Clusterv tutorial

Giorgio Valentini
Dipartimento di Informatica "Giovanni Degli Antoni"
Università degli Studi di Milano
e-mail : valentini@di.unimi.it

Contents

1 Introduction **3**

2 Overview of the *clusterv* R package **3**

3 Main functionalities implemented in *clusterv* **4**

4 Getting started with *Clusterv* **5**

5 Generation of randomly projected data **11**

6 Computation of the distortion induced by random projections **13**

7 Estimate of the reliability of clusters generated by specific clustering algorithms **17**

7.1 Estimate of the reliability of clusters generated by hierarchical clustering algorithms 18

7.2 Estimate of the reliability of clusters generated by the k-means clustering algorithm 20

7.3 Estimate of the reliability of clusters generated by the fuzzy k-means clustering algorithm 21

7.4 Estimate of the reliability of clusters generated by the PAM (Prediction Around Medoids) clustering algorithm 25

8 Estimate of the reliability of clusters generated by a generic clustering algorithm **26**

9 An applicative example to the analysis of DNA microarray data: analysis of cluster reliability in lung tumor patients **30**

1 Introduction

In this tutorial we provide some practical examples to introduce the usage of the *clusterv* R package. For details about the single functions implemented in the library, please, see the *Reference manual*.

In the next two section we provide an overview of the package and of its main functionalities. Then we introduce a first complete example on how to easily use the package to evaluate the results of a specific clustering algorithm (Getting started with *Clusterv*). The remaining sections describe the different specific functionalities of the package and provide also some applicative examples to the analysis of the reliability of clusters discovered in DNA microarray data.

2 Overview of the *clusterv* R package

The *clusterv* R package implements a set of functions to assess the reliability of clusters discovered by clustering algorithms. This library is tailored to the analysis of high dimensional data and in particular it is conceived for the analysis of the reliability of clusters discovered using DNA microarray data.

Indeed cluster analysis has been used for investigating structure in microarray data, such as the search of new tumor taxonomies [1],[4],[8]. It provides a way for validating groups of patients according to prior biological knowledge or to discover new "natural groups" inside the data. Anyway, clustering algorithms always find structure in the data, even when no structure is present instead. Hence we need methods for assessing the validity of the discovered clusters to test the existence of biologically meaningful clusters.

To assess the reliability of the discovered classes, *clusterv* provides a set of measures that estimate the stability of the clusters obtained by perturbing the original data set. This perturbation is achieved through random projections of the original high dimensional data to lower dimensional subspaces, approximately preserving the distances between examples, in order to avoid too large distortions of the data. These random projections are repeated many times and each time a new clustering is performed. The obtained multiple clusterings are then compared with the clustering for which we need to evaluate its reliability. Intuitively a cluster will be reliable if it will be maintained across multiple clusterings performed in the lower dimensional subspaces. The measures provided by *clusterv* are based on the evaluation of the stability of the clusters across multiple random projections. By these measures we can assess:

1. the reliability of single individual clusters inside a clustering

2. the reliability of the overall clustering (that is, an estimate of the "optimal" number of clusters)

3. the confidence by which example may be assigned to each cluster

This tutorial introduces to the usage of the package, providing also some examples of applications of the stability measures to synthetic and real DNA microarray data.

3 Main functionalities implemented in *clusterv*

The *clusterv* R package provides a set of functionalities to assess the reliability of clusters discovered in data characterized by high-dimensionality.

Most of the functions are independent of the specific clustering algorithm used, in the sense that may be used by different distance-based clustering algorithms (e.g. k-means, hierarchical clustering, Self-Organizing-Maps, PAM) to compute the stability indices for assessing the reliability of the clusters.

Other functions are high-level functions for a specific clustering algorithm: they directly cluster the data and provide the stability measures to evaluate the reliability of clusters produced by a specific clustering algorithm.

The functionalities provided by the *clusterv* package can be summarized in the following list:

- **Functions clustering-algorithm-dependent**
  - Functions for high dimensional synthetic data generation
  - Functions to implement different types of random projections from high to lower dimensional subspaces
  - Functions to evaluate the distortion induced by random projections
  - Functions to compute the similarity matrix
  - Functions to compute the stability indices:
    * Individual cluster stability index for the estimate of the reliability of individual clusters inside a clustering.
    * Overall cluster stability index for the estimate of the "optimal" number of clusters.
    * Assignment-Confidence index for the estimate of the confidence by which an example may be assigned to a specific cluster.
- **Functions clustering-algorithm dependent**

- Functions to perform multiple clusterings on multiple instances of projected data
- Functions to compute the stability indices for a specific clustering algorithm

4 Getting started with *Clusterv*

In this section we analyze the reliability of clusters generated by the application of a hierarchical clustering algorithm to high dimensional synthetic data.

As a first step, we need to load the *clusterv* library:

```
> library(clusterv)
```

The clusterv library requires two libraries *cluster* and *MASS* that are usually available on all R environments. In the unlikely hypothesis that these libraries are not installed in your R environment it is straightforward to download them from the *R web site*: `http://www.r-project.org`.

Then we generate a synthetic data set that we will use for our reliability analysis:

```
> M <- generate.sample0(n=10, m=1, sigma=1, dim=6000)
```

The function `generate.sample0` generates a 6000-dimensional data set with 3 clusters composed each one of 1O examples. The data are distributed according to a multivariate spherical gaussian distribution with a covariance matrix equal to an identity matrix. The three clusters are centered, the first one in the **0** vector (that is a a 6000-dimensional vector with all '1'), the second in the **1** and the third in **-1** vectors.

Then we want to perform a reliability analysis of the clustering obtained with the hierarchical clustering Ward's method, choosing a cut (number of clusters) corresponding to 2. To this end we choose an Achlioptas random projection and a subspace dimension such that the maximum distortion will be less than 1.2 (see the *clusterv* web site for more details about this topic).

Hence we need to compute first the subspace dimension according to the *JL lemma* with 1+epsilon distortion:

```
> subspace.dim <- ceiling(JL.predict.dim(30, epsilon=0.2));
> subspace.dim
[1] 341
```

That is we will perform random projection from 6000 to 341-dimensional subspaces. Then we perform the clustering on the original space, and to evaluate its

reliability we perform 20 Achlioptas random projections into 341-dimensional subspaces, performing 20 hierarchical clusterings on that subspaces to compute the stability indices:

```
> l2 <- Random.hclustering.validity (M, dim=subspace.dim, hmethod="ward.D",
pmethod="Achlioptas", c=2, n=20, scale=TRUE, seed=100, AC=TRUE);
```

The list `l2` is composed by different elements that store the different stability indices computed and other informations:

```
> l2$overall.validity
[1] 0.9210526
> l2$validity
[1] 1.0000000 0.8421053
```

These results show that the reliability (overall validity) of the clustering is high (0.9210) and the validity of the obtained 2 individual clusters are respectively 1.0000 and 0.8421.

We could repeat the same test, but this time choosing 3 clusters for the partition (we need only to change the parameter `c=3`, indicating that we test a 3-clusters clustering:

```
> l3 <- Random.hclustering.validity (M, dim=subspace.dim, c=3, n=20,
pmethod="Achlioptas", hmethod="ward.D", scale=TRUE, seed=100, AC=TRUE);
> l3$overall.validity
[1] 1
> l3$validity
[1] 1 1 1
```

In this case we achieve the maximum of the reliability, both for the overall clustering and for the individual clusters.

