Focus on Clustering | 22

by **Jen Schellinck** and **Patrick Boily**, with contributions from **Aditya Maheshwari**

In Chapter 19 (*Machine Learning 101*), we provided a (mostly) math-free general overview of machine learning.

Supervised learning methods can be presented in a formalism which generalizes statistical and regression analysis, and their performance are easy to evaluate; consequently, they have been studied extensively and often form the backbone of machine learning training.

On the other hand, apart from a select few classical models, **unsupervised learning** tasks are not usually presented with quite the same depth, primarily due to the vagueness which infect their core – some of the concepts are defined **ambiguously**; results validation is at times **elusive**, and the actionable applications of the outcomes are not always clear.

The interest in such methods and tasks (clustering and segmentation, association rules mining, link profiling, etc.) is mounting, however, with the recent advances in artificial intelligence and machine learning research. In this chapter, we describe various clustering algorithms, and discuss related issues and challenges.

This is a continuation of the treatment started in Chapter 20 (*Regression and Value Estimation*) and is a companion piece to Chapter 21 (*Focus on Classification and Supervised Learning*).

22.1 Overview

We introduced some of the basic notions of unsupervised learning in Chapter 19, *Introduction to Machine Learning*; in this chapter, we review some of these concepts in the context of clustering, discuss the problems of validation and model selection, and present some simple and sophisticated clustering algorithms.

22.1.1 Unsupervised Learning

In supervised learning (SL), we differentiate a dataset's **response variables** $Y_1, \ldots, Y_m$ from its **predictor variables** $X_1, \ldots, X_p$. Which variables are predictors and which are responses depend on the context – for some questions, a given variable could be a predictor, for others, a response.

Unsupervised learning tasks do away with the responses altogether, which means that **prediction** is off the table; the variables that would have been deemed response variables in a SL framework are not necessarily removed from the dataset during the analysis – they are simply not viewed as an outcome to predict, and the predictor variables are just observation **features**.

In UL, the objective is to **identify** and **uncover interesting insights** about the dataset and the system that it represents (see Section 14.2.2, *Information Gathering*), such as:

- informative ways of visualizing the dataset (often associated with *Feature Selection and Dimension Reduction*, see Chapter 23);
- highlighting subgroups among the dataset's variables or observations (clustering), or
- finding links between variables (association rules mining, link profiling, etc.), say.

22.1.2 Clustering Framework

Clustering consists of a large family of algorithms and methods used to discover so-called **latent groups** in the datasets – natural groups that exist but have not been identified or labeled as such.

Clustering is a **subjective** analytical task; unlike classification and regression, clustering analysis does not have as "simple" a goal as predicting a response for a new observation based on historical data patterns, and there is no "solution key" against which to compare analysis results.

Applications:

- finding subgroups of breast and/or prostate cancer patients based on their gene expression measurements or their socio-demographic characteristics in order to better understand the disease and potential treatment side-effects;
- grouping products in an online shop based on ratings and reviews assigned by customers, or grouping customers based on their purchasing history, in order to make product recommendations;
- finding documents that apply to search queries, and finding similar queries to those entered by a user to increase the odds of finding the documents they are really looking for;
- identifying population segments to test various incentives for vaccination;
- etc.

In each of these cases, the **number** of these latent groups is unknown (and can in fact be taken as a true unknown of the problem). The **subjectivity** of unsupervised learning tasks may seem to be an insurmountable flaw: analysts attempting to find latent groups in a dataset, say, may obtain a different number of such groups, or assign different observations to their groups if their numbers are identical, without one of them being necessarily "wrong".[1]

In spite of this, clustering is a popular analytical task, in part because it is much easier (and cheaper), typically, to obtain **unlabeled data** than it is to obtain labeled data (against which supervised methods could be evaluated). A **cluster** is a subset of observations that all have something in common – they are **similar**, according to some measure of similarity.

[1] Although it is conceivable that some of them could produce **sub-optimal** groups; see Section 22.3 for a detailed discussion on this topic.

Furthermore, a cluster's observations should be **dissimilar** to other clusters' observations.

Clusters do not necessarily need to be **disjoint** (as in so-called **hard** clustering) – in some cases, it might be sufficient to quantify the likelihood or the degree to which an observation belongs to a cluster (**soft** clustering).

The choice of a **similarity/dissimilarity measure** is also entirely **subjective**; there are contexts for which **proximity** could be used as a decent proxy for similarity, and others where it could not. Even in the former case, a **distance measure** (metric) has to be selected, and infinitely many choices are available to analysts.

Without **domain-specific considerations** (this requires thorough data and context understanding), the choice of measure is arbitrary; but understanding the data and the context does not guarantee that all reasonable analysts would agree on such a measure.

