings are instances of classes. Suppose e.g. that a package has a function `glvq()` for "generalized" (i.e., non-Euclidean) Learning Vector Quantization which returns an object of class `"glvq"`, in turn being a list with component `class_ids` containing the class ids. To integrate this into the **clue** framework, all that is necessary is to provide the following methods.

```
> cl_class_ids.glvq <- function(x) as.cl_class_ids(x$class_ids)
> is.cl_partition.glvq <- function(x) TRUE
> is.cl_hard_partition.glvq <- function(x) TRUE
```

2.2 Cluster ensembles

Cluster ensembles are realized as lists of clusterings with additional class information. All clusterings in an ensemble must be of the same "kind" (i.e., either all partitions as known to `is.cl_partition()`, or all hierarchies as known to `is.cl_hierarchy()`, respectively), and have the same number of objects. If all clusterings are partitions, the list realizing the ensemble has class `"cl_partition_ensemble"` and inherits from `"cl_ensemble"`; if all clusterings are hierarchies, it has class `"cl_hierarchy_ensemble"` and inherits from `"cl_ensemble"`. Empty ensembles cannot be categorized according to the kind of clusterings they contain, and hence only have class `"cl_ensemble"`.

Function `cl_ensemble()` creates a cluster ensemble object from clusterings given either one-by-one, or as a list passed to the `list` argument. As unclassed lists could be used to represent single clusterings (in particular for results from `kmeans()` in versions of R prior to 2.1.0), we prefer not to assume that an unnamed given list is a list of clusterings. `cl_ensemble()` verifies that all given clusterings are of the same kind, and all have the same number of objects. (By the notion of cluster ensembles, we should in principle verify that the clusterings come from the *same* objects, which of course is not always possible.)

The list representation makes it possible to use `lapply()` for computations on the individual clusterings in (i.e., the components of) a cluster ensemble.

Available methods for cluster ensembles include those for subscripting, `c()`, `rep()`, `print()`, and `unique()`, where the last is based on a `unique()` method for lists added in R 2.1.1, and makes it possible to find unique and duplicated elements in cluster ensembles. The elements of the ensemble can be tabulated using `cl_tabulate()`.

Function `cl_boot()` generates cluster ensembles with bootstrap replicates of the results of applying a "base" clustering algorithm to a given data set. Currently, this is a rather simple-minded function with limited applicability, and mostly useful for studying the effect of (uncontrolled) random initializations of fixed-point partitioning algorithms such as `kmeans()` or `cmeans()` in package **e1071**. To study the effect of varying control parameters or explicitly providing random starting values, the respective cluster ensemble has to be generated explicitly (most conveniently by using `replicate()` to create a list `lst` of suitable instances of clusterings obtained by the base algorithm, and using `cl_ensemble(list = lst)` to create the ensemble). Resampling the training data is possible for base algorithms which can predict the class memberships of new data using `cl_predict` (e.g., by classifying the out-of-bag data to their closest prototype). In fact, we believe that for unsupervised learning methods such as clustering, *reweighting* is conceptually superior to resampling, and have therefore recently enhanced package **e1071** to provide an implementation of weighted fuzzy c-means, and package **flexclust** contains an implementation of weighted k-means. We are currently experimenting with interfaces for providing "direct" support for reweighting via `cl_boot()`.

2.3 Cluster proximities

2.3.1 Principles

Computing dissimilarities and similarities ("agreements") between clusterings of the same objects is a key ingredient in the analysis of cluster ensembles. The "standard" data structures available for such proximity data (measures of similarity or dissimilarity) are classes `"dist"` and `"dissimilarity"` in package **cluster** (which basically, but not strictly, extends `"dist"`), and are both not entirely suited to our needs. First, they are confined to *symmetric* dissimilarity data. Second, they do not provide enough reflectance. We also note that the Bioconductor package **graph** (Gentleman and Whalen, 2005) contains an effi-

cient subscript method for objects of class `"dist"`, but returns a "raw" matrix for row/column subscripting.

For package **clue**, we use the following approach. There are classes for symmetric and (possibly) non-symmetric proximity data (`"cl_proximity"` and `"cl_cross_proximity"`), which, in addition to holding the numeric data, also contain a description "slot" (attribute), currently a character string, as a first approximation to providing more reflectance. Internally, symmetric proximity data are store the lower diagonal proximity values in a numeric vector (in row-major order), i.e., the same way as objects of class `"dist"`; a `self` attribute can be used for diagonal values (in case some of these are non-zero). Symmetric proximity objects can be coerced to dense matrices using `as.matrix()`. It is possible to use 2-index matrix-style subscripting for symmetric proximity objects; unless this uses identical row and column indices, it results in a non-symmetric proximity object.

This approach "propagates" to classes for symmetric and (possibly) non-symmetric cluster dissimilarity and agreement data (e.g., `"cl_dissimilarity"` and `"cl_cross_dissimilarity"` for dissimilarity data), which extend the respective proximity classes.

Ultrametric objects are implemented as symmetric proximity objects with a dissimilarity interpretation so that self-proximities are zero, and inherit from classes `"cl_dissimilarity"` and `"cl_proximity"`.

Providing reflectance is far from optimal. For example, if `s` is a similarity object (with cluster agreements), `1 - s` is a dissimilarity one, but the description is preserved unchanged. This issue could be addressed by providing high-level functions for transforming proximities.

Cluster dissimilarities are computed via `cl_dissimilarity()` with synopsis `cl_dissimilarity(x, y = NULL, method = "euclidean")`, where `x` and `y` are cluster ensemble objects or coercible to such, or `NULL` (`y` only). If `y` is `NULL`, the return value is an object of class `"cl_dissimilarity"` which contains the dissimilarities between all pairs of clusterings in `x`. Otherwise, it is an object of class `"cl_cross_dissimilarity"` with the dissimilarities between the clusterings in `x` and the clusterings in `y`. Formal argument `method` is either a character string specifying one of the built-in methods for computing dissimilarity, or a function to be taken as a user-defined method, making it reasonably straightforward to add methods.

Function `cl_agreement()` has the same interface as `cl_dissimilarity()`, returning cluster similarity objects with respective classes `"cl_agreement"` and `"cl_cross_agreement"`. Built-in methods for computing dissimilarities may coincide (in which case they are transforms of each other), but do not necessarily do so, as there typically are no canonical transformations. E.g., according to needs and scientific community, agreements might be transformed to dissimilarities via $d = -\log(s)$ or the square root thereof (e.g., Strehl and Ghosh, 2003b), or via $d = 1 - s$.

2.3.2 Partition proximities

When assessing agreement or dissimilarity of partitions, one needs to consider that the class ids may be permuted arbitrarily without changing the underlying partitions. For membership matrices M, permuting class ids amounts to replacing M by $M\Pi$, where Π is a suitable permutation matrix. We note that the co-membership matrix $C(M) = MM'$ is unchanged by these transformations; hence, proximity measures based on co-occurences, such as the Katz-Powell (Katz and Powell, 1953) or Rand (Rand, 1971) indices, do not explicitly need to adjust for possible re-labeling. The same is true for measures based on the "confusion matrix" $M'\tilde{M}$ of two membership matrices M and $\tilde{M}$ which are invariant under permuations of rows and columns, such as the Normalized Mutual Information (NMI) measure introduced in Strehl and Ghosh (2003a). Other proximity measures need to find permutations so that the classes are optimally matched, which of course in general requires exhaustive search through all $k!$ possible permutations, where k is the (common) number of classes in the partitions, and thus will typically be prohibitively expensive. Fortunately, in some important cases, optimal matchings can be determined very efficiently. We explain this in detail for "Euclidean" partition dissimilarity and agreement (which in fact is the default measure used by `cl_dissimilarity()` and `cl_agreement()`).

