Cluster Analysis: Tutorial with R

Jari Oksanen

January 27, 2014

Contents

1 Introduction

In this tutorial we inspect classification. Classification and ordination are alternative strategies of simplifying data. Ordination tries to simplify data into a map showing similarities among points. Classification simplifies data by putting similar points into same class. The task of describing a high number of points is simplified to an easier task of describing a low number of classes.

2 Hierarchic Clustering

The classification methods are available in standard R packages. The **vegan** package does not have many support functions for classification, but we still load vegan to have access to its data sets and some of its support functions.[1]

[1] If you do not have a package, but get an error message, you must install package using `install.packages("vegan")` or the installation menu.

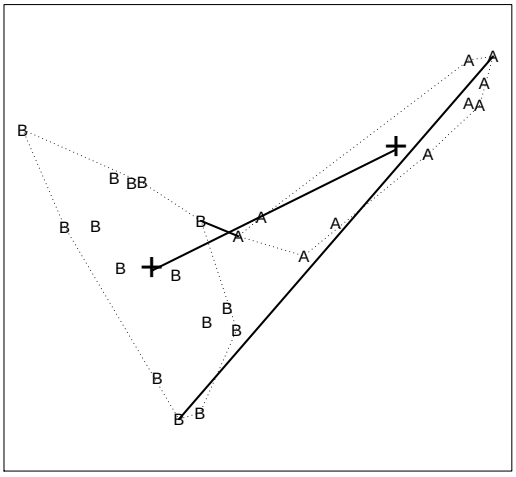

Figure 1: Distance between two clusters `A` and `B` defined by single, complete and average linkage. Mark each of the linkage types in the connecting line. The fusion level in the cluster dendrogram would be the length of the corresponding connecting line of the linkage type.

```
R> library(vegan)
R> data(dune)
```

Hierarchic clustering (function `hclust`) is in standard R and available without loading any specific libraries. Hierarchic clustering needs dissimilarities as its input. Standard R has function `dist` to calculate many dissimilarity functions, but for community data we may prefer **vegan** function `vegdist` with ecologically useful dissimilarity indices. The default index in `vegdist` is Bray–Curtis:

```
R> d <- vegdist(dune)
```

Ecologically useful indices in **vegan** have an upper limit of 1 for absolutely different sites (no shared species), and they are based on differences of abundances. In contrast, the standard Euclidean distance has no upper limit, but varies with the sum of total abundances of compared sites when there are no shared species, and uses squares of differences of abundances. There are many other ecologically useful indices in `vegdist`, but Bray–Curtis is usually not a bad choice.

There are several alternative clustering methods in the standard function `hclust`. We shall inspect three basic methods: single linkage, complete linkage and average linkage. All these start in the same way: they fuse two most similar points to a cluster. They differ in the way they combine clusters to each other, or new points to existing clusters (Fig. 1). In single linkage (a.k.a. nearest neighbour, or neighbour joining tree in genetics) the distance between two clusters is the shortest possible distance among members of the clusters, or the best of the friends. In complete linkage (a.k.a. furthest neighbour) the distance between two clusters is the longest possible distance between the groups, or the worst among the friends. In average linkage, the distance between the clusters

is the distance between cluster centroids. There are several alternative ways of defining the average and defining the closeness, and hence a huge number of average linkage methods. We only use one of these methods commonly known as UPGMA. The lecture slides discuss the methods in more detail.

In the following we will compare three different clustering strategies. If you want to plot three graphs side by side, you can divide the screen into three panels by

```
R> par(mfrow=c(1,3))
```

This defines three panels side by side. You probably want to stretch the plotting window if you are using this option. Alternatively, you can have three panels above each other with

```
R> par(mfrow=c(3,1))
```

You can get back to the single panel mode with

```
R> par(mfrow=c(1,1))
```

You may also wish to use narrower empty margins for the panels:

```
R> par(mar=c(3,4,1,1)+.1)
```

The `mar` command defines plot margins in order bottom, left, up, right using row height (text height) as a unit.