For instance, in any group of human beings, which of

> age, ethnic background, gender, postal code, sexual orientation, linguistic abilities, mathematical skills, career, social class, political affiliation, operating system preferences, educational achievements, hockey club fandom, etc.

is responsible for separating its members into "US" vs. "THEM" groups? Is it some combination of these characteristics? Are the groups fixed? Is everybody in the "US" group based on age also in the "US" group based on "gender"?

We could bypass the problem by creating more groups; given an age group and gender, we could create the clusters: "same age group and gender" (US), "same age group, different gender" (THEM$_1$), "different age group, same gender' (THEM$_2$)', "different age group and gender" (THEM$_3$).

It is clear how the process can be expanded to include more combinations of feature levels, but at the price of introducing an ever increasing number of clusters – how many "THEM" groups are too many for analysts or human brains to process?

Clustering algorithms are designed to try to model various aspects of this problem, but the latter's complexity gives rise to an enormous number of algorithms: at least 100 have been published, as of January 2022 [175]. Most of these belong to one of six main families [4]:

- **partitional** ($k-$means and variants, CLARA, etc.);
- **hierarchical** (AGNES, DIANA, BIRCH, etc.);
- **density-based** (DBSCAN, DENCLUE, OPTICS, etc.);
- **connectivity-based** (spectral and variants, etc.);
- **grid-based** (GRIDCLUS, STING and variants, etc.);
- **model-based** (mixture models, latent Dirichlet allocation, expectation-maximization, etc.).

As is the case for all analytical methods, some modifications are required when dealing with "**Big Datasets**", for **high-dimensional data**, or for specific **types of datasets**, such as stream data, network data, categorical data, text and multimedia data, time series data, and so on. **Ensemble methods**, which combine various clustering results, can also prove useful.

Distance, Similarity, and Dissimilarity Measures Although the choice of how to interpret and compute **similarity** between observations is, to all intents and purposes, completely up to the analysts, all such measures must satisfy certain properties: they must take on

- large values for similar objects, and
- small (or even negative) values for dissimilar objects.

Dissimilarity measures function in the opposite manner. The **kernel functions**[2] of machine learning (see Section 21.4.2) are examples of similarity (or dissimilarity) measures, most notably the **Gaussian** (or radial) kernel

$$K_\gamma(\mathbf{x},\mathbf{y}) = \exp(-\gamma\|\mathbf{x}-\mathbf{y}\|_2^2),$$

for a given $\gamma > 0$, for which points near one another (in the $\|\cdot\|_2$ sense) have a similarity measure $w = K(\mathbf{x},\mathbf{y}) \approx 1$ (and thus are **similar**), and points far from one another have a similarity measure near 0 (and thus are dissimilar).

Some similarity measures are derived from **distance (metrics)** functions $d : \mathbb{R}^p \times \mathbb{R}^p \longrightarrow \mathbb{R}_0^+$, with special properties:

1. $d(\mathbf{x},\mathbf{y}) = 0 \iff \mathbf{x} = \mathbf{y}$;
2. $d(\mathbf{x},\mathbf{y}) \geq 0$ for all $\mathbf{x},\mathbf{y} \in \mathbb{R}^p$;
3. $d(\mathbf{x},\mathbf{y}) = d(\mathbf{y},\mathbf{x})$ for all $\mathbf{x},\mathbf{y} \in \mathbb{R}^p$;
4. $d(\mathbf{x},\mathbf{y}) \leq d(\mathbf{x},\mathbf{z}) + d(\mathbf{z},\mathbf{y})$ for all $\mathbf{x},\mathbf{y} \in \mathbb{R}^p$.

In effect, distances are positive-definite symmetric functions $\mathbb{R}^p \times \mathbb{R}^p \longrightarrow \mathbb{R}_0^+$ satisfying the **Triangle Inequality**. Commonly used distances include the:

- **Euclidean** distance $d_2(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|_2$;
- **Manhattan** distance $d_1(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|_1$;
- **supremum** distance $d_\infty(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|_\infty$;
- more general **Minkowski** distance $d_p(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|_p$, for $p \geq 1$, of which the first three examples are special cases;
- and more esoteric distances such as the **Jaccard** distance for binary vectors, the **Hamming** distance for categorical vectors, the **Canberra** distance for ranked lists, the **cosine** distance for text data, **mixed** distances for mixed variables, and so on [254, 255].

Given a distance d, a common construction is to define the associated similarity measures

$$w = \ell - d, \quad w = \exp(-kd^2), \quad \text{or} \quad w = \frac{1}{1+d}.$$

Note that there are similarity measures that cannot be derived from distance measures, however.