We repeat now the same test with `c=4, 5, 10` clusters:
4 clusters partition:

```
> l4 <- Random.hclustering.validity (M, dim=subspace.dim, c=4,  n=20,
pmethod="Achlioptas", hmethod="ward.D", scale=TRUE, seed=100, AC=TRUE);
> l4$overall.validity
[1] 0.8245833
> l4$validity
[1] 0.8911111 0.7755556 0.8250000 0.8066667
```

5 clusters partition:

```
> l5 <- Random.hclustering.validity (M, dim=subspace.dim, c=5, n=20,
pmethod="Achlioptas", hmethod="ward.D",  scale=TRUE, seed=100, AC=TRUE);
> l5$overall.validity
[1] 0.7097778
> l5$validity
[1] 0.7500000 0.7350000 0.6055556 0.7416667 0.7166667
```

10 clusters partition:

```
> l10 <- Random.hclustering.validity (M, dim=subspace.dim, c=10, n=20,
pmethod="Achlioptas", hmethod="ward.D", scale=TRUE, seed=100, AC=TRUE);
> l10$overall.validity
[1] 0.3213333
> l10$validity
 [1] 0.3800000 0.1500000 0.3250000 0.2750000 0.4500000 0.2500000
 [7] 0.2833333 0.3166667 0.4000000 0.3833333
```

We know in advance that the correct number of clusters is 3. The stability indices correctly detect that the most likely clustering is composed by 3 clusters, and each cluster is highly reliable. Note that with 2, 4, 5 clusters partitions we obtain lower values for the stability indices, and with 10-clusters partition, the unnatural fragmentation of the clusters lead to very low values of the stability indices.

The element `l$AC` of the list returned by `Random.hclustering.validity` is a matrix that returns the "confidence" by which we can assign an example to a cluster:

```
> l3$AC
      [,1] [,2] [,3]
 [1,]    0    1    0
 [2,]    0    1    0
 [3,]    0    1    0
 [4,]    0    1    0
 [5,]    0    1    0
 [6,]    0    1    0
 [7,]    0    1    0
 [8,]    0    1    0
 [9,]    0    1    0
[10,]    0    1    0
[11,]    1    0    0
[12,]    1    0    0
```

```
[13,]    1    0    0
[14,]    1    0    0
[15,]    1    0    0
[16,]    1    0    0
[17,]    1    0    0
[18,]    1    0    0
[19,]    1    0    0
[20,]    1    0    0
[21,]    0    0    1
[22,]    0    0    1
[23,]    0    0    1
[24,]    0    0    1
[25,]    0    0    1
[26,]    0    0    1
[27,]    0    0    1
[28,]    0    0    1
[29,]    0    0    1
[30,]    0    0    1
```

The rows refer to the examples, columns to the cluster: in this case we have that the assignment is highly reliable for the example, but the *AC-index* may have values between 0 (no reliable assignment) to 1 (highly reliable assignment).

However, to perform a deeper and systematic analysis is preferable to use R scripts that automatically launch multiple instances of the function `Random.hclustering.validity` and automatically store the corresponding results for further analysis and visualization. An example of such a script is downloadable from:
https://valentini.di.unimi.it/SW/clusterv/examples/sample0.RSvalidity.R . This script performs a reliability analysis on a data set generated with the same generator we used in our example, but random subspace projections are used instead.

The results are summarized in the following figure (Fig. 1) that represents the dendrogram of the clustering and table (Tab. 1) where the corresponding validity indices are shown. Values with **S** refer to the overall stability index, while the other values inside the table represent the stability index of an individual cluster. In each column are computed the stability measures using random projections into different subspace dimensions corresponding to different 1 + epsilon (eps) distortion, according to the *JL lemma* (see the *clusterv* web site for more details about this topic).

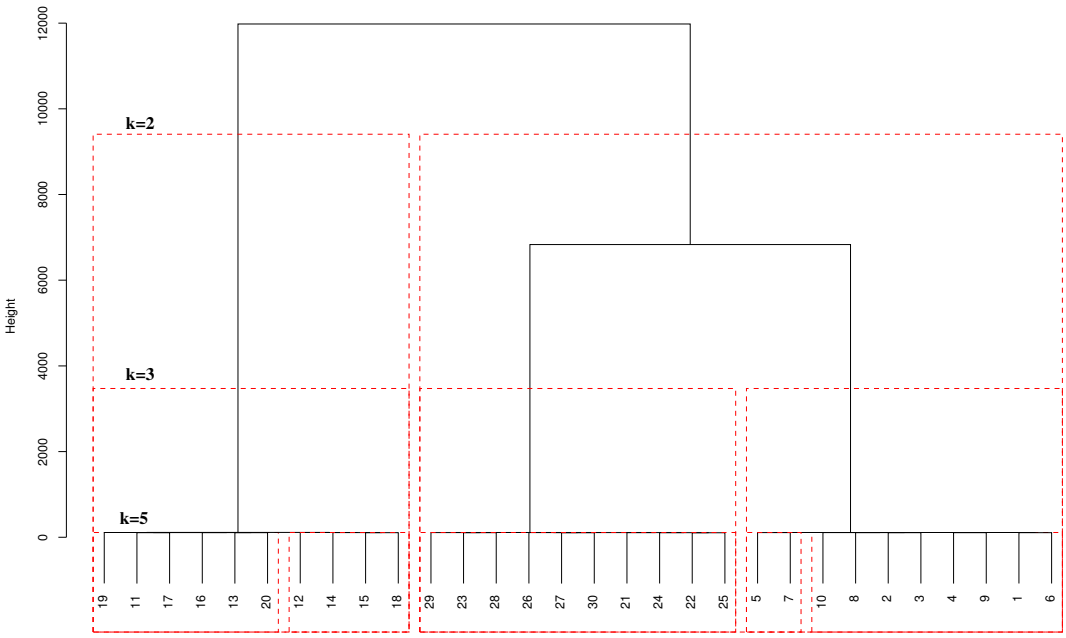

Figure 1: Hierarchical clustering of examples obtained with the Ward method. Gray dotted lines cut the dendrogram such that exactly k clusters are produced, for k=2,3,5.

Table 1: Estimate of cluster stability with random subspace projections

Clusters	**Stability index s**				
	eps=0.5	eps=0.4	eps=0.3	eps=0.2	eps=0.1
2 clusters	S = 0.8631	S = 0.8684	S = 0.8684	S = 0.9157	S = 0.9421
1	1.0000	1.0000	1.0000	1.0000	1.0000
2	0.7263	0.7368	0.7368	0.8314	0.8842
3 clusters	S = 1.0000	S = 1.0000	S = 1.0000	S = 1.0000	S = 1.0000
1	1.0000	1.0000	1.0000	1.0000	1.0000
2	1.0000	1.0000	1.0000	1.0000	1.0000
3	1.0000	1.0000	1.0000	1.0000	1.0000
5 clusters	S = 0.7059	S = 0.6843	S = 0.7044	S = 0.7004	S = 0.7472
1	0.6973	0.7346	0.7293	0.6506	0.7560
2	0.6666	0.7066	0.6866	0.6466	0.7133
3	0.7155	0.7582	0.7448	0.7591	0.8364
4	0.7600	0.5600	0.6800	0.7400	0.7800
5	0.6900	0.6621	0.6814	0.7057	0.6507
10 clusters	S = 0.3093	S = 0.3043	S = 0.2651	S = 0.3286	S = 0.3936
1	0.0600	0.1200	0.0600	0.2000	0.2400
2	0.4260	0.3520	0.2900	0.3360	0.4560
3	0.1400	0.1600	0.1600	0.2000	0.1400
4	0.4066	0.3533	0.3200	0.3800	0.4200
5	0.3733	0.3000	0.2866	0.3600	0.4200
6	0.3276	0.3419	0.3285	0.3866	0.3933
7	0.3600	0.2800	0.3000	0.3600	0.3800
8	0.3000	0.3366	0.3066	0.3433	0.3866
9	0.3400	0.4000	0.2600	0.4200	0.5000
10	0.3600	0.4000	0.3400	0.3000	0.6000

5 Generation of randomly projected data

Different types of *random projections* are available with *clusterv*:

- *Plus-Minus-One (PMO)*
- *Achlioptas*
- *Normal*
- *Random Subspace (RS)*

Before looking at some examples, we'll see how to generate synthetic data that we will use in our examples. Different synthetic data generators named `generate.sampleX` are available, where X is between 0 and 5. They generate clusters of data distributed according to multivariate gaussian distributions. Each generator provides from 2 to 5 clusters, each one characterized by its mean and covariance matrix. Usually the mean (center of each cluster) and the covariance matrix are input parameters for the functions (see the *Clusterv* reference manual for more details).