Euclidean partition dissimilarity (Dimitriadou, Weingessel, and Hornik, 2002) is defined as

$$d(M, \tilde{M}) = \min_\Pi \|M - \tilde{M}\Pi\|$$

where the minimum is taken over all permutation matrices Π, $\|\cdot\|$ is the Frobenius norm (so that $\|Y\|^2 = \mathrm{tr}(Y'Y)$), and n is the (common) number of objects in the partitions. As $\|M - \tilde{M}\Pi\|^2 = \mathrm{tr}(M'M) - 2\,\mathrm{tr}(M'\tilde{M}\Pi) + \mathrm{tr}(\Pi'\tilde{M}'\tilde{M}\Pi) = \mathrm{tr}(M'M) - 2\,\mathrm{tr}(M'\tilde{M}\Pi) + \mathrm{tr}(\tilde{M}'\tilde{M})$, we see that minimizing $\|M - \tilde{M}\Pi\|^2$ is equivalent to maximizing $\mathrm{tr}(M'\tilde{M}\Pi) = \sum_{i,k} \mu_{ik}\tilde{\mu}_{i,\pi(k)}$, which for hard partitions is the number of objects with the same label in the partitions given by M and $\tilde{M}\Pi$. Finding the optimal Π is thus recognized as an instance of the *linear sum assignment problem* (LSAP, also known as the weighted bipartite graph matching problem). The LSAP can be solved by linear programming, e.g., using Simplex-style primal algorithms as done by function `lp.assign()` in package **lpSolve** (Buttrey, 2005), but primal-dual algorithms such as the so-called Hungarian method can be shown to find the optimum in time $O(k^3)$ (e.g., Papadimitriou and Steiglitz, 1982). Available published implementations include TOMS 548 (Carpaneto and Toth, 1980), which however is restricted to integer weights and $k < 131$. One can also transform the LSAP into a network flow problem, and use e.g. RELAX-IV (Bertsekas and Tseng, 1994) for solving this, as is done in package **optmatch** (Hansen, 2005). In package **clue**, we use an efficient C implementation of the Hungarian algorithm kindly provided to us by Walter Böhm, which has been found to perform very well across a wide range of problem sizes.

Gordon and Vichi (2001) use a variant of Euclidean dissimilarity ("GV1

dissimilarity") which is based on the sum of the squared difference of the memberships of matched (non-empty) classes only, discarding the unmatched ones (see their Example 2). This results in a measure which is discontinuous over the space of soft partitions with arbitrary numbers of classes.

The partition agreement measures "angle" and "diag" (maximal cosine of angle between the memberships, and maximal co-classification rate, where both maxima are taken over all column permutations of the membership matrices) are based on solving the same LSAP as for Euclidean dissimilarity.

Finally, Manhattan partition dissimilarity is defined as the minimal sum of the absolute differences of M and all column permutations of $\tilde{M}$, and can again be computed efficiently by solving an LSAP.

For hard partitions, both Manhattan and squared Euclidean dissimilarity give twice the *transfer distance* (Charon, Denoeud, Guénoche, and Hudry, 2005), which is the minimum number of objects that must be removed so that the implied partitions (restrictions to the remaining objects) are identical. This is also known as the *R-metric* in Day (1981), i.e., the number of augmentations and removals of single objects needed to transform one partition into the other, and the *partition-distance* in Gusfield (2002).

Note when assessing proximity that agreements for soft partitions are always (and quite often considerably) lower than the agreements for the corresponding nearest hard partitions, unless the agreement measures are based on the latter anyways (as currently done for Rand, Katz-Powell, and NMI).

Package **clue** provides additional agreement measures, such as the Jaccard and Fowles-Mallows (Fowlkes and Mallows, 1983, quite often incorrectly attributed to Wallace (1983)) indices, and dissimilarity measures such as the "symdiff" and Rand distances (the latter is proportional to the metric of Mirkin (1996)) and the metrics discussed in Boorman and Arabie (1972). One could easily add more proximity measures, such as the "Variation of Information" (Meila, 2003). However, all these measures are rigorously defined for hard partitions only. To see why extensions to soft partitions are far from straightforward, consider e.g. measures based on the confusion matrix. Its entries count the cardinality of certain intersections of sets. In a fuzzy context for soft partitions, a natural generalization would be using fuzzy cardinalities (i.e., sums of memberships values) of fuzzy intersections instead. There are many possible choices for the latter, with the product of the membership values (corresponding to employing the confusion matrix also in the fuzzy case) one of them, but the minimum instead of the product being the "usual" choice. A similar point can be made for co-occurrences of soft memberships. We are not aware of systematic investigations of these extension issues.

2.3.3 Hierachy proximities

Available built-in dissimilarity measures for hierarchies include *Euclidean* (again, the default measure used by `cl_dissimilarity()`) and Manhattan dissimilarity, which are simply the Euclidean (square root of the sum of squared differences) and Manhattan (sum of the absolute differences) dissimilarities be-

tween the associated ultrametrics. Cophenetic dissimilarity is defined as $1 - c^2$, where c is the cophenetic correlation coefficient (Sokal and Rohlf, 1962), i.e., the Pearson product-moment correlation between the ultrametrics. Gamma dissimilarity is the rate of inversions between the associated ultrametrics u and v (i.e., the rate of pairs (i, j) and (k, l) for which $u_{ij} < u_{kl}$ and $v_{ij} > v_{kl}$). This measure is a linear transformation of Kruskal's γ. Finally, symdiff dissimilarity is the cardinality of the symmetric set difference of the sets of classes (hierarchies in the strict sense) induced by the dendrograms.

Associated agreement measures are obtained by suitable transformations of the dissimilarities d; for Euclidean proximities, we prefer to use $1/(1 + d)$ rather than e.g. $\exp(-d)$.

One should note that whereas cophenetic and gamma dissimilarities are invariant to linear transformations, Euclidean and Manhattan ones are not. Hence, if only the relative "structure" of the dendrograms is of interest, these dissimilarities should only be used after transforming the ultrametrics to a common range of values (e.g., to $[0, 1]$).

2.4 Consensus clusterings

Consensus clusterings "synthesize" the information in the elements of a cluster ensemble into a single clustering. There are three main approaches to obtaining consensus clusterings (Hornik, 2005a; Gordon and Vichi, 2001): in the *constructive* approach, one specifies a way to construct a consensus clustering. In the *axiomatic* approach, emphasis is on the investigation of existence and uniqueness of consensus clusterings characterized axiomatically. The *optimization* approach formalizes the natural idea of describing consensus clusterings as the ones which "optimally represent the ensemble" by providing a criterion to be optimized over a suitable set $\mathcal{C}$ of possible consensus clusterings. If d is a dissimilarity measure and $C_1, \ldots, C_B$ are the elements of the ensemble, one can e.g. look for solutions of the problem

$$\sum_{b=1}^{B} w_b d(C, C_b)^p \Rightarrow \min_{C \in \mathcal{C}},$$

for some $p \geq 0$, i.e., as clusterings C^* minimizing weighted average dissimilarity powers of order p. Analogously, if a similarity measure is given, one can look for clusterings maximizing weighted average similarity powers. Following Gordon and Vichi (1998), an above C^* is referred to as (weighted) *median* or *medoid* clustering if $p = 1$ and the optimum is sought over the set of all possible base clusterings, or the set $\{C_1, \ldots, C_B\}$ of the base clusterings, respectively. For $p = 2$, we have *least squares* consensus clusterings (generalized means).