[2] Formally, a **kernel** is a symmetric (semi-)positive definite operator $K : \mathbb{R}^p \times \mathbb{R}^p \longrightarrow \mathbb{R}_0^+$. By analogy with positive definite square matrices, this means that $\sum_{i,j=1}^N c_i c_j K(\mathbf{x}_i,\mathbf{x}_j) \geq 0$ for all $\mathbf{x}_i \in \mathbb{R}^p$ and all $c_j \in \mathbb{R}_+$, and $K(\mathbf{x},\mathbf{w}) = K(\mathbf{w},\mathbf{x})$ for all $\mathbf{x},\mathbf{w} \in \mathbb{R}^p$.

Data Transformations Prior to Clustering Prior to clustering, it is crucial that the data be **scaled** (and potentially **centered**) so that none of the variables unduly influence the outcomes, or, as the expression prosaically puts it, so that we do not have to compare apples with oranges – if age in years and height in cm are dataset variables, a 10-unit difference in age is likely to be more significant (in real terms) than a 10-unit difference in height.

Putting everything on a (min, max) scale, for instance, guarantees that relative differences (relative to the distributions of each variables), and not absolute distances, play the important role. However, there are many ways to scale the data, and the scaling approach may have an effect on the clustering results.[3]

Common Difficulties There are issues related to clustering other than the vagueness we have already discussed:

- in many instances, the underlying assumption is that **nearness of observations** (in whatever metric) is linked with **object similarity**, and that **large distances** are linked with **dissimilarity**;
- the **lack of a clear-cut definition** of what a cluster actually is makes it difficult to validate clustering results;
- various clustering algorithms are **non-deterministic**;
- the number of clusters cannot usually be known before the analysis;
- even when a cluster scheme has been accepted as valid, a **cluster description** might be difficult to come by;
- most methods will find clusters in the data even if there are none;
- once clusters have been found, it is tempting to try to "explain" them, but that is a job for domain experts.

22.1.3 A Philosophical Approach to Clustering

In the context of artificial general intelligence,[4] clustering provides a basic way for **intelligences** to structure their experience of the world.

Clustering techniques can allow such machines to identify **object instances** in the world around them and then, on the basis of this identification, to identify or define **types of objects** by grouping together the object instances they have discovered. With this in mind, we can view creating **concepts** as the fundamental purpose of identifying groupings of similar datapoints; these concepts allow an intelligent agent (whether machine or person) to:

- work in **shorthand** when dealing with objects (i.e., it is easier to deal with 10 'cats' than with 10 unique objects), and
- make assumptions about the object instances in a cluster associated with the concept (if an object is a cat, then that object probably likes fish).

If the existence of some "**ground truth**" about what **should** be clustered together (and by extension what should be counted as a concept) can be presupposed, then regardless of what is currently known about that truth, neither the choice of algorithms nor of clustering algorithm parameters is wholly **subjective**, in the psychological sense of the term (where it has

[3] As we are sure you will not be surprised to find out, by this point – that is the way, with clustering: out of the frying pan and into the fire.

[4] Think free-ranging robots, roughly speaking.

the connotation of "coming from a person's experience", which tends to indicate that whatever it is that is being talked about does not exist separate from such an experience); choosing one algorithm over another (or one set of parameters over another) may lead to a "better" or "worse" reflection of the underlying ground truth.

But what counts as a ground truth? There are, of course, debates about this in philosophical circles. Suppose that **natural kinds** exist, that is to say, suppose that there is a **privileged** and **objectively essential** way in which objects are **properly grouped** in nature.

This assumption is very commonly made in the sciences, where uncovering or discovering **universal truths** about natural kinds of objects is a major objective. In such a case, natural kinds can count as a ground truth, with some clusterings more closely reflecting this reality than others.

The fact that the ground truth is not known by the clustering agent does not mean that it does not exist, or that it cannot be sought out using various techniques. Indeed, this is arguably what scientists do when they are using the **scientific method**; they do not know, *a priori*, which of their hypotheses are true or which are false, but they nonetheless engage in various techniques to try to get a better sense of what is true and false.

Even if the existence of natural kinds is rejected, it can still be the case that, relative to a particular circumstance, some clustering results are of **higher quality** than others. Or, stated in terms of goals, some clustering results could achieve a stated goal to a greater or lesser extent than others.

This does appear to be more subjective, in the sense that the goal, and the success of the outcome relative to the goal, are both defined by an individual or individuals, rather than being **independent** of them.