The single linkage clustering can be found with:

```
R> csin <- hclust(d, method="single")
R> csin

Call:
hclust(d = d, method = "single")

Cluster method   : single
Distance         : bray
Number of objects: 20
```

The dendrogram can be plotted with:

```
R> plot(csin)
```

The default is to plot an inverted tree with the root at the top, and branches hanging down. You can force the branches down to the base line giving the `hang` argument:

```
R> plot(csin, hang=-1)
```

If you plotted the `csin` tree twice you consumed two panels out of three you have, and there will not be space for the next two trees in the same plot. In that case you can start a new plot by issuing again the `mfrow` command and then drawing `csin` again.

The complete linkage and average linkage methods are found in the same way:

Figure 2: Vegemite is an Australian national delicacy made of yeast extract. The `vegemite` function was named because its output is just as dense as Vegemite.

```
R> ccom <- hclust(d, method="complete")
R> plot(ccom, hang=-1)
R> caver <- hclust(d, method="aver")
R> plot(caver, hang=-1)
```

The vertical axes of the cluster dendrogram show the fusion level. The two most similar observations are combined first, and they are at the same level in all dendrograms. At the upper fusion levels, the scales diverge: they are the shortest dissimilarities among cluster members in single linkage, the longest possible dissimilarities in complete linkage, and the distances among cluster centroids in average linkage (Fig. 1).

2.1 Description of Classes

One problem with hierarchic clustering is that it gives a classification of observations (plots, sampling units), but it does not tell how these classes differ from each other. For community data, there is no information how the species composition differs between classes (we return to this subject in Chapter 3.2).

The **vegan** package has function `vegemite` (Fig. 2) that can produce compact community tables ordered by a dendrogram, ordination or environmental variables. With the help of these tables it is possible to see which species differ in classification:

```
R> vegemite(dune, caver)

          111211 11    11    1
          46507912334891856720
 Comapalu 2.2.................
 Callcusp 43.3................
 Eleopalu 4854.......4........
```

```
Ranuflam 2224....2..2.......
Airaprae ....23.............
Empenigr .....2.............
Agrostol 4745...454843.......
Juncarti .334.......44.......
Salirepe ...5.3........3.....
Hyporadi ....25.......2......
Chenalbu ........1...........
Alopgeni .4.....857253.....2.
Sagiproc .....3.42.5222......
Bracruta .444.3.4.222246262.2
Cirsarve ..........2.........
Juncbufo .......43...4....2..
Scorautu 2.2226.2222325533353
Elymrepe ......4..44.6..4..4.
Trifrepe 6.1..2.3221233225256
Anthodor ....44.........432.4
Poatriv  .2....2496545..64574
Poaprat  ....1.4.254444323444
Lolipere ......7..65427226656
Rumeacet .......2....2..563..
Bellpere .........22...22..32
Vicilath ............21....1
Planlanc ....2........33555.3
Achimill ....2.1........22234
Trifprat ..............252..
Bromhord ..........3....2.244
 sites species
    20      30
```

The `vegemite` command will always use one-character columns. If the observed values do not fit one character, the `vegemite` refuses to work. With argument `scale` you can recode the values to one-character width. The `vegemite` has a graphical sister function `tabasco` that is described in section 2.4.

2.2 Numbers of Classes

The hierarchic clustering methods produce all possible levels of classifications. The extremes are all observations in a single class, and each observation in its private class. The user normally wants to have a clustering into a certain number of classes. The fixed classification can be visually demonstrated with `rect.hclust` function:

```
R> plot(csin, hang=-1)
R> rect.hclust(csin, 3)
R> plot(ccom, hang=-1)
R> rect.hclust(ccom, 3)
R> plot(caver, hang=-1)
R> rect.hclust(caver, 3)
```

Single linkage has a tendency to chain observations: most common case is to fuse a single observation to an existing class: the single link is the nearest

neighbour, and a close neighbour is more probably in a large group than in a small group or a lonely point. Complete linkage has a tendency to produce compact bunches: complete link minimizes the spread within the cluster. The average linkage is between these two extremes.

