How to use the `vegclust` package (ver. 1.6.0)

Miquel De Cáceres[1]

[1]Centre Tecnològic Forestal de Catalunya. Ctra. St. Llorenç de Morunys km 2, 25280, Solsona, Catalonia, Spain

February 18, 2013

Contents

1 Introduction

1.1 What is this tutorial about?

Classification of vegetation plot records involves different activities, including the design an appropriate vegetation survey, the use of a classification method to group vegetation observations and the characterization, validation and naming of the resulting vegetation groups. In this tutorial we focus in only one of this steps, namely to group vegetation observations, and we show how to conduct it with the help of the R package `vegclust`. Before starting our examples we need to load the `vegclust` package, which also loads the required R packages `vegan` and `permute`:

```
> library(vegclust)
```

1.2 Example vegetation data

In order to illustrate the functions in `vegclust` we will use a small wetland vegetation data set, consisting of 41 sites and 33 species and published by Bowman and Wilson [1986]. The data is included with the `vegclust` package:

```
> data(wetland)
> dim(wetland)

[1] 41 33
```

For a number of reasons we will not detail here, the Euclidean distance is not an appropriate index to measure the resemblance in species composition between vegetation plot records. Therefore, we transform our community data using the chord transformation [Legendre and Gallagher, 2001], which divides each value by the norm of the row vector of the corresponding site:

```
> wetlandchord = decostand(wetland,"normalize")
```

Function `decostand` is provided within the `vegan` package. The Euclidean distance on the transformed data is equal to the chord distance [Orlóci, 1967] with respect to the raw community data:

```
> dchord = dist(wetlandchord)
```

In some of our examples we will use `wetlandchord` or `dchord` indistinctively, because package `vegclust` allows conducting classification of vegetation from either a site-by-species data table or from a site-by-site dissimilarity matrix. In the next section we briefly explain the bases of the classification methods that are provided in the `vegclust` package. We later show how to run those methods with functions in the package.

2 Clustering methods in `vegclust`

2.1 Resemblance space

Generally speaking, the goal of clustering is to derive c 'natural' classes or clusters from a set of n unlabelled objects. Those objects inside a 'natural' cluster show a certain degree of closeness or similarity and the cluster itself shows a certain degree of isolation from other clusters. In classification of vegetation the n 'objects' to be grouped are plant communities (i.e. plot records or relevés) and the goal is to define vegetation types.

When speaking of 'closeness' or 'similarity', we implicitly assume there is a way to assess the resemblance between the objects to be grouped. This can be obtained by describing our objects using a set of p features (these are normally species in the case of vegetation) and specifying a resemblance measure (e.g. distance or dissimilarity). Let $\mathbf{X} = [x_{js}]$ be a site-by-species data table of dimensions $n \times p$, where x_{js} is the abundance of species s in site j, and let d be an appropriate dissimilarity or distance measure.

Another way to formalize the resemblance between objects is to directly give similarity or dissimilarity between pairs of objects in a symmetric resemblance matrix. Let $\mathbf{D} = [d_{ij}]$ a symmetric dissimilarity matrix of dimensions $n \times n$, where $d_{ij} = d(\mathbf{x}_i, \mathbf{x}_j)$ is the dissimilarity between objects i and j. In classification of vegetation $d_{ij} = d(\mathbf{x}_i, \mathbf{x}_j)$ will normally represent the compositional dissimilarity between plant communities i and j.

Regardless of whether we use $\mathbf{X}$ or $\mathbf{D}$, we will speak here of *resemblance space* and our objects (plant communities) will be formally represented as points in this space. Although we do not cover this topic in detail here, the reader should be aware that building an appropriate resemblance space is crucial in vegetation classification.

2.2 Prototype-based clustering

Prototype-based clustering methods assume that the properties of objects in a cluster can be represented using a cluster *prototype*, which is formalized as a point in the resemblance space. The problem is thus to find c prototypes and assign the n objects according to their closeness to those prototypes, such that the resulting clusters are compact and isolated one from another.

All the clustering methods discussed here follow an *alternate optimization* scheme, meaning that one group of parameters (e.g. the membership matrix) is optimized holding the other group (e.g. the cluster prototypes) fixed and vice versa. Assuming an initial cluster configuration, this is achieved by iterating the following three steps:

(S.1) Determine the prototype of each cluster i.

(S.2) Calculate e_{ij}, the distance from each object j to the prototype of each cluster i.

(S.3) Calculate u_{ij}, the (hard or fuzzy) membership of each object j to each cluster i (i.e. re-assign all objects into clusters).

Each iteration improves the so called *objective function* of the clustering method. The alternate optimization algorithm stops when there are no further changes in object memberships. More technically, it stops if the maximum difference in object membership values of the last two iterations does not exceed a user-specified threshold. The starting configuration of clusters is a critical issue in this kind of classification methods, because the iterations can led the alternate optimization algorithm to get stuck in a suboptimal value of the objective function. For this reason, several executions of the algorithm are usually tried, each one using a different starting configuration.

2.3 Clustering models

There are several prototype-based clustering models in the `vegclust` package. All of them following the iterative algorithm presented above. The differences between clustering models arise due to differences in the specific implementation of each step, resulting from different assumptions of how clusters should be defined. The clustering models can be divided according to their properties into:

a) Whether object memberships are **fuzzy** or **hard** (crisp).

b) Whether cluster prototypes are **centroids** or **medoids**.

c) How to deal with outlier objects, which gives three kinds of models: **partitive**, **noise clustering** or **possibilistic clustering**.

In the following subsections we describe the implications of each of these decisions.

2.3.1 Hard (crisp) or fuzzy memberships

With the kind of classification methods described here, it is costumary to represent the classification of objects into groups using a $c \times n$ membership

matrix, $\mathbf{U} = [u_{ij}]$, where u_{ij} is the degree of membership of object j in cluster i. The classification is said to be *hard* when the u_{ij} values are either 0 (object j DOES NOT belong to cluster i) or 1 (object j DOES belong to cluster i). In contrast, if the classification is *fuzzy* the membership of its objects is expressed through a degree of membership bounded between 0 (i.e. the object does not belong to the set at all) and 1 (i.e. the object belongs completely to the set).

The advantages of using fuzzy set theory in vegetation are that it acknowledges the individualistic concept of vegetation [Moravec, 1989], and therefore avoids assuming that relevés are unequivocal representatives of a type without admixture of any other types [Dale, 1995].

2.3.2 Centroids or medoids

The *centroid* of a cluster i that contains n_i objects is a vector $\mathbf{c}_i$ whose coordinates are the average, in each dimension s, of the coordinates of the n_i objects belonging to the cluster. In vector notation:

$$\mathbf{c}_i = \frac{\sum_{j=1}^{n_i} \mathbf{x}_j}{n_i} \tag{1}$$

The centroid has the property that minimizes the sum of squared Euclidean distances between itself and each point belonging to the cluster. Equation 1 can be generalized to the case of a fuzzy cluster by weighting the coordinates each object by its degree of membership to the cluster, u_{ij}:

$$\mathbf{c}_i = \frac{\sum_{j=1}^{n} u_{ij}^m \mathbf{x}_j}{\sum_{j=1}^{n} u_{ij}^m} \tag{2}$$

In equation 2, $m > 0$ is the *fuzziness exponent*, which is used to modulate the influence of fuzzy memberships in the calculation of the centroids. If m is very large only the objects whose membership is close to 1 will have an influence in the centroid. On the contrary, if m is small (i.e. close to 0) then all n objects will have influence equally the centroid and it will approach the overall data center.