Outside of clustering, it is not unusual for people to create **contextual definitions** of what counts as 'good'. Consider as an example the concept of a 'good meal'. What qualifies as a good meal when camping in the backcountry is not the same as what counts as a good meal when staying at a four-star resort. Context matters.

This does not mean that it is impossible to have a bad meal under either of these circumstances, or that anything counts as a good meal – we do not use subjective in the stultifying post-modernist sense that there are no constraints whatsoever and everything is a social construct.[5]

Nonetheless, such situations are difficult to pin down or define in a rigorous fashion. Even if there were some more abstract or subjective sense in which something could be said to be **common to all types** of good meals – or to all types of good clustering results, in this analogy – it is difficult to imagine how this could be stated with any precision. This is, frustratingly, a typically human limitation to dealing with the world.

However, since the underlying objective of machine learning and artificial intelligence is to create machines with abilities similar to those of humans, perhaps it is worth trying to capture this less than rigorous approach within the context of machine learning.

Given this inherent lack of rigour, are there applied situations where clustering is **useful**? More concretely, suppose we desire to cluster furniture, based on data about the furniture. We could make **measurements**

[5] This might be a bit of a straw-man definition of "post-modernist subjectivity", but perhaps not that much of one, in the final analysis; all things being equal, we lean more toward the objective side of things, both in nature and in data analysis.

of various kinds on physical objects, either selected randomly or haphazardly; perhaps, rather than working with the furniture itself, we could use a website catalogue in which each page showcases a particular type of furniture available for sale.

In this scenario, there may be one grouping (created by tagging and linking pages, for instance) of the furniture pages that allows users to **visit the website with maximum efficiency**, and another that helps the store **maximize its sales**.

Moreover, if we believe that natural kinds exist (which, as noted, is debated by philosophers but is a common assumption in science), there might also be one grouping of the furniture that best matches the underlying furniture natural kinds.

When considering outcomes relative to a particular situation, the most appropriate strategy for a particular clustering will depend, broadly speaking, on two considerations:

- the **chosen goal** it is intended to support, and
- the **underlying structure** of the data itself.

The first can be explicitly known and stated, but the second will likely not be known in advance, which leads to numerous **technical issues** when applying clustering algorithms.

A multitude of clustering algorithms can be applied to problems like the website furniture problems described above, and for each of those, many different parameter settings exist. Suppose six different clustering process are carried out in the case of the furniture website example and they generate six (potentially) different clustering outcomes.

Presumably, some will be more effective than others if the objective is to get people to spend a maximum amount of money on the online store. If the objective is to allow customers to make their purchases most quickly, the "optimal" clustering might not be the one that leads customers to spend the most money.

It is difficult to say ahead of time which of the six groupings will be the most effective one, in each of these cases. However, it might be possible to carry out **A/B testing** to determine which one is the most effective, once they have been generated. But can the A/B testing step be avoided?

If the applied goal (e.g. the goal to group furniture pages in order to maximize profits) can be operationalized more directly in terms of similarity and difference, then it can be more directly tied to clustering approaches.

If we think that the best way to increase sales is to have loose clusters where people are forced to browse a certain amount, while being exposed to somewhat similar (but still interesting) pieces of furniture to find what they want, it might be possible to select a clustering approach to generate clusters with this desirable property.

Returning for a moment to the less applied general artificial intelligence scenario introduced earlier, if the fundamental functionality of clustering is viewed as creating concepts, then it would seem to make sense to operationalize this goal in terms of creating groupings where the **observations** in a group are similar to each other and different from those in other groups. In this context, a poor grouping would *de facto* be one where multiple observations are similar to those in other clusters, or very different from those in their own cluster. This could happen if a clustering process runs into technical difficulties, but it can also happen if there is no such **strongly grouped structure** in the data itself.

To eliminate the possibility that the problem is not linked with the chosen clustering procedure, one strategy is to use **multiple clustering techniques**, as well as **multiple parameter settings** for each technique.

If the issue remains, then we could conclude that it is likely that there is no good clustering structure in the data and by extension, in the objects being represented by the data.

Interested readers can get more information on clustering, as well as examples of applications, in [3–5, 156, 159, 160, 162, 175, 219–226, 256–263].

22.2 Simple Clustering Algorithms

We start by briefly discussing two of the simplest clustering algorithms: $k-$**means** and **hierarchical clustering**.[6]

22.2.1 $k-$Means and Variants

One potential clustering objective could be to achieve minimal **within-cluster variation** – observations within a cluster should be very similar to one another, and the total variation over all clusters should be small.