For computing consensus clusterings, package **clue** provides function `cl_consensus()` with synopsis `cl_consensus(x, method = NULL, weights = 1, control = list())`. This allows (similar to the functions for computing cluster proximities, see Section 2.3.1 on Page 7) argument `method` to be a character string specifying one of the built-in methods discussed below, or a function to be taken as a user-defined method (taking an ensemble, the case weights,

and a list of control parameters as its arguments), again making it reasonably straightforward to add methods. In addition, function `cl_medoid()` can be used for obtaining medoid partitions (using, in principle, arbitrary dissimilarities). Modulo possible differences in the case of ties, this gives the same results as (the medoid obtained by) `pam()` in package **cluster**.

If all elements of the ensemble are partitions, package **clue** provides algorithms for computing soft least squares consensus partitions for weighted Euclidean, GV1 and co-membership dissimilarities. Let $M_1, \ldots, M_B$ and M denote the membership matrices of the elements of the ensemble and their sought least squares consensus partition, respectively. For Euclidean dissimilarity, we need to find

$$\sum_b w_b \min_{\Pi_b} \|M - M_b\Pi_b\|^2 \Rightarrow \min_M$$

over all membership matrices (i.e., stochastic matrices) M, or equivalently,

$$\sum_b w_b \|M - M_b\Pi_b\|^2 \Rightarrow \min_{M,\Pi_1,\ldots,\Pi_B}$$

over all M and permutation matrices $\Pi_1, \ldots, \Pi_B$. Now fix the Π_b and let $\bar{M} = s^{-1} \sum_b w_b M_b \Pi_b$ be the weighted average of the $M_b\Pi_b$, where $s = \sum_b w_b$. Then

$$\begin{aligned}
&\sum_b w_b \|M - M_b\Pi_b\|^2 \\
&\quad = \sum_b w_b (\|M\|^2 - 2\,\mathrm{tr}(M' M_b \Pi_b) + \|M_b\Pi_b\|^2) \\
&\quad = s\|M\|^2 - 2s\,\mathrm{tr}(M'\bar{M}) + \sum_b w_b \|M_b\|^2 \\
&\quad = s(\|M - \bar{M}\|^2) + \sum_b w_b \|M_b\|^2 - s\|\bar{M}\|^2
\end{aligned}$$

Thus, as already observed in Dimitriadou et al. (2002) and Gordon and Vichi (2001), for fixed permutations Π_b the optimal soft M is given by $\bar{M}$. The optimal permutations can be found by minimizing $-s\|\bar{M}\|^2$, or equivalently, by maximizing

$$s^2\|\bar{M}\|^2 = \sum_{\beta,b} w_\beta w_b \,\mathrm{tr}(\Pi'_\beta M'_\beta M_b \Pi_b).$$

With $U_{\beta,b} = w_\beta w_b M'_\beta M_b$ we can rewrite the above as

$$\sum_{\beta,b} w_\beta w_b \,\mathrm{tr}(\Pi'_\beta M'_\beta M_b \Pi_b) = \sum_{\beta,b} \sum_{j=1}^{k} [U_{\beta,b}]_{\pi_\beta(j),\pi_b(j)} =: \sum_{j=1}^{k} c_{\pi_1(j),\ldots,\pi_B(j)}$$

This is an instance of the *multi-dimensional assignment problem* (MAP), which, contrary to the LSAP, is known to be NP-hard (e.g., via reduction to 3-DIMENSIONAL MATCHING, Garey and Johnson, 1979), and can e.g.

be approached using randomized parallel algorithms (Oliveira and Pardalos, 2004). Branch-and-bound approaches suggested in the literature (e.g., Grundel, Oliveira, Pardalos, and Pasiliao, 2005) are unfortunately computationally infeasible for "typical" sizes of cluster ensembles ($B \geq 20$, maybe even in the hundreds).

Package **clue** provides two heuristics for (approximately) finding the soft least squares consensus partition for Euclidean dissimilarity. Method `"DWH"` of function `cl_consensus()` is an extension of the greedy algorithm in Dimitriadou et al. (2002) which is based on a single forward pass through the ensemble which in each step chooses the "locally" optimal Π. Starting with $\tilde{M}_1 = M_1$, $\tilde{M}_b$ is obtained from $\tilde{M}_{b-1}$ by optimally matching $M_b\Pi_b$ to this, and taking a weighted average of $\tilde{M}_{b-1}$ and $M_b\Pi_b$ in a way that $\tilde{M}_b$ is the weighted average of the first b $M_\beta\Pi_\beta$. This simple approach could be further enhanced via backfitting or several passes, in essence resulting in an "on-line" version of method `"SE"`. This, in turn, is a fixed-point algorithm, which iterates between updating M as the weighted average of the current $M_b\Pi_b$, and determining the Π_b by optimally matching the current M to the individual M_b. Finally, method `"GV1"` implements the fixed-point algorithm for the "first model" in Gordon and Vichi (2001), which gives least squares consensus partitions for GV1 dissimilarity.

In the above, we implicitly assumed that all partitions in the ensemble as well as the sought consensus partition have the same number of classes. The more general case can be dealt with through suitable "projection" devices.

When using co-membership dissimilarity, the least squares consensus partition is determined by minimizing

$$\begin{aligned}
&\sum_b w_b \|MM' - M_b M_b'\|^2 \\
&\quad = \quad s\|MM' - \bar{C}\|^2 + \sum_b w_b \|M_b M_b'\|^2 - s\|\bar{C}\|^2
\end{aligned}$$

over all membership matrices M, where now $\bar{C} = s^{-1}\sum_b C(M_b) = s^{-1}\sum_b M_b M_b'$ is the weighted average co-membership matrix of the ensemble. This corresponds to the "third model" in Gordon and Vichi (2001). Method `"GV3"` of function `cl_consensus()` provides a SUMT approach (see Section 2.1.2 on Page 4) for finding the minimum. We note that this strategy could more generally be applied to consensus problems of the form

$$\sum_b w_b \|\Phi(M) - \Phi(M_b)\|^2 \Rightarrow \min_M,$$

which are equivalent to minimizing $\|\Phi(B) - \bar{\Phi}\|^2$, with $\bar{\Phi}$ the weighted average of the $\Phi(M_b)$. This includes e.g. the case where generalized co-memberships are defined by taking the "standard" fuzzy intersection of co-incidences, as discussed in Section 2.3.2 on Page 9.

