

# Cluster Analysis: Tutorial with R

Jari Oksanen

January 22, 2010

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Hierarchic Clustering</b>	<b>1</b>
2.1	Numbers of Classes . . . . .	3
2.2	Clustering and Ordination . . . . .	4
2.3	Minimum Spanning Tree . . . . .	5
2.4	Cophenetic Distance . . . . .	6
<b>3</b>	<b>Interpretation of Classes</b>	<b>7</b>
<b>4</b>	<b>Optimized Clustering at a Given Level</b>	<b>7</b>
4.1	Optimum Number of Classes . . . . .	8
<b>5</b>	<b>Fuzzy Clustering</b>	<b>8</b>

## 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.

I did not have time to lecture on the classification methods. My slides are available on the Internet, and if you wish, I can talk about on classification. If you prefer, you can just go through this tutorial (and preferably my lecture slides), and you will probably get out of the class sooner. The choice is yours.

## 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.

```
> library(vegan)
> data(varespec)
```

Hierarchical clustering (function `hclust`) is in standard R and available without loading any specific libraries. Hierarchical 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` having ecologically useful dissimilarity indices. The default index is Bray–Curtis:

```
> d <- vegdist(varespec)
```

The ecologically useful indices in **vegan** have an upper limit of 1 for completely different sites (no shared species), and use simple 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 cluster. In single linkage (a.k.a. nearest neighbour) 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 among the groups, or the worst among the friends. In average linkage, the distance among 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

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

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

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

You can get back to the single panel mode with

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

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

```
> par(mar = c(3, 3, 1, 1) + 0.1)
```

The single linkage clustering can be found with:

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

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

```
Cluster method : single
Distance       : bray
Number of objects: 24
```

The dendrogram can be plotted with:

```
> 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:

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

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

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

The vertical axes of the cluster dendrogram shows 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.

## 2.1 Numbers of Classes

The hierarchic clustering methods produce all possible levels of classifications. The extremes are to have all observations in a single class, and to have 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:

```
> plot(csin, hang = -1)
> rect.hclust(csin, 3)
> plot(ccom, hang = -1)
> rect.hclust(ccom, 3)
> plot(caver, hang = -1)
> 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. Complete linkage has a tendency to produce compact bunches (*fasciculae*): 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`:

```

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

18 15 24 27 23 19 22 16 28 13 14 20 25 7 5 6 3 4 2 9 12 10 11 21
 1  2  2  2  2  2  2  2  2  1  2  2  2  1  1  1  1  1  3  3  3  3  1  3

```

This gives a numeric classification vector of cluster identities. We can tabulate the numbers of observations in each cluster:

```

> table(cl)

cl
 1  2  3
 8 11  5

```

We can compare two clustering schemes by cross-tabulation:

```

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

cl  1  2  3
 1  8  0  0
 2 10  1  0
 3  4  0  1

```

```

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

cl  1  2  3
 1  8  0  0
 2  0 11  0
 3  4  0  1

```

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

```

> 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`, `isoMDS` (non-metric multidimensional scaling), `prcomp` and `princomp` (both for principal components analysis):

```

> 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** function to overlay classification onto ordination. For distinct, non-overlapping classes convex hulls are practical:

```
> ordihull(ord, cl)
```

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`.

The other clustering results can be seen with:

```
> ordiplot(ord, dis = "si")
> ordihull(ord, cutree(caver, 3))
> ordiplot(ord, dis = "si")
> 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:

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

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

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

## 2.3 Minimum Spanning Tree

In the 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 is closely related to single linkage clustering, which also connects all points with minimum total connecting distance. However, minimum spanning tree really combines points, whereas R representations of single linkage clustering hierarchically connects clusters instead of single points.

Minimum spanning tree (MST) can be found with **vegan** function `spantree`<sup>1</sup>

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

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

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

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.

---

<sup>1</sup>There are many other implementations MST in R, but the implementation in **vegan** probably is the fastest.

```

> plot(mst, type = "t")

Initial stress          : 0.06097
stress after 10 iters: 0.02924, magic = 0.500
stress after 20 iters: 0.02843, magic = 0.500
stress after 30 iters: 0.02842, magic = 0.500

```

## 2.4 Cophenetic Distance

The estimated distance between two points is the level at which they are fused in the dendrogram. 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.

```

> plot(d, cophenetic(csin))
> abline(0, 1)
> plot(d, cophenetic(ccom))
> abline(0, 1)
> plot(d, cophenetic(caver))
> 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.

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

```

> cor(d, cophenetic(csin))

[1] 0.530619

> cor(d, cophenetic(ccom))

[1] 0.7240383

> cor(d, cophenetic(caver))

[1] 0.760622

```

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, but we assume these classes predict environmental conditions.

For interpreting the classes we take into use the **vegan** data set on soil variables:

```
> data(varechem)
```

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

```
> cl <- factor(c1)
```

We can visually display the differences in environmental conditions using `boxplot`:

```
> boxplot(pH ~ cl, varechem, notch = TRUE)
```

The box plot 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 median 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 or any other method for rigid numerical testing of differences among classes (but in general you should avoid nonparametric methods which make stricter assumptions on data than slack parametric methods like ANOVA). In R, ANOVA is a display method for linear models (`lm`):

```
> anova(lm(pH ~ cl, data = varechem))
```

Analysis of Variance Table

Response: pH

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
cl	2	0.32549	0.16275	4.6956	0.02063 *
Residuals	21	0.72784	0.03466		

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

### 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. 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 hierarchical 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 for 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:

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

18 15 24 27 23 19 22 16 28 13 14 20 25 7 5 6 3 4 2 9 12 10 11 21
 2  1  1  1  1  3  1  1  1  2  1  1  1  2  2  2  3  2  3  3  3  3  3  3
```

We can display the results in the usual way:

```
> ordiplot(ord, dis = "si")
> 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 artefacts. 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 but incidentally we do not have any transformation now (if you try, you can see that transformations with `decostand` have capricious effects: obviously the World is not classy in this case).

```
> ccas <- cascadeKM(decostand(varespec, "hell"), 2, 15)
> 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.

## 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 certain class. In the crisp case, it has probability 1 of belonging

to a certain 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.

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.

The fuzzy clustering is provided by function `fanny` in package `cluster`<sup>2</sup>:

```
> library(cluster)
> cfuz <- fanny(varespec, 3)
```

Fuzzy clustering uses Euclidean metric, but again we do with no transformation (you can try what is the effect of transformation).

The function returns an object with the following items:

```
> names(cfuz)

[1] "membership" "coeff"      "memb.exp"   "clustering" "k.crisp"
[6] "objective"  "convergence" "diss"       "call"       "silinfo"
[11] "data"
```

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:

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

This uses `stars` function (with many optional parameters) to show the probability profile, and the 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.

---

<sup>2</sup>All functions in the `cluster` package have stupid names.