Assume that there are k clusters in the (scaled) dataset

$$\mathbf{X}_{n\times p} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}.$$

Let $C_1, \ldots, C_k$ denote the set of indices in each cluster, so that

$$\{1, \ldots, n\} = C_1 \sqcup \cdots \sqcup C_k \quad \textbf{(hard clustering)};$$

we use the notation $\mathbf{x}_i \in C_\ell$ to indicate that observation i lies in cluster ℓ. The **within-cluster variation** $\text{WCV}(C_\ell)$ measures the degree to which the observations in C_ℓ differ from one another.

The approach is partition-based; we look for a **partition** $\{C_\ell^*\}_{\ell=1}^k$ such that the total within cluster variation is minimized:

$$\{C_\ell^*\} = \arg\min_{\{C_\ell\}} \left\{ \sum_{\ell=1}^k \text{WCV}(C_\ell) \right\}.$$

[6] In this section, we borrow heavily from [3].

The first challenge is that there are numerous ways to define $\text{WCV}(C_\ell)$, and that they do not necessarily lead to the same results;[7] most definitions, however, fall in line with expressions looking like

$$\text{WCV}(C_\ell) = \frac{1}{(|C_\ell| - g)^\mu} \sum_{\mathbf{x}_i, \mathbf{x}_j \in C_\ell} \text{variation}(\mathbf{x}_i, \mathbf{x}_j),$$

where it is understood that $\text{variation}(\mathbf{x}, \mathbf{x}) = 0$.

Common choices for the **variation** include

$$\text{variation}(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 = \sum_{m=1}^{p} (x_{i,m} - x_{j,m})^2$$
$$\text{variation}(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_1 = \sum_{m=1}^{p} |x_{i,m} - x_{j,m}|;$$

these are typically used because of the ease of vectorizing the distance measurements, and not necessarily because they make the most sense in context.

With these choices, if all observations $\mathbf{x}$ within a cluster C are near one another, we would expect $\text{WCV}(C)$ to be small. The values of the parameter μ can be adjusted to influence the cluster sizes.

Traditionally, we use $\mu = 0$ (or $\mu = 1$) and $g = 0$, and the partition problem reduces to

$$\{C_\ell^*\} = \arg\min_{\{C_\ell\}} \left\{ \sum_{\ell=1}^{k} \frac{1}{|C_\ell|^\mu} \sum_{\mathbf{x}_i, \mathbf{x}_j \in C_\ell} \text{variation}(\mathbf{x}_i, \mathbf{x}_j) \right\}.$$

As an optimization problem, obtaining $\{C_\ell^*\}_{\ell=1}^k$ is NP−hard due to the **combinatorial explosion of possible partitions** $\{C_\ell\}_{\ell=1}^k$ when n is large.[8]

Algorithm: k**−Means** We can obtain a partition which is reasonably close to the optimal one,[9] without having to go through all possible partitions:

1. **randomly assign** a cluster number $\{1, \ldots, k\}$ to each observation in the dataset;
2. for each C_ℓ, compute the cluster **centroid**;
3. assign each observation to the cluster whose centroid is **nearest** to the observation;
4. repeat steps 2-3 until the clusters are **stable**.

Three choices need to be made in order for the algorithm to run:

- the **number of clusters** k in step 1;
- the **centroid computation measure** in step 2;
- the **distance metric** used in step 3.

The most common choice of centroid measure for numerical data is to use the vector of means along each feature of the observations in each cluster (hence, k**−means**); using other centrality measures yield different

[7] As one would expect from clustering.

[8] Computing the number of such partitions in general cannot be done by elementary means, but it is easy to show that the number is bounded above by n^k.

[9] Hopefully...

methods (such as $k-$**medians**, for instance). For categorical data, the algorithm becomes $k-$**modes**.

The distance used in step 3 is usually aligned with the centroid measure of step 2 (and with the choice of a variation function in the problem statement): Euclidean for $k-$means, Manhattan for $k-$medians, Hamming for $k-$modes. Variants of these approaches may use a different random initialization step: the first iteration centroids may be selected randomly from the list of observations, say.[10]

Other variants indicate how to process computations in parallel (for Big Data, see Chapter **??**) or for data streams (with an updating rule, see Chapter 24). The algorithm can be shown to converge to a **stable cluster assignment**, but there is no guarantee that this assignment is the **global minimizer** of the objective function; indeed, different initial conditions can find different **local minima**, i.e., different **clustering schemes.**

Example We have worked with the Gapminder data in Chapters 20 and 21; we will use a variant (`gapminder_all.csv`) to illustrate some of the notions in this chapter. The 2011 data contains observations on $n = 184$ countries, for the following variables:

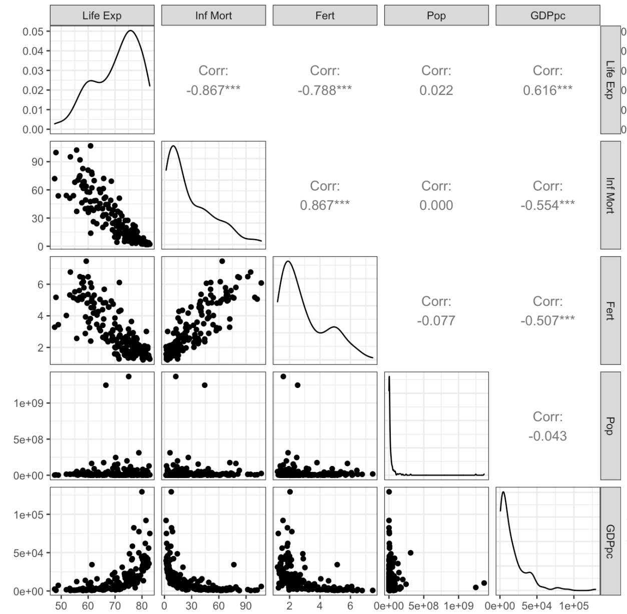
- life expectancy (in years);
- infant mortality rate (per 1000 births);
- fertility rate (in children per woman);
- population (we use the logarithm for clustering), and
- GDP per capita (same).

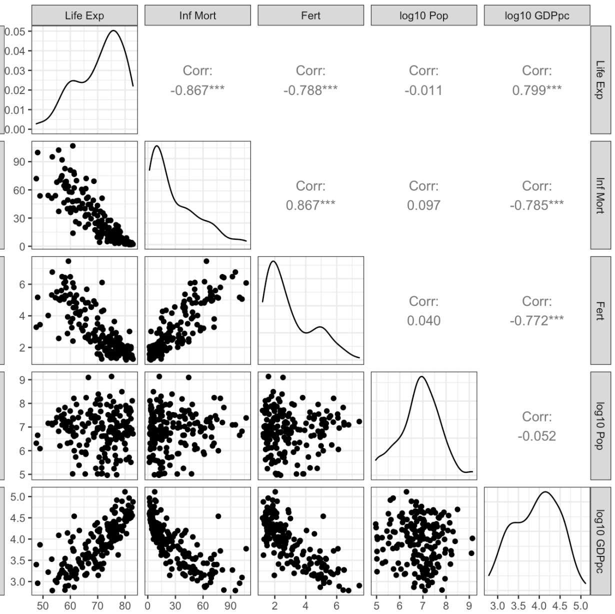
```
library(dplyr)
library(tidyverse) # remove_rownames(), column_to_rownames()

gapminder.SoCL = read.csv("gapminder_all.csv",
    stringsAsFactors=TRUE)

gapminder.SoCL.2011  = gapminder.SoCL |>
  filter(time==2011) |>
  select(geo,
         population_total,
         income_per_person_gdppercapita_ppp_inflation_adjusted,
         life_expectancy_years,
         infant_mortality_rate_per_1000_births,
         children_per_woman_total_fertility) |>
  mutate(log10_pop=log10(population_total),
         log10_gdppc=log10(income_per_person_gdppercapita_ppp_inflation_adjusted)) |>
  na.omit() |>
  remove_rownames() |>
  column_to_rownames(var="geo")

colnames(gapminder.SoCL.2011)<- c("Pop","GDPpc","Life Exp",
         "Inf Mort", "Fert","log10 Pop","log10 GDPpc")
```

A scatter plot of the original and transformed datasets are shown below. We use the logarithm of the population and the logarithm of GDP per capita due to outlying observations in the population variable (China and India).

[10] Unfortunately, the clustering results depend very strongly on the initial randomization – a “poor” selection can yield arbitrarily “bad” (sub-optimal) results; $k-$means++ selects the initial centroids so as to maximize the chance that they will be well-spread in the dataset (which also speeds up the run-time).

```
GGally::ggpairs(gapminder.SoCL.2011[,c(3:5,1:2)])
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)])
```





Throughout, we work with the **scaled** dataset.

```
gapminder.SoCL.2011.s <- data.frame(scale(gapminder.SoCL.2011[,c(3:7)]))
str(gapminder.SoCL.2011.s)
```

```
'data.frame':   184 obs. of  5 variables:
 $ Life.Exp   : num  -1.647 -1.151 0.635 0.377 1.366 ...
 $ Inf.Mort   : num  1.834 3.175 -0.602 -0.498 -0.959 ...
 $ Fert       : num  1.77 2.093 -0.404 -0.981 -0.7 ...
 $ log10.Pop  : num  0.742 0.635 0.921 -0.49 0.595 ...
 $ log10.GDPpc: num  -1.336 -0.304 0.672 -0.164 1.286 ...
```