A *medoid* of a cluster i that contains n_i objects is defined as the object, chosen among the n_i objects, for which the sum of dissimilarities to all the n_i objects is minimal i.e. it is a most centrally located point in the cluster. Formally, the medoid is the object k for which:

$$\sum_{j=1}^{n_i} d(\mathbf{x}_k, \mathbf{x}_j) \tag{3}$$

is minimal. When using fuzzy logic, the medoid of cluster i is defined as the object k (among all n objects) that minimizes:

$$\sum_{j=1}^{n} u_{ij}^m d(\mathbf{x}_k, \mathbf{x}_j) \tag{4}$$

Note that, because the medoid is a point chosen among the n input objects, we do not need to calculate its coordinates (although see explanation below for centroids), moreover, the distance between a medoid and the other objects (step S.2 in the alternate optimization algorithm) will be readily available from the beginning, so it does not need to be computed. All this makes dealing with medoids computationally much faster than dealing with centroids.

2.3.3 Partitive clustering

A clustering method is called *partitive* if object memberships (crisp or fuzzy) are constrained to sum to one for each object:

$$\sum_{i=1}^{c} u_{ij} = 1 \tag{5}$$

This constrain is usually referred to as the *partition restriction*. It ensures that all objects will be classified as either belonging to a single cluster or dividing their membership among different clusters. No objects will be left unclassified.

K-means (KM, also known as hard c-means) [MacQueen, 1967] and Fuzzy c-means (FCM) [Bezdek, 1981] are two, centroid-based, partitive clustering algorithms widely used in many unsupervised pattern recognition applications. The main difference between the two methods is that in KM every object belongs to a single cluster (i.e clusters are 'hard') whereas in FCM the memberships are fuzzy and a given object may have some degree of membership for more than one cluster.

Cluster memberships (step S.3) are determined in KM simply by *assigning each object to the cluster whose center is closest*. In the case of FCM, fuzzy memberships are calculated using the following formula:

$$u_{ij} = \frac{1}{\sum_{l=1}^{c} (e_{ij}/e_{lj})^{2/(m-1)}} \tag{6}$$

As said above, $m > 1$ is the fuzziness coefficient. The smaller the value of m, the closer to a hard partition will be the result. If m is set too high and the data is noisy the resulting partition may be completely fuzzy (i.e. where $u_{ij} = 1/c$ for all objects and clusters) and therefore uninformative.

As indicated above, KM and FCM are centroid-based, meaning they use equations 1 and 2, respectively, for step S.1. The corresponding medoid-based methods are Hard C-medoids and Fuzzy C-medoids, which instead use equations 3 and 4, respectively [Krishnapuram et al., 1999].

2.3.4 Noise clustering

The noise clustering (NC) method [Dave, 1991] is an attempt to make the FCM method more robust to the effect of outliers. The rationale underlying

NC is the following: if an object is an outlier, this means that it lies far from all cluster prototypes and, therefore, it should have low membership values to all clusters. In order to achieve these low memberships, the NC considers an additional cluster, called *Noise*. This class is represented by an imaginary 'prototype' that lies at a constant distance δ from all the data points. Even if such a prototype does not really exist, the effect of including the Noise class is that it 'captures' objects that lie farther than δ from all the c 'good' prototypes. The NC membership function (step S.3) for the 'good' clusters is:

$$u_{ij} = \frac{1}{(e_{ij}/\delta)^{2/(m-1)} + \sum_{l=1}^{c} (e_{ij}/e_{lj})^{2/(m-1)}} \tag{7}$$

whereas the fuzzy membership to the Noise class, u_{Nj}, is one minus the sum of memberships to the good clusters.

$$u_{Nj} = 1 - \sum_{i=1}^{c} u_{ij} \tag{8}$$

Outlier objects have small membership values to the c good clusters because the first term in the denominator of equation 7 is large. The smaller the δ, the higher will be memberships to the Noise class. In contrast, large values of δ make NC equivalent to FCM. The fuzziness exponent m has in NC the same interpretation as in FCM. Including the Noise class has the effect of relaxing the partition restriction. In NC, the partition restriction is fulfilled when considering all c good clusters and the Noise class.

Note that, like FCM and KM, we can define a 'hard' counterpart of the (fuzzy) noise clustering method. Indeed, the hard noise clustering (HNC) method differs from the fuzzy one in that memberships are not fuzzy. Like KM, its membership function can be described verbally. One assigns the object to the noise class if the distances to all cluster centers is larger than δ. Otherwise, one assigns the object to the cluster whose center is closest, as in KM.

The noise clustering method was originally defined with centroids as prototypes. However, either hard or fuzzy noise clustering can be applied to medoids instead of centroids. While we have not found any references about this possibility, the corresponding algorithms could be named 'hard noise clustering with medoids' (HNCdd) and '(fuzzy) noise clustering with medoids' (NCdd).

2.3.5 Possibilistic clustering

Possibilistic C-means [Krishnapuram and Keller, 1993, 1996] is another modification of FCM seeking increased cluster robustness. The partition restriction is eliminated in PCM, which produces c independent fuzzy clusters,

each corresponding to a dense region of points. Whereas the FCM and NC membership functions compare the distance from the object to the cluster of interest, e_{ij}, with the distances to the remaining centres (and to the noise class in the case of NC), in PCM the membership value for a given object to a cluster does not depend on the distances to the remaining cluster centres. Instead, the distance to the cluster of interest is compared to a reference distance (η_i):

$$u_{ij} = \frac{1}{1 + (e_{ij}^2/\eta_i)^{1/(m-1)}} \tag{9}$$

The reference distance is a parameter that must be provided for each cluster. All objects whose distance to the cluster center is smaller than η_i will obtain a membership value higher than 0.5.

Cluster repulsion is eliminated in PCM, which increases cluster mobility. Good estimation of η_i is crucial for the success of the PCM method [De Cáceres et al., 2006]. The high cluster mobility resulting from the inadequate initialization of η_i can lead to a miss of cluster structures, even with the correct partition as initial starting configuration. A single PCM run can be regarded as c independent runs of NC, each looking for a single cluster and where $\delta_i^2 = \eta_i$ [Dave and Krishnapuram, 1997]. In vegetation data plant communities with intermediate characteristics are frequent. This fact makes the PCM method impractical for classification of vegetation, because without cluster repulsion PCM clusters are frequently highly mobile and converge to the same cluster, leaving large parts of the data unassigned [De Cáceres et al., 2010].

2.4 Dissimilarity-based duals

All the clustering methods presented above can be executed on a resemblance space described using either **X** or **D**. The latter case avoids dealing with the coordinates of prototypes explicitly.