We can extract classification in a certain level using function `cutree`:

```
R> cl <- cutree(ccom, 3)
R> cl

 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 1  1  2  2  1  1  1  2  2  1  3  2  2  2  2  2  3  3  3  2
```

This gives a numeric classification vector of cluster identities. The clusters are numbered in the order the observations appear in the data: the first item will always belong to cluster 1, and the numbering does not match the dendrogram. We can tabulate the numbers of observations in each cluster:

```
R> table(cl)

cl
 1  2  3
 6 10  4
```

We can compare two clustering schemes by cross-tabulation which gives as a confusion matrix:

```
R> table(cl, cutree(csin, 3))

cl   1  2  3
  1  6  0  0
  2 10  0  0
  3  2  1  1

R> table(cl, cutree(caver, 3))

cl  1 2 3
  1 6 0 0
  2 6 4 0
  3 2 0 2
```

The confusion matrix tabulates the classifications against each other. The rows give the first classification, and the columns the second classification. If the classifications match and there is no "confusion", each row and each column has only one non-zero entry, but if the classes are divided between several classes in the second classification, the row has several non-zero entries.

2.3 Clustering and Ordination

We can use ordination to display the observed dissimilarities among points. A natural choice is to use metric scaling a.k.a. principal coordinates analysis (PCoA) that maps observed dissimilarities linearly onto low-dimensional graph using the same dissimilarities we had in our clustering.

The metric scaling can be performed with standard R function `cmdscale`:

```
R> ord <- cmdscale(d)
```

We can display the results using **vegan** function `ordiplot` that can plot results of any **vegan** ordination function and many non-**vegan** ordination functions, such as `cmdscale`, `prcomp` and `princomp` (the latter for principal components analysis):

```
R> ordiplot(ord)
```

We got a warning because `ordiplot` tries to plot both species and sites in the same graph, and the `cmdscale` result has no species scores. We do not need to care about this warning.

There are many **vegan** functions to overlay classification onto ordination. For distinct, non-overlapping classes convex hulls are practical:

```
R> ordihull(ord, cl, lty=3)
R> ordispider(ord, cl, col="blue", label=TRUE)
R> ordiellipse(ord, cl, col="red")
```

For overlapping classes we can use `ordispider`. If we are not interested in individual points, but on class centroids and average dispersions, we can use `ordiellipse`, both with the same arguments as `ordihull`.

The other clustering results can be seen with:

```
R> ordiplot(ord, dis="si")
R> ordihull(ord, cutree(caver, 3))
R> ordiplot(ord, dis="si")
R> ordicluster(ord, csin)
```

We set here explicitly the `display` argument to `display = "sites"` to avoid the annoying and useless warnings. The contrasting clustering strategies (nearest vs. furthest vs. average neighbour) are evident in the shapes of the clusters. Single linkage clusters are chained, complete linkage clusters are compact, and average linkage clusters between these two.

The **vegan** package has a special function to display the cluster fusions in ordination. The `ordicluster` function combines sites and cluster centroids similarly as the average linkage method:

```
R> ordiplot(ord, dis="si")
R> ordicluster(ord, caver)
```

We can prune the top level fusions to highlight the clustering:

```
R> ordiplot(ord, dis="si")
R> ordicluster(ord, caver, prune=2)
```

2.4 Reordering a Dendrogram

The leaves of a dendrogram do not have a natural order. You can take a branch and turn around its root, and the tree is the same (see Fig. 3).

R has two alternative dendrogram presentations: the `hclust` result object and a general `dendrogram` object. The cluster type can be changed with:

```
R> den <- as.dendrogram(caver)
```

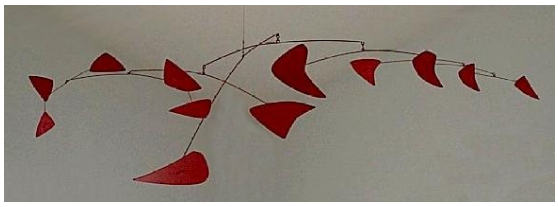

Figure 3: A dendrogram is similar to a mobile: branches can turn around, but the mobile is the same. Alexander Calder: Red Mobile.

The dendrograms are more general, and several methods are available for their manipulation and analysis. It is possible to re-order the leaves of a dendrogram so that they match as closely as possible an external variable.