The following function will allow us to plot the distributions of each of the variables in each of the clusters (plots appearing on the diagonal):

```
library(ggplot2)
my_dens <- function(data, mapping, ..., low = "#132B43",
                    high = "#56B1F7") {
  ggplot(data = data, mapping=mapping) +
    geom_density(..., alpha=0.7)
}
```

We run k−means for $k = 2, 3, 4$ and obtain the results shown in Figure 22.1.

```
set.seed(1) # for replicability
# k=2
gapminder.SoCL.2011.s.k2 = kmeans(gapminder.SoCL.2011.s,2,
       iter.max=250,nstart=1)
```

```
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
    aes(color=as.factor(gapminder.SoCL.2011.s.k2$cluster)),
    diag=list(continuous=my_dens))
table(gapminder.SoCL.2011.s.k2$cluster)
```

```
 1   2
64 120
```

```
# k=3
gapminder.SoCL.2011.s.k3 = kmeans(gapminder.SoCL.2011.s,3,
       iter.max=250,nstart=1)
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
    aes(color=as.factor(gapminder.SoCL.2011.s.k3$cluster)),
    diag=list(continuous=my_dens))
table(gapminder.SoCL.2011.s.k3$cluster)
```

```
 1  2  3
46 84 54
```

```
# k=4
gapminder.SoCL.2011.s.k4 = kmeans(gapminder.SoCL.2011.s,4,
       iter.max=250,nstart=1)
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
    aes(color=as.factor(gapminder.SoCL.2011.s.k4$cluster)),
    diag=list(continuous=my_dens))
table(gapminder.SoCL.2011.s.k4$cluster)
```

```
 1  2  3  4
26 50 61 47
```

The colours (cluster labels) are not used by the clustering algorithm – they are the **outputs**.[11] This next block shows the result of a different initialization for $k = 3$, leading to a different cluster assignment.

```
set.seed(1234) # different initialization
gapminder.SoCL.2011.s.k3 = kmeans(gapminder.SoCL.2011.s,3,
       iter.max=250,nstart=1)
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
    aes(color=as.factor(gapminder.SoCL.2011.s.k3$cluster)),
    diag=list(continuous=my_dens))
table(gapminder.SoCL.2011.s.k3$cluster)
```

```
 1  2  3
56 74 54
```

Another $k-$means example is provided in Section 19.7.3.

[11] The actual cluster label value is entirely irrelevant.