Package **clue** currently does not provide algorithms for obtaining *hard* consensus partitions, as e.g. done in Krieger and Green (1999) using Rand proximity. It seems "natural" to extend the methods discussed above to include a constraint

on softness, e.g., on the partition coefficient PC (see Section 2.1.1 on Page 3). For Euclidean dissimilarity, straightforward Lagrangian computations show that the constrained minima are of the form $\bar{M}(\alpha) = \alpha\bar{M} + (1-\alpha)E$, where E is the "maximally soft" membership with all entries equal to $1/k$, $\bar{M}$ is again the weighted average of the $M_b\Pi_b$ with the Π_b solving the underlying MAP, and α is chosen such that $PC(\bar{M}(\alpha))$ equals a prescribed value. As α increases (even beyond one), softness of the $\bar{M}(\alpha)$ decreases. However, for $\alpha^* > 1/(1-k\mu^*)$, where μ^* is the minimum of the entries of $\bar{M}$, the $\bar{M}(\alpha)$ have negative entries, and are no longer feasible membership matrices. Obviously, the non-negativity constraints for the $\bar{M}(\alpha)$ eventually put restrictions on the admissible Π_b in the underlying MAP. Thus, such a simple relaxation approach to obtaining optimal hard partitions is not feasible.

For ensembles of hierarchies, `cl_consensus()` provides a built-in method (`"cophenetic"`) for approximately minimizing average weighted squared Euclidean dissimilarity

$$\sum_b w_b \|U - U_b\|^2 \Rightarrow \min_U$$

over all ultrametrics U, where $U_1, \ldots, U_B$ are the ultrametrics corresponding to the elements of the ensemble. This is of course equivalent to minimizing $\|U - \bar{U}\|^2$, where $\bar{U} = s^{-1}\sum_b w_b U_b$ is the weighted average of the U_b. The SUMT approach provided by function `ls_fit_ultrametric()` (see Section 2.1.2 on Page 4) is employed for finding the sought weighted least squares consensus hierarchy.

In addition, method `"majority"` obtains a consensus hierarchy from an extension of the majority consensus tree of Margush and McMorris (1981), which minimizes $L(U) = \sum_b w_b d(U_b, U)$ over all ultrametrics U, where d is the symmetric difference dissimilarity.

Clearly, the available methods use heuristics for solving hard optimization problems, and cannot be guaranteed to find a global optimum. Standard practice would recommend to use the best solution found in "sufficiently many" replications of the methods.

Alternative recent approaches to obtaining consensus partitions include "Bagged Clustering" (Leisch, 1999, provided by `bclust()` in package **e1071**), the "evidence accumulation" framework of Fred and Jain (2002), the NMI optimization and graph-partitioning methods in Strehl and Ghosh (2003a), "Bagged Clustering" as in Dudoit and Fridlyand (2003), and the hybrid bipartite graph formulation of Fern and Brodley (2004). Typically, these approaches are constructive, and can easily be implemented based on the infrastructure provided by package **clue**. Evidence accumulation amounts to standard hierarchical clustering of the average co-membership matrix. Procedure BagClust1 of Dudoit and Fridlyand (2003) amounts to computing $B^{-1}\sum_b M_b\Pi_b$, where each Π_b is determined by optimal Euclidean matching of M_b to a fixed reference membership M_0. In the corresponding "Bagged Clustering" framework, M_0 and the M_b are obtained by applying the base clusterer to the original data set and bootstrap samples from it, respectively. This is implemented as method `"DFBC1"` of

`cl_bag()` in package **clue**. Finally, the approach of Fern and Brodley (2004) solves an LSAP for an asymmetric cost matrix based on object-by-all-classes incidences.

2.5 Cluster partitions

To investigate the "structure" in a cluster ensemble, an obvious idea is to start clustering the clusterings in the ensemble, resulting in "secondary" clusterings (Gordon and Vichi, 1998; Gordon, 1999). This can e.g. be performed by using `cl_dissimilarity()` (or `cl_agreement()`) to compute a dissimilarity matrix for the ensemble, and feed this into a dissimilarity-based clustering algorithm (such as `pam()` in package **cluster** or `hclust()` in package **stats**). (One can even use `cutree()` to obtain hard partitions from hierarchies thus obtained.) If prototypes ("typical clusterings") are desired for partitions of clusterings, they can be determined post-hoc by finding suitable consensus clusterings in the classes of the partition, e.g., using `cl_consensus()` or `cl_medoid()`.

Package **clue** additionally provides `cl_pclust()` for direct prototype-based partitioning based on minimizing criterion functions of the form $\sum u_{bj}^m d(x_b, p_j)^e$, the sum of the membership-weighted e-th powers of the dissimilarities between the elements x_b of the ensemble and the prototypes p_j, for suitable dissimilarities d and exponents e. (The underlying feature spaces are that of membership matrices and ultrametrics, respectively, for partitions and hierarchies.)

Parameter m must not be less than one and controls the softness of the obtained partitions, corresponding to the "fuzzification parameter" of the fuzzy c-means algorithm. For $m = 1$, a generalization of the Lloyd-Forgy variant (Lloyd, 1957; Forgy, 1965; Lloyd, 1982) of the k-means algorithm is used, which iterates between reclassifying objects to their closest prototypes, and computing new prototypes as consensus clusterings for the classes. This may result in degenerate solutions, and will be replaced by a Hartigan-Wong (Hartigan and Wong, 1979) style algorithm eventually. For $m > 1$, a generalization of the fuzzy c-means recipe (e.g., Bezdek, 1981) is used, which alternates between computing optimal memberships for fixed prototypes, and computing new prototypes as the consensus clusterings for the classes.

This procedure is repeated until convergence occurs, or the maximal number of iterations is reached. Consensus clusterings are computed using (one of the methods provided by) `cl_consensus`, with dissimilarities d and exponent e implied by method employed, and obtained via a registration mechanism. The default methods compute Least Squares Euclidean consensus clusterings, i.e., use Euclidean dissimilarity d and $e = 2$.

3 Examples

3.1 Cassini data

Dimitriadou et al. (2002) and Leisch (1999) use Cassini data sets to illustrate

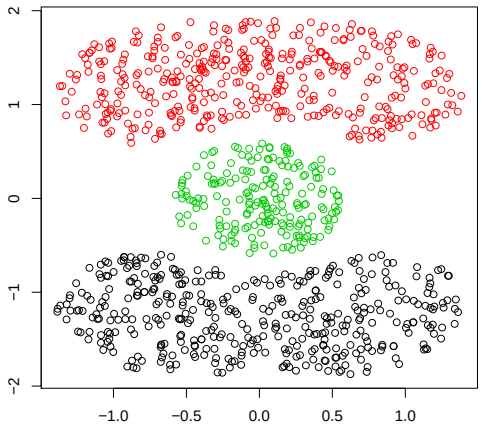

Figure 1: The Cassini data set.

how e.g. suitable aggregation of base k-means results can reveal underlying non-convex structure which cannot be found by the base algorithm. Such data sets contain points in 2-dimensional space drawn from the uniform distribution on 3 structures, with the two "outer" ones banana-shaped and the "middle" one a circle, and can be obtained by function `mlbench.cassini()` in package **mlbench** (Leisch and Dimitriadou, 2005). Package **clue** contains the data sets `Cassini` and `CKME`, which are an instance of a 1000-point Cassini data set, and a cluster ensemble of 50 k-means partitions of the data set into three classes, respectively.