In the following we rearrange the dendrogram so that the ordering of leaves corresponds as closely as possible with the first ordination axis:

```
R> x <- scores(ord, display = "sites", choices = 1)
R> oden <- reorder(den, x)
```

We plot the dendrorgrams together:

```
R> par(mfrow=c(2,1))
R> plot(den)
R> plot(oden)
R> par(mfrow=c(1,1))
```

The reordered dendrogram may also give more regularly structured community table:

```
R> vegemite(dune, oden)
```

```
            1 11    11 111112
          76502183498321794650
 Trifprat 252.................
 Rumeacet 365......2..2.......
 Planlanc 5553.33.......2.....
 Bromhord 2.244...3...........
 Achimill 22243........12.....
 Vicilath ...1.21.............
 Bellpere ..223.222...........
 Lolipere 66265726524..7......
 Poaprat  432444354442.41.....
 Anthodor 2344..........44....
 Poatriv  54647..6554942...2..
 Elymrepe ..4.4..446...4......
 Trifrepe 2526532213223..26.1.
 Cirsarve ........2...........
 Scorautu 3333555222322.262.22
 Juncbufo 2........4.34.......
 Bracruta 2622.462222.4..3.444
 Sagiproc .....2..52224..3....
```

```
Alopgeni ....2..723558....4..
Chenalbu ..........1........
Hyporadi .....2........25....
Agrostol .......483454...4745
Juncarti .........44......334
Salirepe ......3........3...5
Airaprae ..............23....
Empenigr ...............2....
Ranuflam ..........22....2224
Eleopalu ..........4.....4854
Comapalu ................2.2.
Callcusp ................43.3
 sites species
    20      30
```

The `vegemite` function has a graphical sister function `tabasco`[2] that can also display the dendrogram. Moreover, it defaults to rerrange the dendrogram by the first axis of Correspondence Analysis:

```
R> tabasco(dune, caver)
```

Correspondence Analysis packs similar species next to each other, and similar sites next to each other and gives a good diagonal representation of the data. If you want to see the original ordering of the sample plots, you must set `Rowv = FALSE`:

```
R> tabasco(dune, caver, Rowv = FALSE)
R> tabasco(dune, oden, Rowv = FALSE)
```

`tabasco` function also defaults to order species to match the ordering of sites unless you set `Colv = FALSE`.

2.5 Minimum Spanning Tree

In the mathematical graph theory, tree is a connected graph without loops, spanning tree is a tree connecting all points, and minimum spanning tree is the shortest of such trees. Minimum spanning tree (MST) is closely related to single linkage clustering, which also connects all points with minimum total connecting distance. However, MST really combines points, whereas R representations of single linkage clustering hierarchically connects clusters instead of single points.

MST can be found with **vegan** function `spantree`[3]

```
R> mst <- spantree(d)
```

MST can be overlaid onto an ordination with `lines` command:

```
R> ordiplot(ord, dis="si")
R> lines(mst, ord)
```

[2] So called because it is similar to `vegemite` but hotter.

[3] There are many other implementations MST in R, but the implementation in **vegan** probably is the fastest.

Alternatively, the tree can be displayed with a `plot` command that tries to find a locally optimal configuration for points to reproduce the distances among points along the tree. Internally the function uses Sammon scaling (`sammon` function in the **MASS** package) to find the configuration of points. Sammon scaling is a variant of metric scaling trying to reproduce relative distances among points and it is optimal for showing the local (vs. global) structure of points.

```
R> plot(mst, type="t")
```

```
Initial stress        : 0.03111
stress after  10 iters: 0.01302, magic = 0.500
stress after  20 iters: 0.01139, magic = 0.500
stress after  30 iters: 0.01118, magic = 0.500
stress after  40 iters: 0.01114, magic = 0.500
```

2.6 Cophenetic Distance

The estimated distance between two points is the level at which they are fused in the dendrogram, or the height of the root. A good clustering method correctly reproduces the actual dissimilarities. The distance estimated from a dendrogram is called cophenetic distance. The name echoes the origins of hierarchic clustering in old fashioned numeric taxonomy. Standard R function `cophenetic` estimates the distances among all points from a dendrogram.