2.4.1 Medoid-based clustering and dissimilarity matrices

Because medoids are selected among the objects to be classified, it is obvious that the distance to the cluster prototypes, e_{ij}, can be drawn from a symmetric matrix of pairwise distances between objects calculated before starting the alternate optimization algorithm. In other words, one can conduct medoid-based clustering on a site-by-site distance matrix instead of using a site-by-species rectangular matrix. Moreover, one can straightforwardly skip the use of Euclidean distance and use a dissimilarity measure more appropriate for ecological data.

2.4.2 Centroid-based clustering and dissimilarity matrices

When dealing with centroids, it may seem unavoidable to calculate centroid coordinates (step S.1) prior to calculating the (squared) Euclidean distances to cluster centers e_{ij} (step S.2):

$$e_{ij}^2 = \|\mathbf{c}_i - \mathbf{x}_j\|^2 \tag{10}$$

However, there is a mathematical trick that avoids calculating the coordinates $\mathbf{c}_i$ explictly. Let $\mathbf{D}$ be the matrix of Euclidean distances between pairs of objects. We can obtain e_{ij}^2 as follows:

$$e_{ij}^2 = \frac{1}{n_i}\sum_{l=1}^{n_i} d_{lj}^2 - \frac{1}{2n_i^2}\sum_{k=1}^{n_i}\sum_{l=1}^{n_i} d_{lk}^2 \tag{11}$$

The left part of this equation is a sum of squared distances from the target object to all the other objects in the cluster. The right part of the equation is an average of squared distances between objects in the cluster. Equation 11 can be generalized to the case of a fuzzy cluster:

$$e_{ij}^2 = \frac{1}{\sum_{k=1}^{n} u_{ik}^m}\sum_{l=1}^{n_i} d_{lj}^2 - \frac{1}{2\left(\sum_{k=1}^{n} u_{ik}^m\right)^2}\sum_{k=1}^{n}\sum_{l=1}^{n} u_{ik}^m u_{il}^m d_{lk}^2 \tag{12}$$

Equations 11 and 12 allow determining the Euclidean distance to a centroid without calculating its coordinates. Therefore, they allow conducting steps S.1 and S.2 in a single step. In other words, distance-based duals exist for centroid-based clustering methods when Euclidean distances are used in the resemblance space [Hathaway et al., 1989, 1996].

If we transform the original data in order to emulate a distance like the chord, then the duality holds, although centroids are defined in the transformed space. What happens if the values in $\mathbf{D}$ were not calculated using the Euclidean distance? Equations 11 and 12 are also valid for arbitrary dissimilarity measures, although there are important details to be remembered. These equations assume that the resemblance space is Euclidean (i.e. does not produce negative eigenvalues in principal coordinates analysis) and that centroids are appropriate prototypes for clusters. If the dissimilarities do not have the Euclidean property some oddities may arise [Hathaway and Bezdek, 1994]. For example, it is possible to obtain negative e_{ij}^2 values, specially for groups of small size. In practice, however, when these negative distances occur they can be reset to zero [De Cáceres et al., 2006].

3 Managing vegetation classifications

3.1 Creating classifications: `vegclust` and `vegclustdist`

Functions `vegclust` and `vegclustdist` allow defining vegetation types from a set of unlabelled vegetation observations (i.e. relevés or plot records)

using any of the clustering models explained in the previous section. While `vegclust` needs a rectangular site-by-species matrix, `vegclustdist` needs a symmetric site-by-site dissimilarity matrix.

3.1.1 The K-means model

The following piece of code produces a classification of our example data set into three groups using the K-means clustering model:

```
> wetland.km = vegclust(x = wetlandchord, mobileCenters=3,
+                       method="KM", nstart=20)
```

The result is an object of class 'vegclust', in fact a list with several components (method, parameters, prototypes, objective function, etc.):

```
> names(wetland.km)
```

```
 [1] "mode"          "method"        "m"
 [4] "dnoise"        "eta"           "memb"
 [7] "mobileCenters" "fixedCenters"  "dist2clusters"
[10] "withinss"      "size"          "functional"
[13] "iter"
```

One of the most important components is the membership matrix $\mathbf{U}$, which we show transposed here:

```
> t(wetland.km$memb)
```

```
   5 8 13 4 17 3 9 21 16 14 2 15 1 7 10 40 23 25 22 20 6 18 12 39
M1 1 1  1 1  1 1 1  1  1  1 1  1 1 1  1  0  0  0  0  0 0  0  0  0
M2 0 0  0 0  0 0 0  0  0  0 0  0 0 0  0  1  1  1  1  1 1  1  1  1
M3 0 0  0 0  0 0 0  0  0  0 0  0 0 0  0  0  0  0  0  0 0  0  0  0
   19 11 30 34 28 31 26 29 33 24 36 37 41 27 32 35 38
M1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1
M2  1  1  0  0  0  0  1  1  1  0  0  0  0  0  0  0  0
M3  0  0  1  1  1  1  0  0  0  1  1  1  1  1  1  0  0
```

Another important component is the matrix containing the coordinates of cluster centroids (i.e. vectors $\mathbf{c}_i$ for each cluster):

```
> round(wetland.km$mobileCenters, dig=3)
```

```
   Abefic Merhed Alyvag Pancam Abemos Melcor Ludoct Eupvac Echpas
M1  0.000  0.000  0.000  0.027  0.000  0.062  0.000  0.039  0.048
M2  0.000  0.000  0.047  0.076  0.016  0.153  0.000  0.050  0.066
M3  0.018  0.068  0.000  0.356  0.015  0.275  0.019  0.205  0.102
   Passcr Poa2 Carhal Dendio Casobt Aesind Cyprot Ipocop Cynarc
```

```
M1  0.000 0.00  0.037  0.016  0.016  0.000  0.000  0.000  0.000
M2  0.000 0.01  0.069  0.000  0.000  0.075  0.081  0.013  0.013
M3  0.026 0.00  0.000  0.000  0.000  0.019  0.176  0.040  0.432
   Walind Sessp. Phynod Echell Helind Ipoaqu Orysp. Elesp. Psespi
M1  0.068  0.144  0.000  0.049  0.125  0.055  0.649  0.154  0.192
M2  0.064  0.104  0.715  0.119  0.172  0.048  0.172  0.138  0.027
M3  0.078  0.157  0.064  0.112  0.019  0.037  0.040  0.000  0.000
   Ludads Polatt  Poa1 Helcri Physp. Goopur
M1  0.204  0.097 0.000  0.000  0.012  0.012
M2  0.000  0.000 0.013  0.013  0.000  0.000
M3  0.000  0.000 0.000  0.000  0.000  0.000
```

The same classification exercise can be conducted from the matrix of chord distances between objects if we use function `vegclustdist` instead of `vegclust`:

```
> wetland.kmdist = vegclustdist(x = dchord, mobileMemb=3,
+                               method="KM", nstart = 20)
> names(wetland.kmdist)

 [1] "mode"          "method"        "m"
 [4] "dnoise"        "eta"           "memb"
 [7] "mobileCenters" "fixedCenters"  "dist2clusters"
[10] "withinss"      "size"          "functional"
```