The data set is shown in Figure 1.

```
> data("Cassini")
> plot(Cassini$x, col = as.integer(Cassini$classes),
+     xlab = "", ylab = "")
```

Figure 2 gives a dendrogram of the Euclidean dissimilarities of the elements of the k-means ensemble.

```
> data("CKME")
> plot(hclust(cl_dissimilarity(CKME)), labels = FALSE)
```

We can see that there are large groups of essentially identical k-means solutions. We can gain more insight by inspecting representatives of these three groups, or by computing the medoid of the ensemble

```
> m1 <- cl_medoid(CKME)
> table(Medoid = cl_class_ids(m1), "True Classes" = Cassini$classes)
```

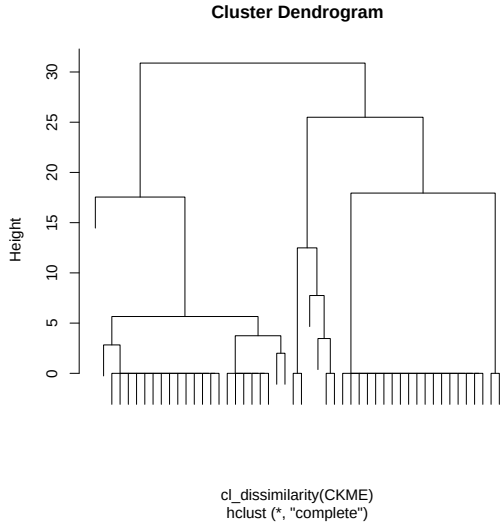

Figure 2: A dendrogram of the Euclidean dissimilarities of 50 k-means partitions of the Cassini data into 3 classes.

```
      True Classes
Medoid   1   2   3
     1 196   0  89
     2 204   0  30
     3   0 400  81
```

and inspecting it (Figure 3):

```
> plot(Cassini$x, col = cl_class_ids(m1), xlab = "",
+     ylab = "")
```

Flipping this solution top-down gives a second "typical" partition. We see that the k-means base clusterers cannot resolve the underlying non-convex structure. For the least squares consensus of the ensemble, we obtain

```
> set.seed(1234)
> m2 <- cl_consensus(CKME)
```

where here and below we set the random seed for reproducibility, noting that one should really use several replicates of the consensus heuristic. This consensus partition has confusion matrix

```
> table(Consensus = cl_class_ids(m2), "True Classes" = Cassini$classes)
```

```
         True Classes
Consensus   1   2   3
```

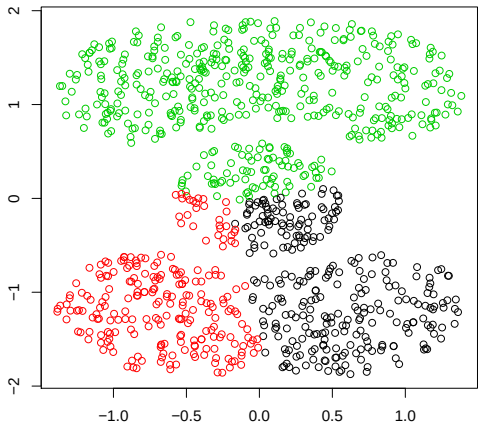

Figure 3: Medoid of the Cassini k-means ensemble.

```
1   0 372  30
2 388   0  30
3  12  28 140
```

and class details as displayed in Figure 4:

```
> plot(Cassini$x, col = cl_class_ids(m2), xlab = "",
+     ylab = "")
```

This has drastically improved performance, and almost perfect recovery of the two outer shapes. In fact, Dimitriadou et al. (2002) show that almost perfect classification can be obtained by suitable combinations of different base clusterers (k-means, fuzzy c-means, and unsupervised fuzzy competitive learning).

3.2 Gordon-Vichi macroeconomic data

Gordon and Vichi (2001, Table 1) provide soft partitions of 21 countries based on macroeconomic data for the years 1975, 1980, 1985, 1990, and 1995. These partitions were obtained using fuzzy c-means on measurements of the following variables: the annual per capita gross domestic product (GDP) in USD (converted to 1987 prices); the percentage of GDP provided by agriculture; the percentage of employees who worked in agriculture; and gross domestic investment, expressed as a percentage of the GDP.

Table 5 in Gordon and Vichi (2001) gives 3-class consensus partitions obtained by applying their models 1, 2, and 3 and the approach in Sato and Sato (1994).

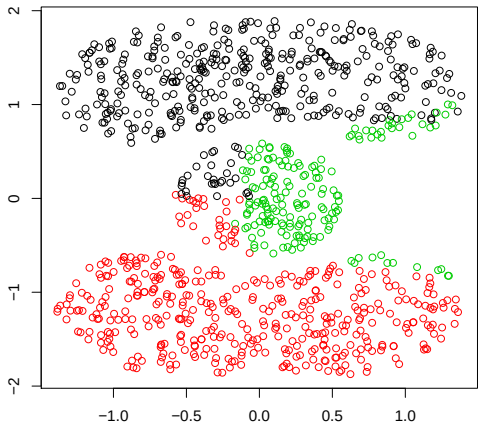

Figure 4: Least Squares Consensus of the Cassini k-means ensemble.

The partitions and consensus partitions are available in data sets `GVME` and `GVME_Consensus`, respectively. We compare the results of Gordon and Vichi (2001) using GV1 dissimilarities (model 1) to ours as obtained by `cl_consensus()` with method `"GV1"`.

```
> data("GVME")
> GVME
```

```
An ensemble of 5 partitions of 21 objects.
```

```
> set.seed(1)
> m1 <- cl_consensus(GVME, method = "GV1", control = list(k = 3,
+     verbose = TRUE))
```

```
Loading required package: quadprog
Iteration: 1 Old value: 7.263485 New value: 1.888366
Iteration: 2 Old value: 1.888366 New value: 1.575884
Iteration: 3 Old value: 1.575884 New value: 1.575884
```

This results in a soft partition with average squared GV1 dissimilarity (the criterion function to be optimized by the consensus partition) of

```
> mean(cl_dissimilarity(GVME, m1, "GV1")^2)
```

```
[1] 1.575884
```

We compare this to the consensus solution given in Gordon and Vichi (2001):

```
> data("GVME_Consensus")
> m2 <- GVME_Consensus[["MF1/3"]]
> mean(cl_dissimilarity(GVME, m2, "GV1")^2)

[1] 1.667204

> table(CLUE = cl_class_ids(m1), GV2001 = cl_class_ids(m2))

    GV2001
CLUE 1 2 3
   1 0 7 0
   2 0 0 5
   3 9 0 0
```

Interestingly, we are able to obtain a "better" solution, which however agrees with the one reported on the literature with respect to their nearest hard partitions.