We can visually inspect the cophenetic distances against observed dissimilarities. In the following, `abline` adds a line with zero intercept and slope of one, or the equivalence line. We also set equal scaling on x and y axes (`asp = 1`):

```
R> plot(d, cophenetic(csin), asp=1)
R> abline(0, 1)
R> plot(d, cophenetic(ccom), asp=1)
R> abline(0, 1)
R> plot(d, cophenetic(caver), asp=1)
R> abline(0, 1)
```

In single linkage clustering, the cophenetic distances are as long as or shorter than the observed distances: the distance between groups is the shortest possible distance between its members. In complete linkage clustering, the cophenetic distances are as long as or longer than observed distances: the distance between two groups is the longest possible distance between groups. In average linkage clustering, the cophenetic distance is the average of observed distances (cf. Fig. 1).

The correlation between observed and cophenetic distances is called the cophenetic correlation:

```
R> cor(d, cophenetic(csin))
```

```
[1] 0.6601692
```

```
R> cor(d, cophenetic(ccom))
```

```
[1] 0.6707479
```

```
R> cor(d, cophenetic(caver))

[1] 0.8168825
```

The ranking of these cophenetic correlations is not entirely random: it is guaranteed that average linkage (UPGMA) method maximizes the cophenetic correlation.

3 Interpretation of Classes

We commonly want to use classes for prediction of external variables — this is the idea om Finnish forest type system, EU Water Framework Directive and theory of gradient analysis. We make classes from community composition and the classes should describe community composition, and we assume the classes predict environmental conditions.

3.1 Environmental Interpretation

For interpreting the classes we take into use the the environmental data:

```
R> data(dune.env)
```

The `cutree` function produced a numerical classification vector, but we need a `factor` of classes:

```
R> cl <- factor(cl)
```

We can visually display the differences in environmental conditions using `boxplot`. The only continuous variable we have is the thickness of the $A1$ horizon, but we can change the Moisture class to a numeric variable:

```
R> Moist <- with(dune.env, as.numeric(as.character(Moisture)))
```

Compare the result of this to a simpler looking alternative:

```
R> with(dune.env, as.numeric(Moisture))

 [1] 1 1 2 2 1 1 1 4 3 2 1 3 4 4 4 4 2 1 4 4
```

The latter uses the internal representation $(1, 2, 3, 4)$ of the factor instead of its nominal values $(1, 2, 4, 5)$ that we can get with the added `as.character` command.

The boxplot is produced with:

```
R> boxplot(Moist ~ cl, notch=TRUE)
```

The boxplot shows the data extremes (whiskers and the points for extreme cases), lower and upper quartiles and the median. The notches show the approximate 95 % confidence limits of the median: if the notches do not overlap, the medians probably are different at level $P < 0.05$. For small data sets like this, the confidence limits are wide, and you get a warning of waist going over the shoulders.

You can use ANOVA for rigid numerical testing of differences among classes. In R, ANOVA is an analysis method for linear models (`lm`):

```
R> anova(lm(Moist ~ cl))

Analysis of Variance Table

Response: Moist
          Df Sum Sq Mean Sq F value    Pr(>F)
cl         2 36.617 18.3083  12.359 0.0004854 ***
Residuals 17 25.183  1.4814
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

This performs parametric ANOVA test. If we want to use non-parametric permutation tests with the same test statistic, we can use redundancy analysis:

```
R> anova(rda(Moist ~  cl))

Permutation test for rda under reduced model
Permutation: free
Number of permutations: 999

Model: rda(formula = Moist ~ cl)
         Df Variance      F Pr(>F)
Model     2   1.9272 12.359  0.001 ***
Residual 17   1.3254
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

This works nicely in this case because the y-variate (`Moist`) is in the current workspace: the command would not work for variates in data frames unless we `attach` the data frame or use the command as `with(varechem, ...)`.