Note the different way to specify the number of clusters. In the case of `vegclustdist` we do not obtain cluster centroids, because they cannot be calculated explicitly:

```
> wetland.kmdist$mobileCenters

NULL
```

But we do obtain cluster memberships:

```
> t(wetland.kmdist$memb)

   5 8 13 4 17 3 9 21 16 14 2 15 1 7 10 40 23 25 22 20 6 18 12 39
M1 0 0  0 0  0 0 0  0  0  0 0  0 0 0  0  0  0  0  0  0 0  0  0  0
M2 1 1  1 1  1 1 1  1  1  1 1  1 1 1  1  0  0  0  0  0 0  0  0  0
M3 0 0  0 0  0 0 0  0  0  0 0  0 0 0  0  1  1  1  1  1 1  1  1  1
   19 11 30 34 28 31 26 29 33 24 36 37 41 27 32 35 38
M1  0  0  1  1  1  1  0  0  0  1  1  1  1  1  1  0  0
M2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1
M3  1  1  0  0  0  0  1  1  1  0  0  0  0  0  0  0  0
```

Because we used the same resemblance space (although in different form) in our examples, both `vegclust` and `vegclustdist` should give the same result provided the algorithm has not been stuck in a relative optimum of the objective function. Although both functions return objects of class 'vegclust', we can identify whether calculations were done from dissimilarities or from rectangular matrices by inspecting the 'mode' element of the list:

```
> wetland.km$mode

[1] "raw"

> wetland.kmdist$mode

[1] "dist"
```

In the following subsections we run `vegclust` using other clustering models, but the same examples could be made using `vegclustdist`.

3.1.2 The Fuzzy C-means model

Let us inspect the distance of each object to each cluster centroid:

```
> round(t(wetland.km$dist2clusters), dig=2)

      5    8   13    4   17    3    9   21   16   14    2   15
M1 0.54 0.68 0.44 0.45 0.64 0.76 0.95 0.50 0.64 0.46 0.38 0.57
M2 1.05 1.10 1.09 1.14 1.11 1.22 1.23 1.09 1.09 1.10 1.12 1.19
M3 1.12 1.12 1.14 1.17 1.07 1.22 1.22 1.20 1.17 1.16 1.17 1.21
      1    7   10   40   23   25   22   20    6   18   12   39
M1 0.47 0.72 0.87 1.11 0.92 0.86 0.73 1.11 1.15 1.19 1.16 1.12
M2 1.12 1.15 1.19 0.40 0.60 0.51 0.67 0.56 0.47 0.45 0.40 0.52
M3 1.21 1.18 1.23 1.09 1.07 1.12 1.03 1.15 1.06 1.12 1.12 1.06
     19   11   30   34   28   31   26   29   33   24   36   37
M1 0.97 1.13 1.20 1.22 1.19 1.20 1.22 1.17 1.22 1.20 1.18 1.17
M2 0.44 0.64 1.16 1.20 1.15 1.19 0.81 0.78 0.49 1.04 0.90 1.15
M3 1.07 1.07 0.86 0.66 0.77 0.69 1.00 0.92 1.00 0.55 0.62 0.55
     41   27   32   35   38
M1 0.99 1.22 1.17 0.80 0.68
M2 1.14 1.19 1.13 1.03 1.10
M3 0.56 0.78 0.73 0.95 0.99
```

For many objects the distance to the cluster where they have been assigned is much smaller than the distance to other clusters. However, for some objects (such as '22', '29' or '35') the distance to the closest cluster center does not differ much from the distance to second closest one. Are those latter objects well assigned? Should these objects have intermediate degrees

of membership instead of picking one cluster arbitrarily? The Fuzzy C-means cluster model allows obtaining fuzzy memberships instead of crisp memberships:

```
> wetland.fcm = vegclust(x = wetlandchord, mobileCenters=3,
+                        method="FCM", m=1.2, nstart=20)
> round(t(wetland.fcm$memb), dig=3)
```

```
       5     8 13 4    17     3     9 21    16 14 2    15 1     7
M1 0.001 0.007  0 0 0.004 0.009 0.070  0 0.004  0 0 0.001 0 0.011
M2 0.001 0.007  0 0 0.007 0.011 0.086  0 0.003  0 0 0.001 0 0.009
M3 0.998 0.985  1 1 0.990 0.981 0.844  1 0.993  1 1 0.999 1 0.981
      10 40    23    25    22    20 6 18 12    39 19    11    30
M1 0.044  1 0.980 0.992 0.581 0.998 1  1  1 0.999  1 0.992 0.040
M2 0.036  0 0.004 0.001 0.014 0.001 0  0  0 0.001  0 0.005 0.928
M3 0.920  0 0.016 0.007 0.405 0.001 0  0  0 0.000  0 0.003 0.031
      34    28    31    26    29    33    24    36    37    41
M1 0.002 0.016 0.003 0.841 0.742 0.999 0.002 0.020 0.001 0.001
M2 0.996 0.971 0.993 0.141 0.241 0.001 0.998 0.978 0.999 0.995
M3 0.002 0.013 0.003 0.017 0.017 0.000 0.000 0.001 0.001 0.004
      27    32    35    38
M1 0.015 0.011 0.057 0.006
M2 0.973 0.980 0.160 0.024
M3 0.012 0.008 0.783 0.970
```

A comparison of these memberships with the distance to the clusters shown before will reveal that intermediate objects obtain fuzzier membership values than other objects.

Although FCM is theoretically a better model than KM for vegetation classification, vegetation scientists are normally interested in crisp assignments. Function `defuzzify` allows turning a fuzzy membership matrix into a crisp one:

```
> groups = defuzzify(wetland.fcm)$cluster
> groups
```

```
   5    8   13    4   17    3    9   21   16   14    2   15    1
"M3" "M3" "M3" "M3" "M3" "M3" "M3" "M3" "M3" "M3" "M3" "M3" "M3"
   7   10   40   23   25   22   20    6   18   12   39   19   11
"M3" "M3" "M1" "M1" "M1" "M1" "M1" "M1" "M1" "M1" "M1" "M1" "M1"
  30   34   28   31   26   29   33   24   36   37   41   27   32
"M2" "M2" "M2" "M2" "M1" "M1" "M1" "M2" "M2" "M2" "M2" "M2" "M2"
  35   38
"M3" "M3"
```

```
> table(groups)
```

```
groups
M1 M2 M3
14 10 17
```

Another way of defuzzifying the membership matrix is by setting a threshold of minimum fuzzy membership:

```
> groups = defuzzify(wetland.fcm, method = "cut", alpha = 0.8)$cluster
> groups

   5    8   13    4   17    3    9   21   16   14    2   15    1
"M3" "M3" "M3" "M3" "M3" "M3" "M3" "M3" "M3" "M3" "M3" "M3" "M3"
   7   10   40   23   25   22   20    6   18   12   39   19   11
"M3" "M3" "M1" "M1" "M1"   NA "M1" "M1" "M1" "M1" "M1" "M1" "M1"
  30   34   28   31   26   29   33   24   36   37   41   27   32
"M2" "M2" "M2" "M2" "M1"   NA "M1" "M2" "M2" "M2" "M2" "M2" "M2"
  35   38
  NA "M3"

> table(groups, useNA = "always")

groups
  M1   M2   M3 <NA>
  12   10   16    3
```