For instance:

```
> M <- generate.sample1(n = 20, m = 6, sigma = 1, dim = 2000)
```

generates a matrix **M** of data (examples are in columns, variables on rows) with 3 clusters, each one composed by n=20 examples whose dimension is dim=2000. All clusters have their last dim-500 variables centered in 0. The first class (first n examples) has its first 500 features centered in 0. The parameter **m** select the center for the second and third cluster: the second class (second n examples) has its first 500 features centered in 6, the third (last n examples) has its first 500 features centered in -6. All the clusters are distributed according to a "spherical" gaussian with sigma=1. The relating matrix **M** is composed by 2000 rows and 60 columns:

```
> dim(M)
[1] 2000   60
```

As another a little bit more complex example consider:

```
> M5 <- generate.sample5(n = 10, m = 2, ratio.noisy = 0.9, dim = 1000)
```

This generates a 1000 x 40 data matrix **M5**: 4 clusters with n = 10 examples 1000-dimensional are randomly generated. The parameter **ratio.noisy** sets the proportion of "noisy" features, where for "noisy" feature we mean features that

are equally distributed in all the classes (these variables are centered in 0), while for "no-noisy" we mean features that are centered in different points (set by the **m** parameter) in the different classes. In this case we have 1000*0.9=900 "noisy" variables, and 100 "no-noisy" variables, centered in 0 for the first cluster, 2 for the second, -2 for the third and centered in (2,-2) alternatively for the fourth. The covariance matrix Sigma is equal for all the cluster: Sigma = (B, Zero; Zero', I) where B is a (dim*(1-ratio.noisy))x(dim*(1-ratio.noisy)) matrix (in this case a 100 x 100 matrix) s.t. B[i,i]=1, B[i,i+1]=B[i,i-1]=0.5 and B[i,j]=0.1 if j!=i-1,i,i+1; Zero is a (dim*(1-ratio.noisy))x(dim*ratio.noisy) zero matrix (in this case a 100 x 900 matrix) and Zero' its transpose; I is a (dim*ratio.noisy)x(dim*ratio.noisy) identity matrix (in this case a 900 x 900 matrix).

Now we will apply different random projections to these two (quite) high dimensional data matrices. For instance we could apply a *Plus-Minus-One (PMO)* random projection:

```
> M.PMO <- Plus.Minus.One.random.projection(d = 50, M)
> dim(M.PMO)
[1] 50 60
```

This function performs a PMO random projection of the data matrix M into a 50-dimensional subspace.

The other functions that implements random projections have a similar syntax:

```
> M.Achlioptas <- Achlioptas.random.projection(d = 50, M)
> M.Normal <- norm.random.projection(d = 50, M)
> M.RS <- random.subspace(d = 50, M)
```

In all cases the functions return a 50 x 60 matrix using different random projections. You can get a look to the different projected matrices: they differ one from another not only because different random projections are performed, but also because each time a different random matrix is generated by the randomized map. For instance if you now perform a second random projection with *Plus-Minus-One (PMO)*, and compare the result with the previously computed *(PMO)* data matrix you'll get different results:

```
> M.PMO.2 <- Plus.Minus.One.random.projection(d = 50, M)
> R <- M.PMO == M.PMO.2
```

Indeed the elements of the resulting boolean R matrix get all the FALSE value, and this is not the effect of round-off errors or permutations of the columns, as shown by the plot of the two first principal components of the data (Fig. 2):

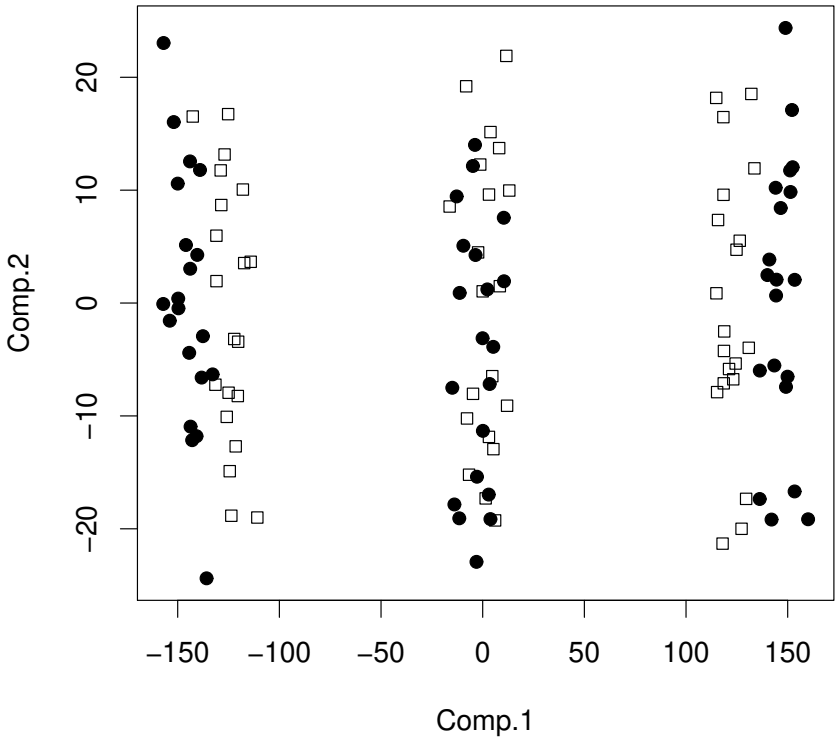

Figure 2: Plot of the two principal components of the data sets represented in matrices M.PMO (solid circles) and M.PMO.2 (squares)

We would like to use random projections to reduce the dimensionality of the data, but without introduce too large distortions into the projected data, in order to use them to perform clustering. How could we do this possibly in a principled way? This is the subject of the next section of the tutorial.

6 Computation of the distortion induced by random projections

Our goal is now to use random projections to reduce the dimensionality of the data, but without introduce too large distortions into the projected data. In this way if the distances between the examples were approximately maintained we could safely apply clustering to lower dimensional data, without destroying

the metric properties of the data.

To do this in a principled way we need to experimentally evaluate the distortion introduced into the projected data. The *clusterv* package implements several functions to evaluate the distortion. The main functions are:

```
> max.exps <- Max.Expansion(t(M), t(M.PMO))
> max.exps
[1] 1.277541
```

`Max.Expansion` computes the maximum ratio of the distances between corresponding examples in the projected space and in the original space (this is named also *distortion*). Note that we used the matrices computed in the previous section (see Sect. 5).

```
> min.exps <- Min.Expansion(t(M), t(M.PMO))
> min.exps
[1] 0.7271939
```

Analogously `Min.Expansion` computes the minimum ratio of the distances between corresponding examples in the projected space and in the original space.

The function `Average.Expansion` computes, of course the average ratio of the distances between corresponding examples in the projected space and in the original space:

```
> avg.exps <- Average.Expansion(t(M), t(M.PMO))
> avg.exps
[1] 1.052216
```

There exist also function that estimate the distortion (that is the maximum expansion) theoretically predicted by the Johnson Lindenstrauss lemma (see the *clusterv* web site for more details about this topic). In the previous case the 1 + epsilon distortion predicted for the examples stored in the matrix **M.PMO** is:

```
> JL.predict.distortion(60, dim = 50)
[1] 0.5723177
```

that is, the predicted distortion is 1..5723177, as the function returns the epsilon value. As you can see this is quite larger than the observed empirical distortion: indeed the *JL lemma* provides an upper bound to the distortion.

Another useful function provides the dimension of the subspace for which we could expect a given epsilon distortion, given the cardinality of the sample to be projected:

```
> JL.predict.dim(n=60, epsilon = 0.1)
[1] 1638
```

That is, according to the *JL lemma* we should project our sample composed by 60 examples into a 1638-dimensional subspace to expect a maximum expansion no larger than 1.1.