We can use confusion matrix to compare factor variables and classification:

```
R> with(dune.env, table(cl, Management))

   Management
cl  BF HF NM SF
  1  2  3  0  1
  2  0  2  3  5
  3  1  0  3  0
```

3.2 Community Summaries

We can use the **labdsv** package for analysing community classification.[4]

```
R> library(labdsv)
```

The package has several functions for summarizing community composition in classes. The species occurrence frequencies in classes can be found with

```
R> const(dune, cl)
```

[4] If the following command fails, you must install **labdsv** using `install.packages("labdsv")` or from the installation menu.

```
            1   2    3
Achimill 1.00 0.0 0.25
Agrostol 0.00 1.0 0.00
Airaprae 0.00 0.0 0.50
Alopgeni 0.17 0.7 0.00
Anthodor 0.67 0.0 0.50
Bellpere 0.50 0.2 0.25
Bromhord 0.67 0.1 0.00
Chenalbu 0.00 0.1 0.00
Cirsarve 0.00 0.1 0.00
Comapalu 0.00 0.2 0.00
Eleopalu 0.00 0.5 0.00
Elymrepe 0.50 0.3 0.00
Empenigr 0.00 0.0 0.25
Hyporadi 0.00 0.0 0.75
Juncarti 0.00 0.5 0.00
Juncbufo 0.17 0.3 0.00
Lolipere 1.00 0.4 0.50
Planlanc 0.67 0.0 0.75
Poaprat  1.00 0.5 0.75
Poatriv  1.00 0.7 0.00
Ranuflam 0.00 0.6 0.00
Rumeacet 0.50 0.2 0.00
Sagiproc 0.00 0.5 0.50
Salirepe 0.00 0.1 0.50
Scorautu 0.83 0.9 1.00
Trifprat 0.50 0.0 0.00
Trifrepe 0.83 0.8 0.75
Vicilath 0.17 0.0 0.50
Bracruta 0.67 0.8 0.75
Callcusp 0.00 0.3 0.00
```

and the mean abundances with

```
R> importance(dune, cl)
```

```
            1    2    3
Achimill 2.33 0.00 2.00
Agrostol 0.00 4.80 0.00
Airaprae 0.00 0.00 2.50
Alopgeni 2.00 4.86 0.00
Anthodor 3.25 0.00 4.00
Bellpere 2.33 2.00 2.00
Bromhord 3.00 3.00 0.00
Chenalbu 0.00 1.00 0.00
Cirsarve 0.00 2.00 0.00
Comapalu 0.00 2.00 0.00
Eleopalu 0.00 5.00 0.00
Elymrepe 4.00 4.67 0.00
Empenigr 0.00 0.00 2.00
Hyporadi 0.00 0.00 3.00
```

```
Juncarti 0.00 3.60 0.00
Juncbufo 2.00 3.67 0.00
Lolipere 5.33 4.25 4.50
Planlanc 4.50 0.00 2.67
Poaprat  3.50 3.80 2.67
Poatriv  4.67 5.00 0.00
Ranuflam 0.00 2.33 0.00
Rumeacet 4.67 2.00 0.00
Sagiproc 0.00 3.00 2.50
Salirepe 0.00 5.00 3.00
Scorautu 3.40 2.11 4.50
Trifprat 3.00 0.00 0.00
Trifrepe 4.00 2.50 2.33
Vicilath 1.00 0.00 1.50
Bracruta 3.00 3.00 4.33
Callcusp 0.00 3.33 0.00
```

The **labdsv** has function `indval` for indicator species analysis.[5] The indicator species analysis combines frequency tables (`const`) and mean abundance tables (`importance`) and finds species that are significantly concentrated into specific classes. The model is fitted with:

```
R> mod <- indval(dune, as.numeric(cl))
```

```
[1] "error code =  0"
```

The `indval` result object contains the following items:

```
R> names(mod)
```

```
[1] "relfrq" "relabu" "indval" "maxcls" "indcls" "pval"   "error"
```

The item `relfrq` is the same as the output of `const`, and `relabu` is based on the output of `importance`, but each row is normalized to unit sum so that the table shows the distribution of total abundances in classes. These values are combined into indicator value (`indval`) of each species for each class, and `maxcls` gives the class to which species has the highest indicator value (`indcls`). Finally, vector `pval` gives the probability of finding higher indicator value in random permutations, and if this probability is low, the species is a significant indicator.