With this second defuzzification approach intermediate objects are left unclassified (indicated as NA's). It is important to realize that FCM fuzzy membership values depend on the fuzziness exponent m. In fact, if we run FCM with a very large m value we will obtain uninformative results:

```
> wetland.fcm2 = vegclust(x = wetlandchord, mobileCenters=3,
+                         method="FCM", m=10, nstart=20)
> round(t(wetland.fcm2$memb), dig=3)

       5     8    13     4    17     3     9    21    16    14
M1 0.333 0.333 0.333 0.333 0.333 0.333 0.333 0.333 0.333 0.333
M2 0.334 0.334 0.335 0.335 0.334 0.334 0.334 0.334 0.334 0.334
M3 0.333 0.333 0.333 0.333 0.333 0.333 0.333 0.333 0.333 0.333
       2    15     1     7    10    40    23    25    22    20
M1 0.333 0.333 0.333 0.333 0.333 0.334 0.334 0.333 0.333 0.334
M2 0.335 0.334 0.334 0.334 0.334 0.333 0.333 0.333 0.334 0.333
M3 0.333 0.333 0.333 0.333 0.333 0.334 0.333 0.333 0.333 0.334
       6    18    12    39    19    11    30    34    28    31
M1 0.334 0.334 0.334 0.334 0.334 0.334 0.333 0.334 0.333 0.334
M2 0.332 0.332 0.332 0.332 0.333 0.333 0.333 0.333 0.333 0.333
```

```
M3 0.334 0.334 0.334 0.334 0.334 0.334 0.333 0.333 0.333 0.333
      26    29    33    24    36    37    41    27    32    35
M1 0.334 0.334 0.334 0.334 0.334 0.334 0.333 0.333 0.334 0.333
M2 0.333 0.333 0.332 0.333 0.333 0.333 0.333 0.333 0.333 0.334
M3 0.334 0.334 0.334 0.334 0.334 0.333 0.333 0.333 0.333 0.333
      38
M1 0.333
M2 0.334
M3 0.333
```

3.1.3 The Noise clustering model

In the previous two models, all objects were assigned, either completely to one cluster or dividing their membership among clusters (in other words, we stuck ourselves to the partition restriction). This may be appropriate in general, but it may cause problems if some plant assemblages describe rare species assemblages. These plant communities should better be classified as 'outliers' and should not influence the prototypes. In the noise clustering (NC) model we allow outlier objects to be excluded from the classification:

```
> wetland.nc = vegclust(x = wetlandchord, mobileCenters=3,
+                       method="NC", m=1.2, dnoise=0.8, nstart=20)
> round(t(wetland.nc$memb), dig=2)
```

```
      5    8 13 4   17    3    9 21   16   14 2   15 1    7   10
M1 0.00 0.00  0 0 0.00 0.01 0.01  0 0.00 0.00 0 0.00 0 0.01 0.01
M2 0.00 0.00  0 0 0.00 0.01 0.01  0 0.00 0.00 0 0.00 0 0.01 0.01
M3 0.99 0.85  1 1 0.93 0.41 0.08  1 0.86 0.99 1 0.93 1 0.49 0.12
N  0.01 0.14  0 0 0.07 0.58 0.90  0 0.13 0.01 0 0.07 0 0.49 0.86
   40   23   25   22   20 6 18 12   39 19   11   30   34   28   31
M1  1 0.93 0.98 0.39 0.98 1  1  1 0.99  1 0.93 0.01 0.00 0.02 0.00
M2  0 0.00 0.00 0.00 0.00 0  0  0 0.00  0 0.00 0.07 0.98 0.08 0.98
M3  0 0.01 0.01 0.50 0.00 0  0  0 0.00  0 0.00 0.01 0.00 0.01 0.00
N   0 0.06 0.01 0.11 0.02 0  0  0 0.01  0 0.06 0.90 0.02 0.89 0.02
     26   29   33 24 36 37   41   27   32   35   38
M1 0.28 0.34 0.99  0  0  0 0.00 0.01 0.02 0.03 0.00
M2 0.03 0.08 0.00  1  1  1 0.93 0.05 0.11 0.02 0.00
M3 0.01 0.01 0.00  0  0  0 0.01 0.01 0.01 0.48 0.90
N  0.68 0.58 0.01  0  0  0 0.06 0.92 0.85 0.47 0.09
```

As with FCM, some objects have intermediate memberships. In addition, there are some objects with high membership to the Noise class, which indicates that they are distant from all 'true' cluster centers. These objects can be considered 'outliers' and are left unclassified:

```
> groups = defuzzify(wetland.nc)$cluster
> groups

   5    8   13    4   17    3    9   21   16   14    2   15    1
"M3" "M3" "M3" "M3" "M3"  "N"  "N" "M3" "M3" "M3" "M3" "M3" "M3"
   7   10   40   23   25   22   20    6   18   12   39   19   11
 "N"  "N" "M1" "M1" "M1" "M3" "M1" "M1" "M1" "M1" "M1" "M1" "M1"
  30   34   28   31   26   29   33   24   36   37   41   27   32
 "N" "M2"  "N" "M2"  "N"  "N" "M1" "M2" "M2" "M2" "M2"  "N"  "N"
  35   38
"M3" "M3"

> table(groups)

groups
M1 M2 M3  N
11  6 14 10
```

Note that we can defuzzify the membership matrix using a threshold, as before, and determine both intermediates ('N') and outliers ('NA'):

```
> groups = defuzzify(wetland.nc, method="cut", alpha=0.8)$cluster
> groups

   5    8   13    4   17    3    9   21   16   14    2   15    1
"M3" "M3" "M3" "M3" "M3"   NA  "N" "M3" "M3" "M3" "M3" "M3" "M3"
   7   10   40   23   25   22   20    6   18   12   39   19   11
  NA  "N" "M1" "M1" "M1"   NA "M1" "M1" "M1" "M1" "M1" "M1" "M1"
  30   34   28   31   26   29   33   24   36   37   41   27   32
 "N" "M2"  "N" "M2"   NA   NA "M1" "M2" "M2" "M2" "M2"  "N"  "N"
  35   38
  NA "M3"

> table(groups, useNA = "always")

groups
  M1   M2   M3    N <NA>
  11    6   12    6    6
```

In vegetation classification, distinguishing between an intermediate or an outlier will not be clearcut. Nevertheless, the distinction may be useful in practice because outlier objects may relate to vegetation patterns that exist in the study area but happen to be underrepresented in the sample. That is, outlier plant communities may be rare for the given vegetation data set only, in the sense that if new data is added they would belong to a

vegetation type. Alternatively, they may represent rare species assemblages for the study area. Distinguishing between one case or the other cannot be done without collecting more data [Wiser and De Cáceres, 2012].