For the 2-class consensus partition, we obtain

```
> set.seed(1)
> m1 <- cl_consensus(GVME, method = "GV1", control = list(k = 2,
+     verbose = TRUE))

Iteration: 1 Old value: 6.96343 New value: 2.370098
Iteration: 2 Old value: 2.370098 New value: 1.553614
Iteration: 3 Old value: 1.553614 New value: 1.553614
```

which is slightly better than the solution reported in Gordon and Vichi (2001)

```
> mean(cl_dissimilarity(GVME, m1, "GV1")^2)

[1] 1.553614

> m2 <- GVME_Consensus[["MF1/2"]]
> mean(cl_dissimilarity(GVME, m2, "GV1")^2)

[1] 1.55363
```

but in fact agrees with it apart from rounding errors:

```
> max(abs(cl_membership(m1) - cl_membership(m2)))

[1] 0.001
```

It is interesting to compare these solutions to the Euclidean 2-class consensus partition for the GVME ensemble:

```
> m3 <- cl_consensus(GVME, method = "GV1", control = list(k = 2,
+     verbose = TRUE))
```

```
Iteration: 1 Old value: 5.914507 New value: 1.889004
Iteration: 2 Old value: 1.889004 New value: 1.889004
```

This is markedly different from the GV1 consensus partition

```
> table(GV1 = cl_class_ids(m1), Euclidean = cl_class_ids(m3))
```

```
   Euclidean
GV1 1 2
  1 9 5
  2 0 7
```

with countries

```
> rownames(m1)[cl_class_ids(m1) != cl_class_ids(m3)]
```

```
NULL
```

classified differently, being with the "richer" class for the GV1 and the "poorer" for the Euclidean consensus partition. (In fact, all these countries end up in the "middle" class for the 3-class GV1 consensus partition.)

3.3 Rosenberg-Kim kinship terms data

Rosenberg and Kim (1975) describe an experiment where perceived similarities of the kinship terms were obtained from six different "sorting" experiments. In one of these, 85 female undergraduates at Rutgers University were asked to sort 15 English terms into classes "on the basis of some aspect of meaning". These partitions were printed in Rosenberg (1982, Table 7.1). Comparison with the original data indicates that the partition data have the "nephew" and "niece" columns interchanged, which is corrected in data set `Kinship82`.

Gordon and Vichi (2001, Table 6) provide consensus partitions for these data based on their models 1–3 (available in data set `Kinship82_Consensus`). We compare their results using co-membership dissimilarities (model 3) to ours as obtained by `cl_consensus()` with method `"GV3"`.

```
> data("Kinship82")
> Kinship82
```

```
An ensemble of 85 partitions of 15 objects.
```

```
> set.seed(1)
> m1 <- cl_consensus(Kinship82, method = "GV3",
+     control = list(k = 3, verbose = TRUE))
```

```
SUMT run: 1
Iteration: 1 Rho: 0.0978703 P: 51.14835
Iteration: 2 Rho: 0.978703 P: 0.2692162
Iteration: 3 Rho: 9.78703 P: 0.1837687
```

```
Iteration: 4 Rho: 97.8703 P: 0.03277258
Iteration: 5 Rho: 978.703 P: 0.000817076
Iteration: 6 Rho: 9787.03 P: 9.285065e-06
Iteration: 7 Rho: 97870.3 P: 9.410234e-08
Iteration: 8 Rho: 978703 P: 9.422904e-10
Iteration: 9 Rho: 9787030 P: 9.424172e-12
Iteration: 10 Rho: 97870297 P: 9.424299e-14
Iteration: 11 Rho: 978702969 P: 9.424312e-16
Iteration: 12 Rho: 9787029692 P: 9.424312e-18
```

This results in a soft partition with average co-membership dissimilarity (the criterion function to be optimized by the consensus partition) of

```
> mean(cl_dissimilarity(Kinship82, m1, "comem")^2)

[1] 28.36927
```

Again, we compare this to the corresponding consensus solution given in Gordon and Vichi (2001):

```
> data("Kinship82_Consensus")
> m2 <- Kinship82_Consensus[["JMF"]]
> mean(cl_dissimilarity(Kinship82, m2, "comem")^2)

[1] 28.49879
```

Interestingly, again we obtain a (this time only "slightly") better solution, with

```
> cl_dissimilarity(m1, m2, "comem")

Dissimilarities using Euclidean comembership distance:
         [,1]
[1,] 0.370891

> table(CLUE = cl_class_ids(m1), GV2001 = cl_class_ids(m2))

    GV2001
CLUE 1 2 3
   1 0 6 0
   2 4 0 0
   3 0 0 5
```

indicating that the two solutions are reasonably close, even though

```
> cl_fuzziness(cl_ensemble(m1, m2))

Fuzziness using normalized partition coefficient:
[1] 0.4360393 0.3894000
```

shows that the solution found by **clue** is "softer".

3.4 Miller-Nicely consonant phoneme confusion data

Miller and Nicely (1955) obtained the data on the auditory confusions of 16 English consonant phonemes by exposing female subjects to a series of syllables consisting of one of the consonants followed by the vowel 'a' under 17 different experimental conditions. Data set `Phonemes` provides consonant misclassification probabilities (i.e., similarities) obtained from aggregating the six so-called flat-noise conditions in which only the speech-to-noise ratio was varied into a single matrix of misclassification frequencies.

These data are used in de Soete (1986) as an illustration of the SUMT approach for finding least squares optimal fits to dissimilarities by ultrametrics. We can reproduce this analysis as follows.

```
> data("Phonemes")
> d <- 1 - as.dist(Phonemes)
```

(Note that the data set has the consonant misclassification probabilities, i.e., the similarities between the phonemes.)

```
> u <- ls_fit_ultrametric(d, control = list(verbose = TRUE))
```

```
SUMT run: 1
Iteration: 1 Rho: 0.001041025 P: 1.461036
Iteration: 2 Rho: 0.01041025 P: 0.1736313
Iteration: 3 Rho: 0.1041025 P: 0.006109456
Iteration: 4 Rho: 1.041025 P: 6.701248e-05
Iteration: 5 Rho: 10.41025 P: 7.051292e-07
Iteration: 6 Rho: 104.1025 P: 7.086383e-09
Iteration: 7 Rho: 1041.025 P: 7.089908e-11
Iteration: 8 Rho: 10410.25 P: 7.090261e-13
Iteration: 9 Rho: 104102.5 P: 7.090296e-15
Iteration: 10 Rho: 1041025 P: 7.090299e-17
```

This gives an ultrametric u for which Figure 5 plots the corresponding dendrogram, "basically" reproducing Figure 1 in de Soete (1986).

```
> plot(u)
```

We can also compare the least squares fit obtained to that of other hierarchical clusterings of d, e.g. those obtained by `hclust()`. The "optimal" u has Euclidean dissimilarity

```
> round(cl_dissimilarity(d, u), 4)
```

```
Dissimilarities using Euclidean ultrametric distance:
       [,1]
[1,] 0.1988
```

to d. For the `hclust()` results, we get

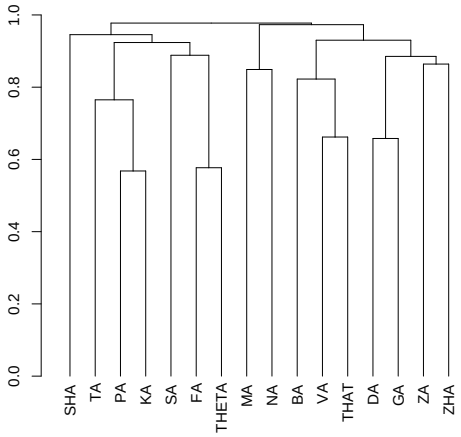

Figure 5: Dendrogram for least squares fit to the Miller-Nicely consonant phoneme confusion data.

```
> hclust_methods <- c("ward", "single", "complete",
+     "average", "mcquitty", "median", "centroid")
> hens <- cl_ensemble(list = lapply(hclust_methods,
+     function(m) hclust(d, m)))
> names(hens) <- hclust_methods
> round(sapply(hens, cl_dissimilarity, d), 4)

    ward   single complete  average mcquitty   median
  4.4122   0.4279   0.3134   0.2000   0.2020   4.0168
centroid
  4.4368
```

which all exhibit greater Euclidean dissimilarity to d than u. We can also compare the "structure" of the different hierarchies, e.g. by looking at the rate of inversions between them:

```
> ahens <- c(L2opt = cl_ensemble(u), hens)
> round(cl_dissimilarity(ahens, method = "gamma"),
+     2)

Dissimilarities using rate of inversions:
         L2opt ward single complete average mcquitty median
ward      0.29
single    0.24 0.45
complete  0.03 0.29   0.27
```

```
average   0.03 0.26   0.24     0.03
mcquitty  0.03 0.26   0.24     0.03    0.00
median    0.79 0.74   0.65     0.79    0.77     0.77
centroid  0.78 0.65   0.74     0.79    0.76     0.76   0.88
```

4 Outlook

Package **clue** was designed as an *extensible* environment for computing on cluster ensembles. It currently provides basic data structures for representing partitions and hierarchies, and facilities for computing on these, including methods for measuring proximity and obtaining consensus and "secondary" clusterings.