These functions, together with the functions to perform random projections described in the previous section, allow us to compare the empirical distortions induced by the different randomized projections, as well as to compare the observed empirical distortions with the theoretical distortion predicted through the *JL lemma*.

In the rest of this section we present the results of an experimental analysis of the distortion induced in DNA microarray data by different types of random projections. An example of the R scripts that we used to implement this empirical analysis is downloadable from:
https://valentini.di.unimi.it/SW/clusterv/examples/Shipp.DLBCL.filtered.norm-PMO.R
and the DNA microarray data set that we used is downloadable in gzipped format from:
https://valentini.di.unimi.it/SW/clusterv/examples/Shipp.DLBCL.filtered.norm.nolabels.txt.gz.
This R script refers to an analysis of the distortion induced by *PMO* random projections, but it is straightforward to modify it for using the other random projections implemented in the package.

The data we used have been downloaded from the MIT Whitehead Institute. The samples are tumor specimens from 8 Diffuse large B-cell lymphoma (DLBCL) and 19 Follicular lymphoma (FL) patients [9]. For each patient, expression levels of 7129 genes or EST sequences are provided from Affymetrix HU6800 oligonucleotide arrays. Raw data have been pre-processed and re-scaled according to the procedures described in [9], obtaining a final set of 6286 genes.

We performed *Achlioptas*, *Normal*, *PMO* and *RS* random projections and then we experimentally evaluated the expectation of the maximum, minimum and average expansion, averaging their values over 50 repeated random projections. Then we compared the results with the estimated theoretical distortion predicted by the *JL lemma*. We performed random projections into subspace whose dimensions correspond to predicted distortions 1+epsilon, with epsilon values ranging from 0.1 to 0.5 (Fig. 3)

In abscissa are represented the dimensions of the projected subspace and in ordinate the corresponding distortion. Continuous lines represent the bounds of the maximum and minimum expansion according to the *JL* lemma; dashed lines

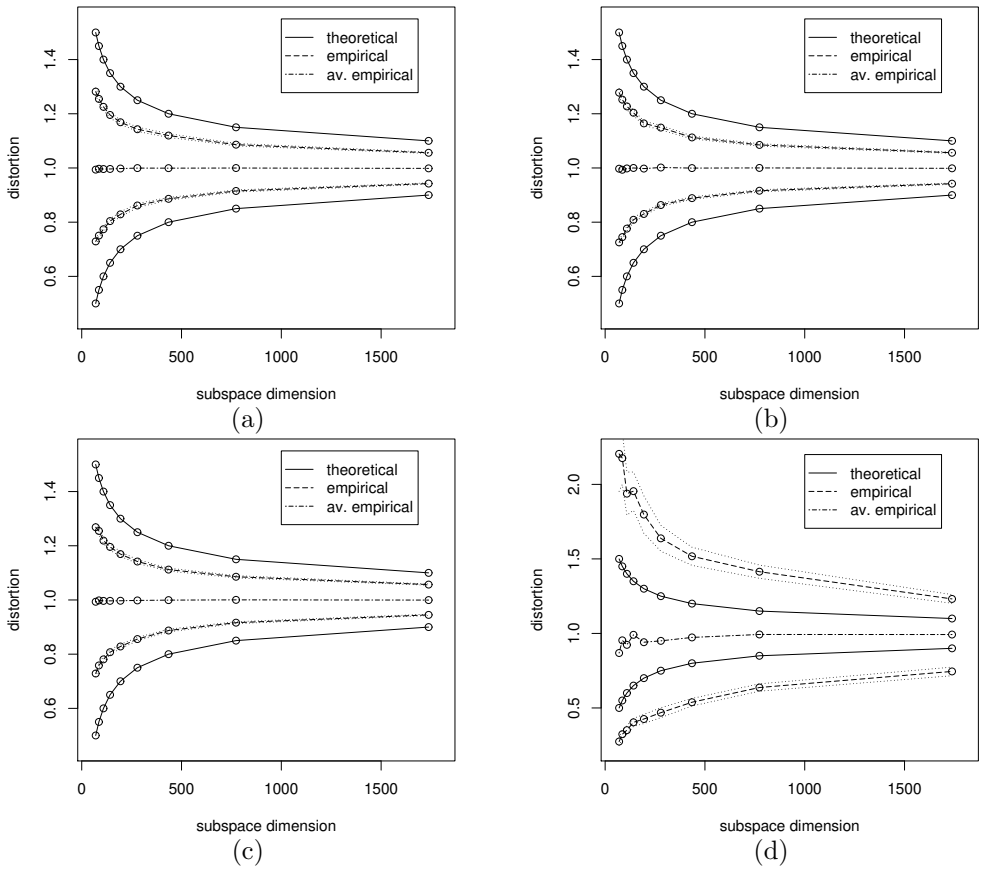

Figure 3: Comparing theoretical and empirical distortion with *DLBCL* samples using (a) *Achlioptas* (b) *Normal* (c) *PMO* and (d) *RS* projections. Continuous lines represent the bounds of the maximum and minimum distortion (expansion) according to the *JL* lemma. Dashed lines represent the average maximum and minimum expansion empirically computed and averaged over 50 random projections. The pairs of dotted lines just above and below the the dashed lines represent the confidence interval at 99 % confidence level. The dash-dotted line represents the average expansion.

represent the empirical average maximum and minimum expansion computed and averaged over 50 random projections. The pairs of dotted lines just above and below the dashed lines represent the confidence interval at 99 % confidence level. The dash-dotted line represents the average distortion. The dashed and dot-dashed lines represent the corresponding estimated empirical values (the lines are simply computed by linear interpolation between points). We recall that distortion equal to 1 means no distortion.

For the *Achlioptas*, *Normal* and *PMO* random projections, the empirical

bounds of the maximum and minimum distortions are largely inside the theoretical bounds predicted by the *JL lemma* (Fig. 3 a, b and c). On the contrary, with *RS* random projections in both cases the maximum and minimum distortions are largely outside the theoretical bounds (Fig.]3 (d)). Note also that these results are significant at 99 % confidence level, as the dashed lines of the confidence intervals do not intersect the continuous lines of the theoretical bounds. In any case the average empirical distortion is very close to 1 (that is we have no distortion on the average), even if with *RS* projections for large values of epsilon (that corresponds to very low dimensional subspaces) the average empirical distortions moves slightly from 1.

These results show that using random subspace (*RS*) projections with DNA microarray data we may introduce large distortions in the data, especially when large values of epsilon are used. We strongly suggest to use one of the other random projections (*Achlioptas*, *Normal* or *PMO*).

7 Estimate of the reliability of clusters generated by specific clustering algorithms

Several high-level function implement the computation of the stability indices for specific clustering algorithms, such as hierarchical [7], k-means [5], fuzzy c-mean [2] and Prediction Around Medoids [6] clustering algorithms.

There are 4 functions that implement the computation of the stability indices for the above clustering algorithms:

1. `Random.hclustering.validity`
2. `Random.kmeans.validity`
3. `Random.fuzzy.kmeans.validity`
4. `Random.PAM.validity`

These functions compute the validity indices for each individual cluster, the overall validity index of the clustering and the AC indices of each example. To this end different randomized maps may be chosen (input parameter of the function) to generate multiple projected data. On each projected data the clustering algorithm is applied; then the similarity matrix across the multiple clusterings on the projected data is computed and finally, by using the previously computed similarity matrix, the validity indices for each individual cluster, the overall validity index of the clustering and the AC indices of each example are

computed. These functions assume that the data are represented as matrices (or data frames) having examples in the columns and variables in rows. Moreover they assume that the labels of the examples are integer starting from 1 to the number of columns of the data matrix.