An advantage of the NC model over FCM or KM is that 'outliers' do not influence the cluster centers. As a result, the cluster centers are more separated from each other than in the previous models. Compare the following distance matrices between cluster centers:

```
> dist(wetland.km$mobileCenters)

          M1        M2
M2 0.9248370
M3 0.9567205 0.8997438

> dist(wetland.fcm$mobileCenters)

          M1        M2
M2 0.8852461
M3 0.9280613 0.9387273

> dist(wetland.nc$mobileCenters)

          M1        M2
M2 1.0102623
M3 0.9735367 1.0666598
```

However, this particular advantage is also obtained (in partitive methods) if medoids are used as prototypes instead of centroids (see below).

3.1.4 Medoid-based clustering

All the examples that we have shown so far could be repeated using medoids as cluster prototypes instead of centroids. For example, with the K-medoids (the K-means analogue) would be:

```
> wetland.kmdd = vegclust(x = wetlandchord, mobileCenters=3,
+                         method="KMdd", nstart=20)
> t(wetland.kmdd$memb)

   5 8 13 4 17 3 9 21 16 14 2 15 1 7 10 40 23 25 22 20 6 18 12 39
M1 0 0  0 0  0 0 0  0  0  0 0  0 0 0  0  0  0  0  0  0 0  0  0  0
M2 1 1  1 1  1 1 1  1  1  1 1  1 1 1  1  0  0  0  1  0 0  0  0  0
M3 0 0  0 0  0 0 0  0  0  0 0  0 0 0  0  1  1  1  0  1 1  1  1  1
   19 11 30 34 28 31 26 29 33 24 36 37 41 27 32 35 38
M1  0  0  1  1  1  1  0  0  0  1  1  1  1  1  1  0  0
M2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1
M3  1  1  0  0  0  0  1  1  1  0  0  0  0  0  0  0  0
```

When ran using a site-by-species matrix as input, `vegclust` returns the coordinates of medoids as the cluster centers:

```
> round(wetland.kmdd$mobileCenters, dig=3)
```

```
   Abefic Merhed Alyvag Pancam Abemos Melcor Ludoct Eupvac Echpas
M1      0      0      0  0.258      0  0.258      0  0.258      0
M2      0      0      0  0.000      0  0.183      0  0.000      0
M3      0      0      0  0.000      0  0.177      0  0.000      0
   Passcr Poa2 Carhal Dendio Casobt Aesind Cyprot Ipocop Cynarc
M1  0.258    0  0.000      0      0      0      0      0  0.775
M2  0.000    0  0.000      0      0      0      0      0  0.000
M3  0.000    0  0.177      0      0      0      0      0  0.000
   Walind Sessp. Phynod Echell Helind Ipoaqu Orysp. Elesp. Psespi
M1      0  0.258  0.258  0.000  0.000  0.000  0.000  0.000  0.000
M2      0  0.000  0.000  0.000  0.000  0.000  0.913  0.183  0.183
M3      0  0.000  0.884  0.177  0.177  0.177  0.000  0.177  0.177
   Ludads Polatt Poa1 Helcri Physp. Goopur
M1  0.000  0.000    0      0      0      0
M2  0.183  0.183    0      0      0      0
M3  0.000  0.000    0      0      0      0
```

However, when using site-by-site dissimilarity matrices as input for `vegclustdist`, the indices of objects are returned instead:

```
> wetland.kmdd = vegclustdist(x = dchord, mobileMemb=3,
+                         method="KMdd", nstart=20)
> wetland.kmdd$mobileCenters
```

```
[1] 23 34 11
```

3.2 Supervised classification: `as.vegclust` and `vegclass`

Vegetation types are meant to be used. For example, a new area may be surveyed and a map of vegetation types may be needed. Here we simulate the process of assigning new observations to a vegetation classification created *a priori*. In order to simulate this two-step process, we split our example data set into two matrices, one with the 31 objects whose group classification will be known *a priori* and the other with the 10 objects whose classification is to be studied:

```
> wetland.31 = wetlandchord[1:31,]
> wetland.31 = wetland.31[,colSums(wetland.31)>0]
> dim(wetland.31)
```

```
[1] 31 27
```

```
> wetland.10 = wetlandchord[-(1:31),]
> wetland.10 = wetland.10[,colSums(wetland.10)>0]
> dim(wetland.10)

[1] 10 22
```

As initial classification, we simply take the two groups resulting from a K-means analysis (using function `kmeans` from the `stats` package) on the first data set:

```
> groups = kmeans(wetland.31, 2)$cluster
> groups

 5  8 13  4 17  3  9 21 16 14  2 15  1  7 10 40 23 25 22 20  6 18
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  2  2  2  2  2  2  2
12 39 19 11 30 34 28 31 26
 2  2  2  2  2  2  2  2  2
```

The idea is to know whether the ten objects of the second data set may be assigned to the vegetation types defined using the first data set. Because our initial classification was not obtained using `vegclust`, we need to transform the input classification of 31 objects into an object of class 'vegclust'. This is done using function `as.vegclust`:

```
> wetland.31.km = as.vegclust(wetland.31, groups)
```

Note that we did not specify the clustering model for our 'vegclust' object. By default, the clustering method is K-means (KM):

```
> wetland.31.km$method

[1] "KM"
```

In this case, this matches the way we obtained our initial classification. In general, however, we may have a classification obtained following an informal (or unknown) procedure, and we will choose the clustering model according to our preferences for assignments. Once we have our object 'vegclust' we can use function `vegclass` to *assign* the second set of observations according to the membership rule of the k-means cluster model:

```
> wetland.10.km = vegclass(wetland.31.km, wetland.10)
> defuzzify(wetland.10.km)$cluster

 29  33  24  36  37  41  27  32  35  38
"2" "2" "2" "2" "2" "2" "2" "2" "1" "1"
```

Note that all the objects of the second set were assigned to the nearest cluster. What if we want to detect which objects should better form a new vegetation type? In that is case, we may do better by using noise clustering. We can choose for example the hard noise clustering (HNC) model:

```
> wetland.31.nc = as.vegclust(wetland.31, groups, method="HNC",
+                             dnoise = 0.8)
> wetland.10.nc = vegclass(wetland.31.nc, wetland.10)
> defuzzify(wetland.10.nc)$cluster

 29  33  24  36  37  41  27  32  35  38
"2" "2" "N" "2" "N" "N" "N" "N" "N" "1"
```

An additional parameter is needed: the distance to the noise class 'dnoise', δ. This can be set either conventionally (depending on the level of abstraction of vegetation types) or relying on the variance of the original clusters. The results of the noise clustering model show that several of the ten objects are assigned to the noise class ('N'), which indicates that some of them could be used to define a new cluster.