Each item can be inspected separately:

```
R> mod$maxcls
```

```
Achimill Agrostol Airaprae Alopgeni Anthodor Bellpere Bromhord Chenalbu
       1        2        3        2        1        1        1        2
Cirsarve Comapalu Eleopalu Elymrepe Empenigr Hyporadi Juncarti Juncbufo
       2        2        2        1        3        3        2        2
Lolipere Planlanc  Poaprat  Poatriv Ranuflam Rumeacet Sagiproc Salirepe
       1        1        1        1        2        1        2        3
Scorautu Trifprat Trifrepe Vicilath Bracruta Callcusp
       3        1        1        3        3        2
```

[5] In old **labdsv** versions the function was called `duleg`, but the usage was similar as for `indval` below.

```
R> mod$pval

Achimill Agrostol Airaprae Alopgeni Anthodor Bellpere Bromhord Chenalbu
   0.002    0.001    0.035    0.014    0.163    0.351    0.015    1.000
Cirsarve Comapalu Eleopalu Elymrepe Empenigr Hyporadi Juncarti Juncbufo
   1.000    0.329    0.062    0.232    0.190    0.006    0.056    0.349
Lolipere Planlanc  Poaprat  Poatriv Ranuflam Rumeacet Sagiproc Salirepe
   0.022    0.175    0.094    0.018    0.010    0.064    0.424    0.114
Scorautu Trifprat Trifrepe Vicilath Bracruta Callcusp
   0.025    0.023    0.468    0.092    0.741    0.208
```

Often it is more practical to look at the `summary`. The `"short"` `summary` lists the significant indicator species with some extra information:

```
R> summary(mod)

         cluster indicator_value probability
Achimill       1          0.8235       0.002
Bromhord       1          0.5797       0.015
Lolipere       1          0.5745       0.022
Poatriv        1          0.5714       0.018
Trifprat       1          0.5000       0.023
Agrostol       2          1.0000       0.001
Alopgeni       2          0.6375       0.014
Ranuflam       2          0.6000       0.010
Hyporadi       3          0.7500       0.006
Airaprae       3          0.5000       0.035
Scorautu       3          0.4874       0.025

Sum of probabilities                =  6.283

Sum of Indicator Values             =  13.19

Sum of Significant Indicator Values  =  7.02

Number of Significant Indicators    =  11

Significant Indicator Distribution

1 2 3
5 3 3
```

The `"long"` `summary` lists all species with their indicator values for classes, but replaces non-significant values with a dot:

```
R> summary(mod, type = "long")

            1    2    3
Achimill 0.82  .    .
Agrostol  .   1.00  .
Airaprae  .    .   0.50
Alopgeni  .   0.64  .
```

```
Anthodor 0.35  .   0.24
Bellpere 0.28  .   0.06
Bromhord 0.58  .    .
Chenalbu  .   0.10  .
Cirsarve  .   0.10  .
Comapalu  .   0.20  .
Eleopalu  .   0.50  .
Elymrepe 0.29 0.12  .
Empenigr  .    .   0.25
Hyporadi  .    .   0.75
Juncarti  .   0.50  .
Juncbufo  .   0.23  .
Lolipere 0.57 0.07 0.12
Planlanc 0.40  .   0.30
Poaprat  0.47 0.13 0.20
Poatriv  0.57 0.30  .
Ranuflam  .   0.60  .
Rumeacet 0.43  .    .
Sagiproc  .   0.27 0.23
Salirepe  .    .   0.38
Scorautu 0.26 0.19 0.49
Trifprat 0.50  .    .
Trifrepe 0.39 0.23 0.19
Vicilath  .    .   0.41
Bracruta 0.17 0.25 0.32
Callcusp  .   0.30  .
```

4 Optimized Clustering at a Given Level

Hierarchic clustering has a history: it starts from the bottom, and combines observations to clusters. At higher levels it can only fuse existing clusters to others, but it cannot break old groups to build better ones. Often we are most interested in classification at a low number of clusters, and these have a long historic burden. Once we have decided upon the number of classes, we can often improve the classification at a given number of clusters. A tool for this task is K-means clustering that produces an optimal solution for a given number of classes.