Consider, for instance, instance a 2000-dimensional synthetic data set composed by three clusters, each one with 15 examples:

```
> M <- generate.sample1(n = 15, m = 4, sigma = 2, dim = 2000)
```

7.1 Estimate of the reliability of clusters generated by hierarchical clustering algorithms

If we want to estimate the reliability of clusters generated by a hierarchical clustering:

```
> l2 <- Random.hclustering.validity(M, dim=JL.predict.dim(45,0.2),
              pmethod = "PMO", c = 2, hmethod = "average", n = 20)
```

We chose a dimension **dim** of the subspace such that the distortion induced by the *PMO* projection (parameter **pmethod**) will be less than 1.2. Moreover we chose an "average linkage" hierarchical clustering (parameter **hmethod**) with 20 replications (parameter **n**) of the clusterings in the projected subspace. The stability indices are stored in list `l2` and are computed with respect to a partition with 2 clusters (parameter **c**). The returned list has 8 elements (comprising the compute stability indices, the similarity matrix, the list of clusterings and other):

```
> l2$overall.validity
[1] 0.9353448
> l2$validity
[1] 1.0000000 0.8706897
```

The overall validity with 2 clusters is equal to 0.9353448, while the stability indices for the two clusters are respectively 1.0000000 and 0.8706897. The element `l2$orig.cluster` stores the clustering for which we computed the stability indices:

```
> l2$orig.cluster
[[1]]
 [1] 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
[[2]]
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 31 32 33 34 35 36 37 38 39 40
[26] 41 42 43 44 45
```

We see that the first cluster is highly reliable (indeed it corresponds to the examples of the second "natural" cluster), while the second less reliable corresponds to the merging of the first and third "natural" cluster.

It is very simple to estimate the reliability of clusters obtained with partitions of 3-clusters; it is sufficient to change only the **c** parameter:

```
> l3 <- Random.hclustering.validity(M, dim=JL.predict.dim(45,0.2),
             pmethod = "PMO", c = 3, hmethod = "average", n = 20)
> l3$overall.validity
[1] 1
> l3$validity
[1] 1 1 1
> l3$orig.cluster
[[1]]
 [1] 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
[[2]]
 [1] 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
[[3]]
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

In this case the clusters obtained are those that correspond to the "true" cluster generated by the synthetic data generator `generate.sample1`. Note that the stability indices denote as highly reliable both the overall clustering and the obtained individual clusterings.

Using other number of clusters (e.g. 4,5,6,8) we obtain less reliable clusterings and less reliable individual clusters. For instance with 4 clusters:

```
> l4 <- Random.hclustering.validity(M, dim=JL.predict.dim(45,0.2),
             pmethod = "PMO", c = 4, hmethod = "average", n = 20)
> l4$overall.validity
[1] 0.7840842
> l4$validity
[1] 0.9666667 0.9542857 0.2500000 0.9653846
> l4$orig.cluster
[[1]]
 [1] 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
[[2]]
 [1] 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
[[3]]
[1] 14
```

```
[[4]]
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 15
```

Note the third cluster is scored as poorly reliable (s=0.25) and indeed it corresponds to an example that should belong to the fourth cluster. Looking at the AC indices that store how much reliable is the assignment of an example to a specific cluster, we see that the AC index of the example 14 (the "unnatural" element of the third singleton cluster) is very low:

```
> l4$AC[14,]
[1] 0.00 0.00 0.25 0.00
```

indeed it reveals a membership to the third cluster equal to 0.25, while the AC indices for, e.g. the first 5 examples (that belong to the fourth cluster) is close to 1:

```
> l4$AC[1:5,]
     [,1] [,2] [,3]      [,4]
[1,]    0    0    0 0.9807692
[2,]    0    0    0 0.8884615
[3,]    0    0    0 0.9807692
[4,]    0    0    0 0.9807692
[5,]    0    0    0 0.9423077
```

Note that the AC indices are stored as matrices, even if it would be sufficient to store them simply through a vector, since with the hierarchical clustering we have, for a given cut of the tree, a strict partition of the data. Anyway we choose this implementation because in future releases we plan to implement fuzzy AC indices specific for fuzzy partitions in order to permit for each example a fuzzy membership to each cluster.

The reader could try to repeat these computations with other random projections: in this case we should obtain the same results, independently of the chosen projection.

7.2 Estimate of the reliability of clusters generated by the k-means clustering algorithm

Similarly we may estimate the reliability of clusters obtained with k-means clustering. The function `Random.kmeans.validity` has about the same syntax as the previous one:

```
> l3 <- Random.kmeans.validity(M, dim=JL.predict.dim(45,0.2),
```

```
                              pmethod = "PMO", c = 3, n = 20)
> l3$overall.validity
[1] 0.9504762
> l3$validity
[1] 1.0000000 0.8961905 0.9552381
> l3$orig.cluster
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
[[2]]
 [1] 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
[[3]]
 [1] 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
```

The results are quite similar to the results obtained with hierarchical clustering, but the validity indices are in this case slightly lower. Let's try to repeat the computations:

```
> l3 <- Random.kmeans.validity(M, dim=JL.predict.dim(45,0.2),
                              pmethod = "PMO", c = 3, n = 20)
> l3$validity
[1] 0.5089655 0.9500000 0.9433333
> l3 <- Random.kmeans.validity(M, dim=JL.predict.dim(45,0.2),
                              pmethod = "PMO", c = 3, n = 20)
> l3$validity
[1] 1.0000000 0.8933333 0.9495238
```

What is happened? The results are quite different. This is due to the fact that the k-means clustering algorithm strongly depends on the initial conditions and we may obtain different results in different runs. If you try to repeat these experiments it is likely that you obtain in turn other results.

7.3 Estimate of the reliability of clusters generated by the fuzzy k-means clustering algorithm

The following example shows how to apply the stability indices to the analysis of clusterings generated through a fuzzy-k-means algorithm. Here we consider only the partition obtained by assigning each example to the cluster with the higher membership. We used now a synthetic 6000-dimensional data set with 5 clusters:

```
> M <- generate.sample4(n = 10, sigma = 0.1)
```

The obtained 5 clusters (each one with 10 examples) are quite well separated, as shown by plotting the points projected along the two main principal components via PCA (Fig. 4). To apply the stability analysis we may use the

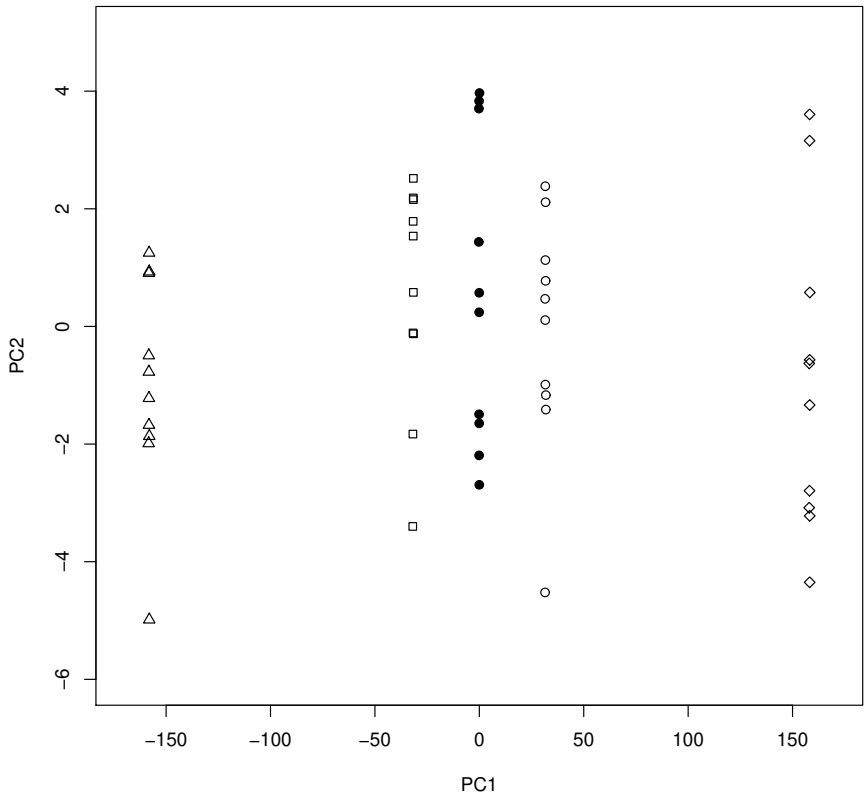

Figure 4: Plot of the two principal components of the 6000-dimensional data set. The examples that belong to the 5 different clusters are plotted with different symbols.

`Random.fuzzy.kmeans.validity` function:

```
> r2 <- Random.fuzzy.kmeans.validity(M, dim=JL.predict.dim(50,0.2),
                                     pmethod = "PMO", c = 2, n = 20);
> r2$overall.validity
[1] 1
> r2$validity
[1] 1 1
```

Hence two clusters are considered reliable. But we know that we have 5 clusters. Getting a look to the clusters, we see that the first 4 clusters are grouped

together against the fifth (the first 10 examples belongs to the first cluster, the next ten to the second and so on):

```
> r2$orig.cluster
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
[21]  21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
[[2]]
 [1] 41 42 43 44 45 46 47 48 49 50
```

Does the stability indices fail or the results depend on how the clusters are defined? Indeed the fuzzy-k-means sees 2 clusters in the data, and the fifth cluster is surely separated from the other data.

Consider now **c**=3 clusters:

```
> r3 <- Random.fuzzy.kmeans.validity(M, dim=JL.predict.dim(50,0.2),
                                     pmethod = "PMO", c = 3, n = 20);
> r3$overall.validity
[1] 1
> r3$validity
[1] 1 1 1
> r3$orig.cluster
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
[21]  21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
[[2]]
 [1] 31 32 33 34 35 36 37 38 39 40
[[3]]
 [1] 41 42 43 44 45 46 47 48 49 50
```

In this case also 3 clusters are considered reliable, because really the first clusters is composed by the three "true" clusters in the center (see Fig. 4), while the two other clusters on the left and on the right are more separated.

And with 5 clusters?

```
> r5 <- Random.fuzzy.kmeans.validity(M, dim=JL.predict.dim(50,0.2),
                                     pmethod = "PMO", c = 5, n = 20);
> r5$overall.validity
[1] 1
> r5$validity
[1] 1 1 1 1 1
```

```
> r5$orig.cluster
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10
[[2]]
 [1] 11 12 13 14 15 16 17 18 19 20
[[3]]
 [1] 21 22 23 24 25 26 27 28 29 30
[[4]]
 [1] 31 32 33 34 35 36 37 38 39 40
[[5]]
 [1] 41 42 43 44 45 46 47 48 49 50
```

Of course, the stability indices considered the five "true" clusters very reliable.

With 4 clusters are considered very reliable only the two "extreme clusters", as the first big cluster losses its example n.5 that is assigned to the second cluster:

```
> r4 <- Random.fuzzy.kmeans.validity(M, dim=JL.predict.dim(50,0.2),
                                     pmethod = "PMO", c = 4, n = 20);
> r4$overall.validity
[1] 0.8899123
> r4$validity
[1] 0.6596491 0.9000000 1.0000000 1.0000000
> r4$orig.cluster
[[1]]
 [1]  1  2  3  4  6  7  8  9 10 21 22 23 24 25 26 27 28 29 30
[[2]]
 [1]  5 11 12 13 14 15 16 17 18 19 20
[[3]]
 [1] 31 32 33 34 35 36 37 38 39 40
[[4]]
 [1] 41 42 43 44 45 46 47 48 49 50
```

If we use an unnatural number of cluster (e.g. 10) we obtain low values of the stability indices:

```
> r10 <- Random.fuzzy.kmeans.validity(M, dim=JL.predict.dim(50,0.2),
                                     pmethod = "PMO", c = 10, n = 20);
> r10$overall.validity
[1] 0.573125
> r10$validity
```

```
[1] 0.5972222 0.0500000 0.7366667 0.7411111 0.5500000 0.6300000
[7] 0.6300000 0.6500000
```

7.4 Estimate of the reliability of clusters generated by the PAM (Prediction Around Medoids) clustering algorithm

Finally we perform clustering analysis with the PAM algorithm, using the same data set we used with the fuzzy-k-means algorithm:

```
> p2 <- Random.PAM.validity(M, dim=JL.predict.dim(50,0.2),
                            pmethod = "PMO", c = 2, n = 20);
> p2$overall.validity
[1] 0.8846154
> p2$validity
[1] 0.7692308 1.0000000
> p2$orig.cluster
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 [21] 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
[[2]]
 [1] 41 42 43 44 45 46 47 48 49 50
```

With two clusters in this case the overall validity is lower, as the first big cluster is considered less reliable. This fact suggest that the PAM clustering algorithm seems to be better suited for this data set. These results show also that the validity indices depend on the clustering algorithm used: indeed the computation of the similarity matrix that is used to compute the stability indices is performed through multiple clusterings on the randomly projected data, and hence depends on the applied clustering algorithm.

With partitions composed by clusters, we obtain, as expected, very reliable clusters:

```
> p3 <- Random.PAM.validity(M, dim=JL.predict.dim(50,0.2),
                            pmethod = "PMO", c = 3, n = 20);
> p3$overall.validity
[1] 1
> p3$validity
[1] 1 1 1
> p3$orig.cluster
```

```
[[1]]
 [1] 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 [20] 20 21 22 23 24 25 26 27 28 29 30
[[2]]
 [1] 31 32 33 34 35 36 37 38 39 40
[[3]]
 [1] 41 42 43 44 45 46 47 48 49 50
```

And, of, course also with 5 clusters:

```
> p5 <- Random.PAM.validity(M, dim=JL.predict.dim(50,0.2),
                            pmethod = "PMO", c = 5, n = 20);
> p5$overall.validity
[1] 1
> p5$validity
[1] 1 1 1 1 1
```

With 4 or e.g. 8 clusters the reliability of the obtained clusters is lower, as the "true" clusters are fragmented:

```
> p4 <- Random.PAM.validity(M, dim=JL.predict.dim(50,0.2),
                           pmethod = "PMO", c = 4, n = 20);
> p4$overall.validity
[1] 0.9047581
> p4$validity
[1] 0.7099415 0.9090909 1.0000000 1.0000000

> p8 <- Random.PAM.validity(M, dim=JL.predict.dim(50,0.2),
                            pmethod = "PMO", c = 8, n = 20);
> p8$overall.validity
[1] 0.6332639
> p8$validity
[1] 0.7450000 0.7900000 0.6633333 0.7000000 0.7166667
[6] 0.6333333 0.0500000 0.7677778
```

8 Estimate of the reliability of clusters generated by a generic clustering algorithm

In this section we see how to use the functions provided by *clusterv* to estimate the reliability of generic clustering algorithms.

It is worth noting that even if the proposed stability indices may be applied to any clustering algorithm, their computation is based on clusterings applied to subspaces of euclidean spaces, where the condition of low distortion is guaranteed if euclidean distances are used.

With this limitation in mind, the logical steps of the procedure you need to compute the stability indices for a given generic *Clust* algorithm applied to a d-dimensional data set D are:

1. Compute the clustering on the d-dimensional data set D using *Clust*
2. Generate multiple randomly projected data using a suitable random projection (with a given distortion)
3. Perform multiple clusterings using the projected data
4. Compute the similarity matrix using the previously computed multiple clusterings
5. Compute the validity indices for the individual clusters obtained at point 1.
6. Compute the overall validity index for the clustering
7. Compute the AC indices for each example
8. Save the results in a suitable way

For each logical step, the *clusterv* functions to be used are listed below:

1. Use `Transform.vector.to.list` if *Clust* returns a vector to represent the clustering
2. `Plus.Minus.One.random.projection`, `Achlioptas.random.projection`, `norm.random.projection`, `random.subspace`
3. Simply iterate the application of *Clust* on the data generated at step 2.
4. `Do.similarity.matrix`, `Do.similarity.matrix.partition`
5. `Validity.indices`, `Cluster.validity`, `Cluster.validity.from.similarity`
6. Simply average the validity indices of the individual clusters or use `Cluster.validity`, `Cluster.validity.from.similarity`
7. `AC.index`, `Cluster.validity`, `Cluster.validity.from.similarity`
8. —

From an implementative point of view, *Clust* needs to return the clustering represented as a vector **v** or a list **l**. More precisely the vector **v** should be an integer vector, whose indices represent the label of the examples, and the content the label (an integer) of the cluster to which the example belong (such as `clustering` component of the list `partition.object` of the *cluster* package).

The elements of list **l** must be vectors. Each vector represents a different cluster of the clustering; the elements of the vector are the (integer) labels of the examples (such as the `orig.cluster` component of the list returned by, e.g., the functions `Random.hclustering.validity` or `Random.PAM.validity` of the *clusterv* package).

Note that the functions implemented in *clusterv* need the clusterings implemented as list. Anyway if your clustering algorithm outputs a vector **v**, you can use the function `Transform.vector.to.list`. Consider, for instance, that the `kmeans` function of the package *stats* returns a vector for the computed k-means clustering. To convert the vector to a list:

```
> M <- generate.sample0(n=10, m=2, sigma=1, dim=500)
> # transforming a clustering vector obtained with kmeans to a list
> r<-kmeans(t(M), 3, 100);
> clustering.list.kmeans <- Transform.vector.to.list(r$cluster);
```

As an example about how to write a function to compute the stability indices for a generic clustering algorithm, consider the code of the function `Random.kmeans.validity` that computes the stability indices for the clusterings obtained with the k-means algorithm (note that the Step 1,2, etc refer to the logical steps listed above):

```
Random.kmeans.validity <- function(M, dim, pmethod="PMO", c=3,
             it.max=1000, n=50, scale=TRUE, seed=-1, AC=TRUE) {
  dim.Sim.M <- ncol(M);
 if (seed == -1)
    seed <- round(runif(1,1,10000));
 # Step 1. Computing the clusters in the original space
 r<-kmeans(t(M), c, iter.max = it.max);
 cl.orig <- Transform.vector.to.list(r$cluster);

 # Step 2. and 3. Perform multiple random projections and multiple
 #                clusterings on the resulting projected data
 cl <- Multiple.Random.kmeans (M=M, dim=dim, pmethod=pmethod,
            c=c, n=n, it.max=it.max, scale=scale, seed=seed);
```

```
 # Step 4. Compute the similarity matrix
 Sim.M <- Do.similarity.matrix(cl, dim.Sim.M);

 # Computing the list of validity measures
 # Step 5. Computing the validity indices vi
 c <- length(cl.orig);  # it corrects for empty sets
 vi <- Validity.indices(cl.orig, c, Sim.M);

 # Step 6. Computing the overall  validity of the clustering:
 ov.vi <- sum(vi)/c;

 # Step 7 and 8. Computing the AC indices and store
 #               the results in a list:
 if (AC == TRUE) {
   ac <- AC.index(cl.orig, c, Sim.M);
  res <- list (validity=vi, overall.validity=ov.vi,
               similarity.matrix=Sim.M, dimension=dim,
         cluster=cl, orig.cluster=cl.orig, AC=ac);
 }
 else
   res <- list (validity=vi, overall.validity=ov.vi,
               similarity.matrix=Sim.M, dimension=dim,
         cluster=cl, orig.cluster=cl.orig);
 return(res);
}
```

For the meaning of the input parameters of the above function, please, see the Reference manual. Finally the code of the function `Multiple.Random.kmeans` used for the Steps 2 and 3 inside the function `Random.kmeans.validity` is the following (see the reference manual for the meaning of the input parameters):

```
Multiple.Random.kmeans <- function(M, dim, pmethod="PMO",
            c=3, n=50, it.max=1000, scale=TRUE, seed=100) {
 set.seed(seed);
 cl <- list();
 for (i in 1:n) {
   # A. selection of the randomized map
   P.M <- switch(pmethod,
     RS = random.subspace(d=dim, M, scaling=scale),
     PMO = Plus.Minus.One.random.projection(d=dim, M, scaling=scale),
```