3.3 Extending vegetation classifications

Vegetation classifications are dynamic entities, in the sense that they may be modified or extended as new surveys are conducted (or in general, when new data becomes available) [De Cáceres et al., 2010, Wiser and De Cáceres, 2012]. The idea here is to preserve the two prototypes of the initial classification and let `vegclust` to define a new vegetation type. We first calculate the centroids of the initial classification in the resemblance space of all species (although we use object `wetlandchord`, one will normally use function `conformveg` to merge the resemblance space of two data sets):

```
> fixed = clustcentroid(wetlandchord[1:length(groups),], groups)
```

And now we are ready to call `vegclust`:

```
> wetland.nc = vegclust(wetlandchord, mobileCenters=1,
+                       fixedCenters = fixed,
+                       method = wetland.31.nc$method,
+                       dnoise=wetland.31.nc$dnoise, nstart=10)
> defuzzify(wetland.nc)$cluster

   5    8   13    4   17    3    9   21   16   14    2   15    1
"F2" "F2" "F2" "F2" "F2" "F2"  "N" "F2" "F2" "F2" "F2" "F2" "F2"
   7   10   40   23   25   22   20    6   18   12   39   19   11
"F2"  "N" "F3" "F3" "F3" "F3" "F3" "F3" "F3" "F3" "F3" "F3" "F3"
  30   34   28   31   26   29   33   24   36   37   41   27   32
```

```
 "N" "M1"  "N" "M1" "F3" "F3" "F3" "M1" "M1" "M1" "M1"  "N"  "N"
  35   38
 "N" "F2"
```

Here, function `vegclust` has renamed the original clusters as 'F2' and 'F3', while the new one is named 'M1'. Note that some of the objects in the first data set may have been reassigned differently (to a different cluster or to the Noise class). While the centroids of the original classification are preserved, the membership of particular objects may change because the clustering model is not the original one. If the new memberships were strikingly different, one could decide to start the three group classification from scratch.

Instead of relying on the Noise clustering model, we could have chosen to use the K-means model to extend the classification:

```
> wetland.km = vegclust(wetlandchord, mobileCenters=1,
+                       fixedCenters = fixed,
+                       method = "KM",
+                       nstart=10)
> defuzzify(wetland.km)$cluster
```

```
   5    8   13    4   17    3    9   21   16   14    2   15    1
"F2" "F2" "F2" "F2" "F2" "F2" "F2" "F2" "F2" "F2" "F2" "F2" "F2"
   7   10   40   23   25   22   20    6   18   12   39   19   11
"F2" "F2" "F3" "F3" "F3" "F3" "F3" "F3" "F3" "F3" "F3" "F3" "F3"
  30   34   28   31   26   29   33   24   36   37   41   27   32
"M1" "M1" "M1" "M1" "F3" "F3" "F3" "M1" "M1" "M1" "M1" "M1" "M1"
  35   38
"F2" "F2"
```

This avoids having objects in the Noise class. However, note that the noise clustering model allows objects in the Noise to be classified later on when new data becomes available, instead of forcing them to belong to one cluster or the other (i.e. the partition restriction).

4 Cluster characterization

In this section we show how to use a set of auxiliary functions that allow extracting cluster properties from an input classification. In all examples we will use this *a priori* classification of our data set:

```
> groups = c(rep(1, 17), rep(2, 14), rep(3,10))
```

4.1 Cluster prototypes: `clustcentroid` and `clustmedoid`

Functions `clustcentroid` and `clustmedoid` allow calculating the cluster centers (i.e. prototypes) corresponding to an input classification structure, which can be specified using either a cluster vector or a membership matrix.

For example, with `clustcentroid` we can calculate the coordinates of the centroids of the initial groups using eq. 1:

```
> centroids = clustcentroid(wetlandchord, groups)
> round(centroids, dig=3)

  Abefic Merhed Alyvag Pancam Abemos Melcor Ludoct Eupvac Echpas
1  0.000  0.000  0.009  0.009  0.000  0.054  0.000  0.020  0.031
2  0.013  0.015  0.036  0.088  0.000  0.178  0.000  0.090  0.116
3  0.000  0.047  0.000  0.371  0.036  0.253  0.019  0.182  0.061
  Passcr Poa2 Carhal Dendio Casobt Aesind Cyprot Ipocop Cynarc
1  0.000 0.00  0.064  0.000  0.016  0.009  0.000  0.000  0.000
2  0.000 0.01  0.036  0.000  0.000  0.064  0.179  0.028  0.106
3  0.026 0.00  0.000  0.027  0.000  0.019  0.039  0.018  0.301
  Walind Sessp. Phynod Echell Helind Ipoaqu Orysp. Elesp. Psespi
1  0.068  0.144  0.090  0.022  0.112  0.055  0.609  0.164  0.192
2  0.055  0.117  0.506  0.119  0.138  0.048  0.126  0.125  0.027
3  0.090  0.138  0.204  0.158  0.090  0.037  0.172  0.000  0.000
  Ludads Polatt  Poa1 Helcri Physp. Goopur
1  0.204  0.097 0.000  0.000   0.00   0.00
2  0.000  0.000 0.013  0.013   0.00   0.00
3  0.000  0.000 0.000  0.000   0.02   0.02
```

As medoids are prototypes that are chosen among the objects to be classified, function `clustmedoid` does not return coordinates but the indices of objects. The following code uses eq. 3 to determine the medoids of each cluster:

```
> medoids = clustmedoid(wetlandchord, groups)
> medoids

 2 12 41
11 23 37
```

The value returned by function `medoid` is a vector of indices with the corresponding object names (which are numbers in this particular case). If the classification structure is a fuzzy membership matrix, the cluster centroids or medoids are determined using eqs. 2 and 4, respectively.

4.2 Cluster internal variability: `clustvar`

Vegetation types may differ in the amount of variability. Function `clustvar` allows determining the amount of compositional variation (i.e. beta diversity) observed among the sites of sites belonging to each cluster in a classification structure. For clusters whose prototype is a centroid, this is calculated as the mean of squared distances from each object of the group to the group centroid. For example, the variability for a given group i would be:

$$Var(i) = \sum_{j=1}^{n_k} e_{ij}^2 / n_i \tag{13}$$

Note that division is by n_i and not by $(n_i - 1)$, which would give an unbiased sample estimate. Thus, the variances calculated in `clustvar` are population variances. For example, the variances of the three groups in our examples are:

```
> clustvar(wetlandchord, groups)

        1         2         3
0.4554668 0.5466579 0.5293836
```

The reason why population values are produced, instead of sample estimates, is because it allows calculating the variance using fuzzy membership values as:

$$Var(i) = \frac{\sum_{j=1}^{n} u_{ij}^m e_{ij}^2}{\sum_{j=1}^{n} u_{ij}^m} \tag{14}$$

Cluster variances can also be obtained using distance or dissimilarity matrices. In this case, the variance for a given group is calculated as:

$$Var(i) = \frac{1}{n_i^2} \sum_{k=1}^{n_i} \sum_{l=1}^{n_i} d_{kl}^2 \tag{15}$$

Again, division by $n_i(n_i - 1)$ instead of n_i^2 would give an unbiased variance estimate. Because in our example the community data had been transformed using the chord transformation, the same variance values can be obtained using a distance matrix with chord distances:

```
> clustvar(dchord, groups)

        1         2         3
0.4554668 0.5466579 0.5293836
```

Finally, if no classification structure is provided function `clustvar` will return the overall variance (beta diversity) of the data table:

```
> clustvar(wetlandchord)

[1] 0.6751038
```