Many extensions to the available functionality are possible and in fact planned (some of these enhancements were already discussed in more detail in the course of this paper).

- Provide mechanisms to generate cluster ensembles based on reweighting (assuming base clusterers allowing for case weights) the data set.
- Explore recent advances (e.g., parallelized random search) in heuristics for solving the multi-dimensional assignment problem.
- Add support for *additive trees* (e.g., Barthélémy and Guénoche, 1991).
- Add heuristics for finding least squares fits based on iterative projection on convex sets of constraints, see e.g. Hubert, Arabie, and Meulman (2006) and the accompanying MATLAB code available at `http://cda.psych.uiuc.edu/srpm_mfiles` for using these methods (instead of SUMT approaches) to fit ultrametrics and additive trees to proximity data.
- Add an "L_1 View". Emphasis in **clue**, in particular for obtaining consensus clusterings, is on using Euclidean dissimilarities (based on suitable least squares distances); arguably, more "robust" consensus solutions should result from using Manhattan dissimilarities (based on absolute distances). Adding such functionality necessitates developing the corresponding structure theory for soft Manhattan median partitions. Minimizing average Manhattan dissimilarity between co-memberships and ultrametrics results in constrained L_1 approximation problems for the weighted medians of the co-memberships and ultrametrics, respectively, and could be approached by employing SUMTs analogous to the ones used for the L_2 approximations.
- Provide heuristics for obtaining *hard* consensus partitions.
- Add facilities for tuning hyper-parameters (most prominently, the number of classes employed) and "cluster validation" of partitioning algorithms, as recently proposed by Roth, Lange, Braun, and Buhmann (2002), Lange,

Roth, Braun, and Buhmann (2004), Dudoit and Fridlyand (2002), and Tibshirani and Walther (2005).

We are hoping to be able to provide many of these extensions in the near future.

Acknowledgments

We are grateful to Walter Böhm for providing efficient C code for solving assignment problems.

References

J.-P. Barthélémy and A. Guénoche. *Trees and Proximity Representations*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Chichester, 1991. ISBN 0-471-92263-3.

D. P. Bertsekas and P. Tseng. RELAX-IV: A faster version of the RELAX code for solving minimum cost flow problems. Technical Report P-2276, Massachusetts Institute of Technology, 1994. URL `http://www.mit.edu/dimitrib/www/noc.htm`.

J. C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum, New York, 1981.

S. A. Boorman and P. Arabie. Structural measures and the method of sorting. In R. N. Shepard, A. K. Romney, and S. B. Nerlove, editors, *Multidimensional Scaling: Theory and Applications in the Behavioral Sciences, 1: Theory*, pages 225–249. Seminar Press, New York, 1972.

L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

C. Buchta and M. Hahsler. *cba: Clustering for Business Analytics*, 2005. R package version 0.1-6.

S. E. Buttrey. Calling the `lp_solve` linear program software from R, S-PLUS and Excel. *Journal of Statistical Software*, 14(4), 2005. URL `http://www.jstatsoft.org/v14/i04/`.

G. Carpaneto and P. Toth. Algorithm 548: Solution of the assignment problem. *ACM Transactions on Mathematical Software*, 6(1):104–111, 1980. ISSN 0098-3500. doi: http://doi.acm.org/10.1145/355873.355883.

J. D. Carroll and S. Pruzansky. Discrete and hybrid scaling models. In E. D. Lantermann and H. Feger, editors, *Similarity and Choice*. Huber, Bern, Switzerland, 1980.

I. Charon, L. Denoeud, A. Guénoche, and O. Hudry. Maximum transfer distance between partitions. Technical Report 2005D003, Ecole Nationale Supérieure des Télécommunications — Paris, May 2005. URL `http://www.enst.fr/_data/files/docs/id_515_1128675112_271.pdf`. ISSN 0751-1345 ENST D.

W. H. E. Day. The complexity of computing metric distances between partitions. *Mathematical Social Sciences*, 1:269–287, 1981.

W. H. E. Day. Foreword: Comparison and consensus of classifications. *Journal of Classification*, 3:183–185, 1986.

G. de Soete. A least squares algorithm for fitting an ultrametric tree to a dissimilarity matrix. *Pattern Recognition Letters*, 2:133–137, 1986.

E. Dimitriadou. *cclust: Convex Clustering Methods and Clustering Indexes*, 2005. URL `http://CRAN.R-project.org/`. R package version 0.6-12.

E. Dimitriadou, A. Weingessel, and K. Hornik. A combination scheme for fuzzy clustering. *International Journal of Pattern Recognition and Artificial Intelligence*, 16 (7):901–912, 2002.

E. Dimitriadou, K. Hornik, F. Leisch, D. Meyer, and A. Weingessel. *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*, 2005. URL `http://CRAN.R-project.org/`. R package version 1.5-7.

S. Dudoit and J. Fridlyand. A prediction-based resampling method for estimating the number of clusters in a dataset. *Genome Biology*, 3(7):1–21, 2002. URL `http://genomebiology.com/2002/3/7/resarch0036.1`.

S. Dudoit and J. Fridlyand. Bagging to improve the accuracy of a clustering procedure. *Bioinformatics*, 19(9):1090–1099, 2003.

X. Z. Fern and C. E. Brodley. Solving cluster ensemble problems by bipartite graph partitioning. In *ICML '04: Twenty-first International Conference on Machine Learning*. ACM Press, 2004. ISBN 1-58113-828-5. doi: http://doi.acm.org/10.1145/1015330.1015414.

A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. John Willey & Sons, New York, 1968.

E. W. Forgy. Cluster analysis of multivariate data: Efficiency vs interpretability of classifications. *Biometrics*, 21:768–769, 1965.

E. B. Fowlkes and C. L. Mallows. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78:553–569, 1983.

C. Fraley and A. E. Raftery. Enhanced model-based clustering, density estimation, and discriminant analysis software: MCLUST. *Journal of Classification*, 20(2):263–286, 2003.

C. Fraley, A. E. Raftery, and R. Wehrens. *mclust: Model-based Cluster Analysis*, 2005. URL `http://www.stat.washington.edu/mclust`. R package version 2.1-11.