```
    Norm = norm.random.projection(d=dim, M, scaling=scale),
    Achlioptas = Achlioptas.random.projection(d=dim, M, scaling=scale),
    stop("Multiple.Random.kmeans: not supported random projection.", call.=FALSE));
  r<-kmeans(t(P.M), c, iter.max = it.max);
  cl[[i]] <- Transform.vector.to.list(r$cluster);
 }
 return(cl);
}
```

9 An applicative example to the analysis of DNA microarray data: analysis of cluster reliability in lung tumor patients

Here we present the results of an application of *clusterv* to the analysis of *lung tumor* patients, using a DNA microarray data composed by 203 histologically defined specimens: 186 lung tumors, subdivided in 127 lung adenocarcinoma (AD), 21 squamous cell lung adenocarcinoma (SQ), 20 pulmonary carcinoids (COID), 6 small-cell lung adenocarcinoma (SMCL) and 17 normal lung (NL) specimens [3]. From the 12600 original genes of the U95A Affymetrix oligonucleotide array 3312 passed the filter (genes with standard deviation units less than 50 have been excluded), according to the procedures described in [3] and then the gene expression levels have been normalized with respect to the mean and standard deviation. In this case also we implemented the pre-processing procedures with R scripts. We evaluated the reliability of the discovered clusters for both normalized and not normalized (with respect to the mean and standard deviation) data.

Bhattacharjee et al. discovered distinct subclasses of lung adenocarcinoma [3] using DNA microarray data. We applied our stability measures using *PMO* projections and the Ward's hierarchical clustering to analyze the reliability of the discovered subclasses.

The results summarized in Tab. 2 and Fig. 5 partially confirmed that the clusters defined by established histological classes [3] are quite reliable. At first, the overall stability indices suggest that pulmonary carcinoid tumors (COID) constitute a well-defined and separated cluster among the different subclasses of lung adenocarcinomas. Indeed the highest overall stability index is obtained with N=2 clusters; the first cluster that collect all the COID patients shows an individual stability index very close to 1. Moreover with N=3 clusters the

first COID cluster is highly supported by the **s** index (Tab. 2). Anyway also partitions characterized by larger number of clusters show relatively high values of the overall stability index, supporting the Bhattacharjee et al. thesis of distinct subclasses of lung adenocarcinoma. For instance, with N=4 clusters, the COID and normal lung (NL) subclasses are classified as reliable by the **s** index, the big second cluster characterized by several adenocarcinomas (AD) with Small-Cell-Lung-adenocarcinoma (SMCL) and some normal examples is scored as quite reliable (stablity index **s**=0.8168), while the fourth cluster that groups together adenocarcinoma and squamous cell lung adenocarcinomas (SQ) is scored as less reliable (**s**=0.7157) (Tab. 2 and Fig. 5). With N=8 the first two subclasses of COID patients are highly reliable, as well as the sixth cluster (normal lung). Interestingly enough, the cluster 3,4,5 represents three distinct subclasses inside adenocarcinoma patients, with a relatively high individual cluster stability (Tab. 2, N=8). Cluster 7 also represents another cluster of adenocarcinomas with also SQ and SMCL specimens inside, even if its individual stability index is quite smaller (**s**=0.7692). These results partially confirm the hypothesis of distinct subclasses among lung adenocarcinoma [3]. Anyway the stability indices show also that the subclasses are not so clearly delineated: these facts show that the results of clustering algorithms should be considered with caution, especially when complex and noisy data (such as DNA microarray data usually are) are analyzed. A stability analysis using other clustering algorithms (using for instance the functions `Random.kmeans.validity`, `Random.fuzzy.kmeans.validity`, `Random.PAM.validity` of the *clusterv* package) could get more insights into this problem.

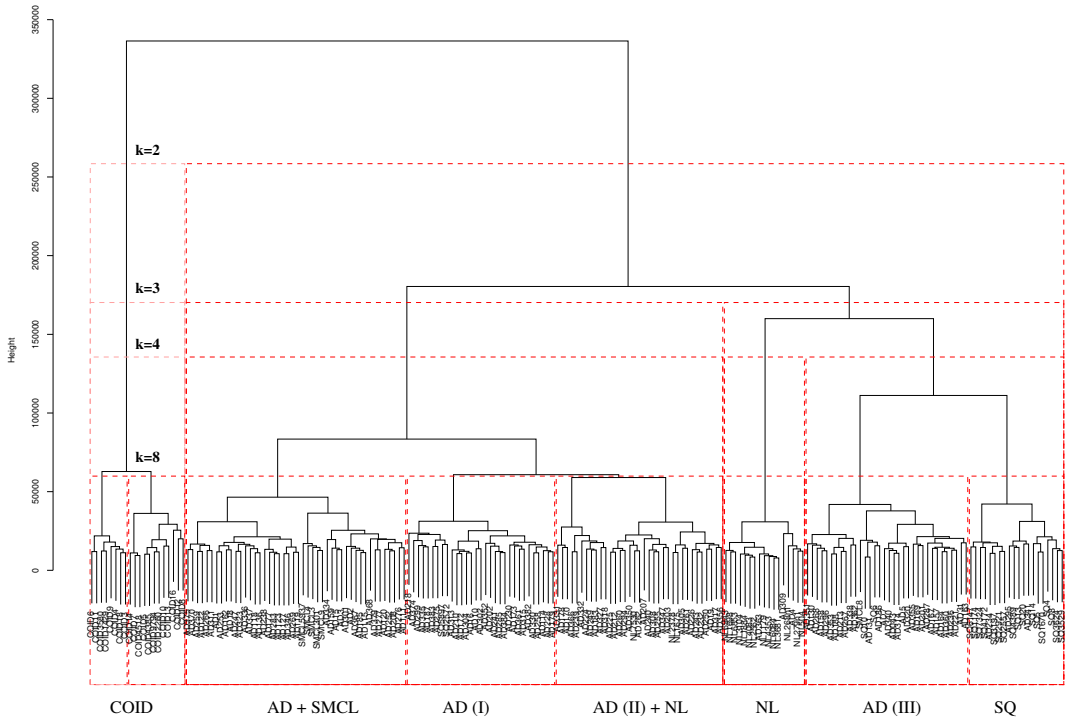

Figure 5: Hierarchical clustering of *Lung tumor* examples (Ward method). Gray dotted lines cut the dendrogram such that exactly k clusters are produced, for k=2,3,4,8. Considering 8 clusters, the first two refers two pulmonary carcinoids patients (COID), the third to a group of lung adenocarcinoma together with small-cell lung adenocarcinoma patients (SMCL), the fourth to a first group of lung adenocarcinoma patients (AD I), the fifth to a second group of adenocarcinoma patients with 3 normal patients (AD II + NL), the sixth to normal (NL) patients, the seventh to a third group of adenocarcinoma patients (AD III) and the last to squamous cell lung adenocarcinomas (SQ). See Table 2 for the corresponding stability indices.

Table 2: *Lung Tumor*: Estimate of cluster stability

N.	**Overall stability index S**				
	eps=0.5	eps=0.4	eps=0.3	eps=0.2	eps=0.1
2	0.9017	0.9376	0.9705	0.9708	0.9883
3	0.7550	0.7723	0.7964	0.8057	0.8611
4	0.7381	0.7571	0.7928	0.8074	0.8698
5	0.7198	0.6994	0.7497	0.7815	0.8294
6	0.6706	0.6797	0.7264	0.7602	0.8273
7	0.6777	0.6982	0.7381	0.7750	0.8225
8	0.6330	0.6575	0.7030	0.7460	0.8096
9	0.6123	0.6355	0.6870	0.7282	0.8098
10	0.5970	0.6304	0.6850	0.7336	0.8105
20	0.5769	0.6271	0.6700	0.7346	0.8056

N.	**Cl.**	**Individual stability index s**				
		eps=0.5	eps=0.4	eps=0.3	eps=0.2	eps=0.1
2	1	0.9185	0.9292	0.9684	0.9724	0.9940
	2	0.8849	0.9459	0.9726	0.9692	0.9826
3	1	0.9034	0.9236	0.9624	0.9723	0.9940
	2	0.7025	0.7455	0.7600	0.7401	0.8475
	3	0.6592	0.6479	0.6667	0.7047	0.7420
4	1	0.8976	0.9236	0.9624	0.9723	0.9940
	2	0.6371	0.6857	0.7237	0.7119	0.8448
	3	0.7997	0.7861	0.8469	0.9020	0.9247
	4	0.6180	0.6331	0.6384	0.6435	0.7157
5	1	0.8875	0.9236	0.9624	0.9723	0.9940
	2	0.5690	0.5597	0.6132	0.6664	0.6992
	3	0.7630	0.7211	0.7930	0.8588	0.9107
	4	0.7637	0.6811	0.7267	0.6971	0.7341
	5	0.6157	0.6116	0.6532	0.7132	0.8093
6	1	0.8495	0.9144	0.9624	0.9723	0.9940
	2	0.7804	0.7993	0.8321	0.8891	0.8861
	3	0.4322	0.4648	0.5111	0.5181	0.6642
	4	0.7270	0.7150	0.7767	0.8261	0.9107
	5	0.6690	0.6275	0.6675	0.6772	0.7236
	6	0.5655	0.5573	0.6086	0.6786	0.7850
8	1	1.0000	1.0000	1.0000	1.0000	1.0000
	2	0.7884	0.8430	0.9345	0.9509	0.9900
	3	0.5514	0.6352	0.6965	0.7776	0.8360
	4	0.4865	0.4828	0.5510	0.6283	0.7813
	5	0.3904	0.4505	0.4588	0.5073	0.5258
	6	0.7064	0.7144	0.7755	0.8255	0.9107
	7	0.5949	0.5957	0.6078	0.6132	0.6484
	8	0.5455	0.5385	0.5998	0.6649	0.7847

References

[1] Ash A Alizadeh, Michael B Eisen, R Eric Davis, Chi Ma, Izidore S Lossos, Andreas Rosenwald, Jennifer C Boldrick, Hajeer Sabet, Truc Tran, Xin Yu, et al. Distinct types of diffuse large b-cell lymphoma identified by gene expression profiling. *Nature*, 403(6769):503–511, 2000.

[2] James C Bezdek. *Pattern recognition with fuzzy objective function algorithms*. Springer Science & Business Media, 2013.

[3] Arindam Bhattacharjee, William G Richards, Jane Staunton, Cheng Li, Stefano Monti, Priya Vasa, Christine Ladd, Javad Beheshti, Raphael Bueno, Michael Gillette, et al. Classification of human lung carcinomas by mrna expression profiling reveals distinct adenocarcinoma subclasses. *Proceedings of the National Academy of Sciences*, 98(24):13790–13795, 2001.

[4] Lars Dyrskjøt, Thomas Thykjaer, Mogens Kruhøffer, Jens Ledet Jensen, Niels Marcussen, Stephen Hamilton-Dutoit, Hans Wolf, and Torben F Ørntoft. Identifying distinct classes of bladder carcinoma using microarrays. *Nature genetics*, 33(1):90–96, 2003.

[5] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[6] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 1990.

[7] Benjamin King. Step-wise clustering procedures. *Journal of the American Statistical Association*, 62(317):86–101, 1967.

[8] Jacques Lapointe, Chunde Li, John P Higgins, Matt Van De Rijn, Eric Bair, Kelli Montgomery, Michelle Ferrari, Lars Egevad, Walter Rayford, Ulf Bergerheim, et al. Gene expression profiling identifies clinically relevant subtypes of prostate cancer. *Proceedings of the National Academy of Sciences*, 101(3):811–816, 2004.

[9] Margaret A Shipp, Ken N Ross, Pablo Tamayo, Andrew P Weng, Jeffery L Kutok, Ricardo CT Aguiar, Michelle Gaasenbeek, Michael Angelo, Michael Reich, Geraldine S Pinkus, et al. Diffuse large b-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning. *Nature medicine*, 8(1):68–74, 2002.