4.3 Distance between clusters: `interclustdist`

Calculating distance between pairs of cluster prototypes is useful to determine which vegetation types are more similar and which are more distinct.When prototypes of vegetation types are chosen to be cluster medoids, then the resemblance between vegetation types can be defined as the resemblance between the corresponding medoids:

```
> as.dist(as.matrix(dchord)[medoids,medoids])
```

```
          2       12
12 1.344006
41 1.093926 1.363297
```

In contrast, if prototypes of vegetation types are chosen to be cluster centroids, the distance between two vegetation types should be defined as the distance between the cluster centroids. Following our example, we can simply use function `dist` on the matrix of centroid coordinates:

```
> dist(centroids)
```

```
          1         2
2 0.7624211
3 0.8004329 0.5298008
```

Alternatively, function `interclustdist` allows calculating the distance between pairs of centroids without having the coordinates of centroids themselves. Instead, the matrix of distances between objects is used. For example, if the distance between the centroids of groups i and j is desired, we can calculate the squared distance by:

$$d^2(i,j) = \frac{\sum_{k=1}^{n}\sum_{l=1}^{n} u_{ik}^m u_{jl}^m {d_{kl}}^2}{\sum_{k=1}^{n} u_{ik}^m \sum_{l=1}^{n} u_{jl}^m} - Var(i) - Var(j) \qquad (16)$$

Using equation 16 in our example would be:

```
> interclustdist(dchord,groups)
```

```
          1         2
2 0.7624211
3 0.8004329 0.5298008
```

4.4 Constancy classes: `clustconst`

One way of characterizing a vegetation type is by examining the how frequently each species occurs in vegetation observations belonging to the type. This frequency is often called *constancy* and the table that contains the constancy of all species in all vegetation types is called *constancy table*. Function `clustconst` allows calculating this table:

```
> c = clustconst(wetlandchord, memb = as.memb(groups))
```

The R object returned by `clustconst` can be examined in several ways. First, it is useful to print the constancy table ordering species from high to low constancy for each cluster:

```
> d=summary(c, mode="all")
```

```
------------ 3 -------------
          3     2     1
Pancam 1.000 0.357 0.059
Melcor 1.000 0.786 0.294
Eupvac 0.800 0.357 0.118
Sessp. 0.600 0.500 0.412
Echell 0.600 0.357 0.118
------------ 2 -------------
          3     2     1
Phynod 0.400 0.714 0.118
Helind 0.300 0.643 0.471
Elesp. 0.000 0.571 0.529
------------ 1 -------------
          3     2     1
Orysp. 0.300 0.286 1.000
Ludads 0.000 0.000 0.824
```

Alternatively, we can examine the constancy vector of each vegetation type:

```
> summary(c, mode="cluster", name=names(c)[1])
```

```
Orysp. 1.000
Ludads 0.824
Elesp. 0.529
```

References

James C Bezdek. *Pattern recognition with fuzzy objective functions*. Plenum Press, New York, 1981.

D.M.J.S. Bowman and B A Wilson. Wetland vegetation pattern on the Adelaide River flood plain, Northern Territory, Australia. *Proceedings of the Royal Society of Queensland*, 97:69–77, 1986.

MB Dale. Evaluating classification strategies. *Journal of Vegetation Science*, 6(3):437–440, 1995. ISSN 1654-1103. URL `http://onlinelibrary.wiley.com/doi/10.2307/3236243/abstract`.

R N Dave. Characterization and detection of noise in clustering. *Pattern Recognition Letters*, 12(11):657–664, 1991.

R.N. Dave and R. Krishnapuram. Robust clustering methods: a unified view. *IEEE Transactions on Fuzzy Systems*, 5(2):270–293, May 1997. ISSN 10636706. doi: 10.1109/91.580801. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=580801.

Miquel De Cáceres, Francesc Oliva, and Xavier Font. On relational possibilistic clustering. *Pattern Recognition*, 39(11):2010–2024, November 2006. ISSN 00313203. doi: 10.1016/j.patcog.2006.04.008. URL http://linkinghub.elsevier.com/retrieve/pii/S0031320306001634.

Miquel De Cáceres, Xavier Font, and Francesc Oliva. The management of vegetation classifications with fuzzy clustering. *Journal of Vegetation Science*, 21(6):1138–1151, December 2010. ISSN 11009233. doi: 10.1111/j.1654-1103.2010.01211.x. URL http://doi.wiley.com/10.1111/j.1654-1103.2010.01211.x.

R Hathaway, J.C. Bezdek, and J Davenport. On relational data versions of c-means algorithms. *Pattern Recognition Letters*, 17(6):607–612, May 1996. ISSN 01678655. doi: 10.1016/0167-8655(96)00025-6. URL http://linkinghub.elsevier.com/retrieve/pii/0167865596000256.

R J Hathaway and James C Bezdek. NERF c-means: Non-euclidean relational fuzzy clustering. *Pattern recognition*, 27(3):429–437, 1994.

R J Hathaway, J W Davenport, and James C Bezdek. Relational duals of the c-means clustering algorithms. *Pattern recognition*, 22(2):205–212, 1989.

R. Krishnapuram and J.M. Keller. A possibilistic approach to clustering. *IEEE Transactions on Fuzzy Systems*, 1(2):98–110, May 1993. ISSN 10636706. doi: 10.1109/91.227387. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=227387.

R. Krishnapuram and J.M. Keller. The possibilistic C-means algorithm: insights and recommendations. *IEEE Transactions on Fuzzy Systems*, 4(3):385–393, 1996. ISSN 10636706. doi: 10.1109/91.531779. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=531779.

Raghu Krishnapuram, Anupam Joshi, and Liyu Yi. A Fuzzy relative of the k-medoids algorithm with application to web document and snippet clustering. In *IEEE International Fuzzy Systems*, pages 1281–1286, 1999. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=790086.

Pierre Legendre and Eugene Gallagher. Ecologically meaningful transformations for ordination of species data. *Oecologia*, 129(2):271–280,

October 2001. ISSN 0029-8549. doi: 10.1007/s004420100716. URL http://www.springerlink.com/openurl.asp?genre=article&id=doi:10.1007/s004420100716

J MacQueen. Some methods for classification and analysis of multivariate observation. Number 0, pages 281–297. University of California Press, Berkeley, 1967.

J. Moravec. Influences of the individualistic concept of vegetation on syntaxonomy. *Vegetatio*, 81(1-2):29–39, July 1989. ISSN 0042-3106. doi: 10.1007/BF00045511. URL http://www.springerlink.com/index/10.1007/BF00045511.

L Orlóci. An agglomerative method for classification of plant comunities. *Journal of Ecology*, 55(0):193–206, 1967.

Susan K. Wiser and Miquel De Cáceres. Updating vegetation classifications: an example with New Zealand's woody vegetation. *Journal of Vegetation Science*, pages n/a–n/a, July 2012. ISSN 11009233. doi: 10.1111/j.1654-1103.2012.01450.x. URL http://doi.wiley.com/10.1111/j.1654-1103.2012.01450.x.