K-means clustering has one huge limitation for community data: it works in Euclidean metric which rarely is meaningful for community data. In hierarchic clustering we could use non-Euclidean Bray-Curtis dissimilarity, but we do not have this choice here. However, we can work around some problems with appropriate standardization. Hellinger transformation has been suggested to be very useful with Euclidean metric and community data: square root of data divided by row (site) totals. The Hellinger transformation can be performed with **vegan** function `decostand` which has many other useful standardizations and transformations:

```
R> ckm <- kmeans(decostand(dune, "hell"), 3)
R> ckm$cluster
```

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 3  3  3  3  3  3  3  2  3  3  3  2  2  2  2  2  1  3  1  2
```

We can display the results in the usual way:

```
R> ordiplot(ord, dis="si")
R> ordihull(ord, ckm$cluster, col="red")
```

4.1 Optimum Number of Classes

The K-means clustering optimizes for the given number of classes, but what is the correct number of classes? Arguably there is no correct number, because the World is not made of distinct classes, but all classifications are patriarchal artifacts. However, some classifications may be less useful (and worse) than others.

Function `cascadeKM` in **vegan** finds some of these criteria for a sequence of K-means clusterings. The default method is the Calinski–Harabasz criterion which is the F ratio of ANOVA for the predictor variables (species in community data). The test uses Euclidean metric and we use again the Hellinger transformation (if you try without transformation or different transformations, you can see that transformations with `decostand` have capricious effects: obviously the World is not classy in this case).

```
R> ccas <- cascadeKM(decostand(dune, "hell"), 2, 15)
R> plot(ccas, sortq=TRUE)
```

We gave the smallest (2) and largest (15) number of inspected classes in the call, and sorted the plot by classification. Each class is shown by a different colour, and bends in the vertical colour bars show non-hierarchical clustering where the upper levels are no simple fusions of lower levels. The highest value of the criterion shows the optimal number of classes. This kind of graph is called Mondrian plot after an excellent Dutch-born artist Piet Mondriaan of *De Stijl* school (Fig. 4).[6]

5 Fuzzy Clustering

We have so far worked with classification methods which implicitly assume that there are distinct classes. The World is not made like that. If there are classes, they are vague and have intermediate and untypical cases. With one word, they are fuzzy.

Fuzzy classification means that each observation has a certain probability of belonging to a class. In the crisp case, it has probability 1 of belonging to one class, and probability 0 of belonging to any other class. In a fuzzy case, it has probability < 1 for the best class, and probabilities > 0 for several other classes (and the sum of these probabilities is 1).

Fuzzy classification is similar to K-means clustering in finding the optimal classification for a given number of classes, but the produced classification is fuzzy: the result is a probability profile of class membership.

[6] A similar plot for cross classifications is called a Marimekko plot: Google to see what it looks like and how it is constructed.

Figure 4: Piet Mondrian: Composition in red, blue and yellow.

The fuzzy clustering is provided by function `fanny` in package **cluster**[7]. Fuzzy clustering defaults to Euclidean metric, but current versions can accept any dissimilarities. In the following, we also need to use lower "membership exponent" (`memb.exp`): the default 2 gives complete fuzziness and fails here, and lower values give crisper classifications.[8]

```
R> library(cluster)
R> cfuz <- fanny(d, 3, memb.exp=1.7)
```

The function returns an object with the following items:

```
R> names(cfuz)
```

```
 [1] "membership"  "coeff"       "memb.exp"    "clustering"
 [5] "k.crisp"     "objective"   "convergence" "diss"
 [9] "call"        "silinfo"
```

Here `membership` is the probability profile of belonging to a certain class, and `clustering` is the most likely crisp classification.

It is difficult to show the fuzzy results graphically, but here is one idea:

```
R> ordiplot(ord, dis="si")
R> ordiplot(ord, dis="si", type="n")
R> stars(cfuz$membership, locatio=ord, draw.segm=TRUE, add=TRUE, scale=FALSE, len=0.1)
R> ordihull(ord, cfuz$clustering, col="blue")
```

This uses `stars` function (with many optional parameters) to show the probability profile, and draws a convex hull of the crisp classification. The size of the sector shows the probability of the class membership, and in clear cases, one of the segments is dominant.

[7] All functions in the **cluster** package have stupid names.

[8] Old version of `fanny` may not have all these options and can fail; you should upgrade R to get a new version of the **cluster** package.