A. L. N. Fred and A. K. Jain. Data clustering using evidence accumulation. In *Proceedings of the 16th International Conference on Pattern Recognition (ICPR 2002)*, pages 276–280, 2002. URL `http://citeseer.ist.psu.edu/fred02data.html`.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2):337–407, 2000.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, 1979.

R. Gentleman and E. Whalen. *graph: A Package to Handle Graph Data Structures*, 2005. URL `http://www.bioconductor.org/`. R package version 1.5.9.

A. D. Gordon. *Classification*. Chapman & Hall/CRC, Boca Raton, Florida, 2nd edition, 1999.

A. D. Gordon and M. Vichi. Partitions of partitions. *Journal of Classification*, 15: 265–285, 1998.

A. D. Gordon and M. Vichi. Fuzzy partition models for fitting a set of partitions. *Psychometrika*, 66(2):229–248, 2001.

D. Grundel, C. A. Oliveira, P. M. Pardalos, and E. Pasiliao. Asymptotic results for random multidimensional assignment problems. *Computational Optimization and Applications*, 31, 2005. In press.

D. Gusfield. Partition-distance: A problem and class of perfect graphs arising in clustering. *Information Processing Letters*, 82:159–164, 2002.

B. B. Hansen. *optmatch: Functions for Optimal Matching*, 2005. URL `http://www.stat.lsa.umich.edu/~bbh/optmatch.html`. R package version 0.1-3.

J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *Applied Statistics*, 28:100–108, 1979.

J. Hoeting, D. Madigan, A. Raftery, and C. Volinsky. Bayesian model averaging: A tutorial. *Statistical Science*, 14:382–401, 1999.

K. Hornik. Cluster ensembles. In C. Weihs and W. Gaul, editors, *Classification – The Ubiquitous Challenge*, pages 65–72. Springer-Verlag, 2005a. Proceedings of the 28th Annual Conference of the Gesellschaft für Klassifikation e.V., University of Dortmund, March 9–11, 2004.

K. Hornik. A CLUE for CLUster Ensembles. *Journal of Statistical Software*, 14(12), September 2005b. URL `http://www.jstatsoft.org/v14/i12/`.

K. Hornik, T. Hothorn, and A. Karatzoglou. *RWeka: R/Weka Interface*, 2006. R package version 0.2-0.

L. Hubert and P. Arabie. Iterative projection strategies for the least squares fitting of tree structures to proximity data. *British Journal of Mathematical and Statistical Psychology*, 48:281–317, 1995.

L. Hubert, P. Arabie, and J. Meulman. *The Structural Representation of Proximity Matrices With MATLAB*. SIAM, Philadelphia, 2006.

A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis. kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004. URL `http://www.jstatsoft.org/v11/i09/`.

L. Katz and J. H. Powell. A proposed index of the conformity of one sociometric measurement to another. *Psychometrika*, 18:249–256, 1953.

A. M. Krieger and P. E. Green. A generalized Rand-index method for consensus clustering of separate partitions of the same data base. *Journal of Classification*, 16:63–89, 1999.

M. Krivanek. On the computational complexity of clustering. In E. Diday, Y. Escoufier, L. Lebart, J. Pages, Y. Schektman, and R. Tomassone, editors, *Data Analysis and Informatics 4*, pages 89–96. Elsevier/North-Holland, 1986.

M. Krivanek and J. Moravek. NP-hard problems in hierarchical tree clustering. *Acta Informatica*, 23:311–323, 1986.

T. Lange, V. Roth, M. L. Braun, and J. M. Buhmann. Stability-based validation of clustering solutions. *Neural Computation*, 16(6):1299–1323, 2004.

F. Leisch. Bagged clustering. Working Paper 51, SFB "Adaptive Information Systems and Modeling in Economics and Management Science", August 1999. URL `http://www.ci.tuwien.ac.at/~leisch/papers/wp51.ps`.

F. Leisch. FlexMix: A general framework for finite mixture models and latent class regression in R. *Journal of Statistical Software*, 11(8), 2004. URL `http://www.jstatsoft.org/v11/i08/`.

F. Leisch. A toolbox for k-centroids cluster analysis. *Computational Statistics and Data Analysis*, 2006. Accepted for publication.

F. Leisch and E. Dimitriadou. *mlbench: Machine Learning Benchmark Problems*, 2005. URL `http://CRAN.R-project.org/`. R package version 1.0-1.

S. P. Lloyd. Least squares quantization in PCM. Technical Note, Bell Laboratories, 1957.

S. P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28:128–137, 1982.

T. Margush and F. R. McMorris. Consensus n-trees. *Bulletin of Mathematical Biology*, 43(2):239–244, 1981.

M. Meila. Comparing clusterings by the variation of information. In B. Schölkopf and M. K. Warmuth, editors, *Learning Theory and Kernel Machines*, volume 2777 of *Lecture Notes in Computer Science*, pages 173–187. Springer-Verlag, 2003.

G. A. Miller and P. E. Nicely. An analysis of perceptual confusions among some English consonants. *Journal of the Acoustical Society of America*, 27:338–352, 1955.

B. G. Mirkin. *Mathematical Classification and Clustering*. Kluwer Academic Publishers Group, 1996.

C. A. S. Oliveira and P. M. Pardalos. Randomized parallel algorithms for the multidimensional assignment problem. *Applied Numerical Mathematics*, 49(1):117–133, April 2004.

C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, 1982.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2005. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.

S. Rosenberg. The method of sorting in multivariate research with applications selected from cognitive psychology and person perception. In N. Hirschberg and L. G. Humphreys, editors, *Multivariate Applications in the Social Sciences*, pages 117–142. Erlbaum, Hillsdale, New Jersey, 1982.

S. Rosenberg and M. P. Kim. The method of sorting as a data-gathering procedure in multivariate research. *Multivariate Behavioral Research*, 10:489–502, 1975.

V. Roth, T. Lange, M. Braun, and J. M. Buhmann. A resampling approach to cluster validation. In W. Härdle and B. Rönz, editors, *COMPSTAT 2002 – Proceedings in Computational Statistics*, pages 123–128. Physika Verlag, 2002. ISBN 3-7908-1517-9.

P. Rousseeuw, A. Struyf, M. Hubert, and M. Maechler. *cluster: Functions for Clustering (by Rousseeuw et al.)*, 2005. URL `http://CRAN.R-project.org/`. R package version 1.9.8.

M. Sato and Y. Sato. On a multicriteria fuzzy clustering method for 3-way data. *International Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, 2: 127–142, 1994.

R. R. Sokal and F. J. Rohlf. The comparisons of dendrograms by objective methods. *Taxon*, 11:33–40, 1962.

A. Strehl and J. Ghosh. Cluster ensembles – A knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 2003a. ISSN 1533-7928.

A. Strehl and J. Ghosh. Relationship-based clustering and visualization for high-dimensional data mining. *INFORMS Journal on Computing*, 15:208–230, 2003b. ISSN 1526-5528.

A. Struyf, M. Hubert, and P. Rousseeuw. Clustering in an object-oriented environment. *Journal of Statistical Software*, 1(4), 1996. URL `http://www.jstatsoft.org/v01/i04/`.

R. Tibshirani and G. Walther. Cluster validation by prediction strength. *Journal of Computational and Graphical Statistics*, 14(3):511–528, 2005.

D. L. Wallace. Comments on "A method for comparing two hierarchical clusterings". *Journal of the American Statistical Association*, 78:569–576, 1983.