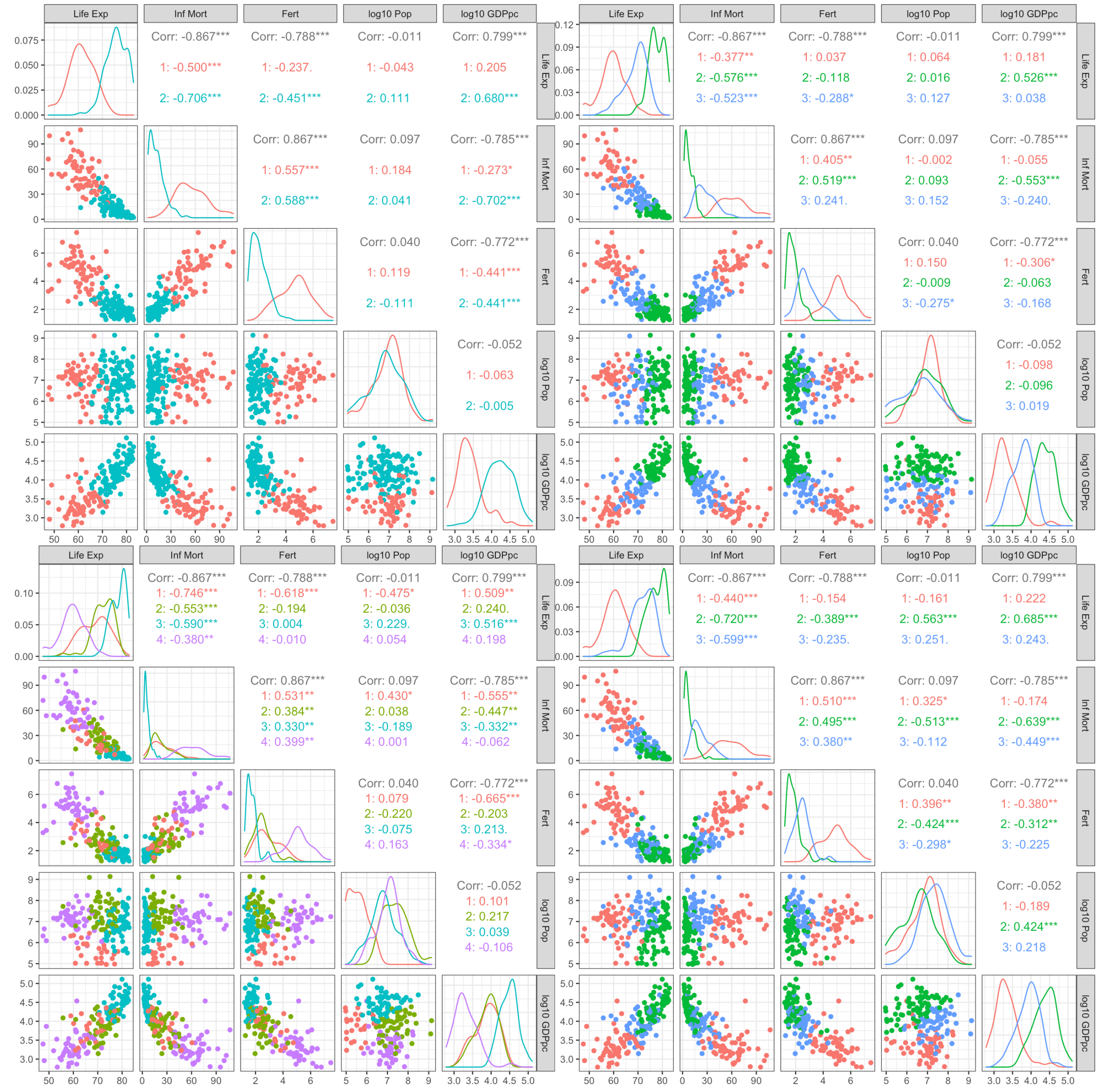


Figure 22.1: Realizations of k−means on the 2011 Gapminder data: $k = 2$ (top left); $k = 3$ (top right); $k = 4$ (bottom left); $k = 3$ with a different seed (bottom right). Are the two $k = 3$ clustering outcomes clearly distinct, to your eye?

22.2.2 Hierarchical Clustering

One of the issues surrounding the use of $k-$means (and its variants) is that nothing in the result of a single run of the algorithm indicates if the choice of k was a good one.[12]

Determining a "good" value of k can only be achieved by repeatedly running the algorithm over a range of "reasonable" values of k (to account for initialization variability), and by comparing the outputs using some of the methods discussed in Section 22.3. This process can be memory- (and time-)extensive.

Hierarchical clustering (HC) can sidestep this difficulty altogether by building a deterministic **global clustering structure** (for a given choice of parameters), from which we can select a specific number of clusters after the algorithm has converged; the advantage of this approach is that if we want to use a different number of clusters, we do not need to re-run the clustering algorithm – we simply read off the new clusters from the global clustering structure.

There are two main conceptual approaches:

- **bottom-up/agglomerative** (AGNES) – initially, each observation sits in its own separate cluster, which are then merged (in pairs) as the hierarchy is climbed, leading (after the last merge) to a large cluster containing all observations;
- **top-down/divisive** (DIANA) – initially, all observations lie in the same cluster, which is split (and re-split) in pairs as the hierarchy is traversed downward, leading (after the last split) to each observation sitting in its own separate cluster.

Both approaches are illustrated in Figure 22.2: the first one is an illustration of AGNES and DIANA. The corresponding hierarchical structure is shown in the second image; the dendrogram in the third.

In theory, the two approaches are equivalent;[13] in practice, we use AGNES over DIANA for anything but small datasets as the former approach runs in polynomial time,[14] whereas the latter runs in exponential time.

With AGNES, the **clustering dendrogram** is built starting from the leaves, and combining clusters by pairs, up to the root, as in Figure 22.3.

Algorithm: AGNES The **global** clustering structure is built as follows:

1. each observation is assigned to **its own cluster** (there are n clusters, initially);
2. the two clusters that are the least dissimilar are merged into a **supra-cluster**;
3. repeat step 2 until **all of the observations** belong to a **single** large merged cluster (with n observations).

Three decisions need to be made in order for the algorithm to run:

- the choice of a **linkage strategy** in steps 2 and 3;
- the **dissimilarity measure** used in step 2;
- the **dissimilarity threshold** required to "cut" the dendrogram into clusters.

[12] The results might look good on a 2-dimensional representation of the data, but how do we know it could not look better?

[13] They produce the same hierarchical structure given a similarity metric and a **linkage strategy** (more on this later).

[14] With respect to the number of observations.

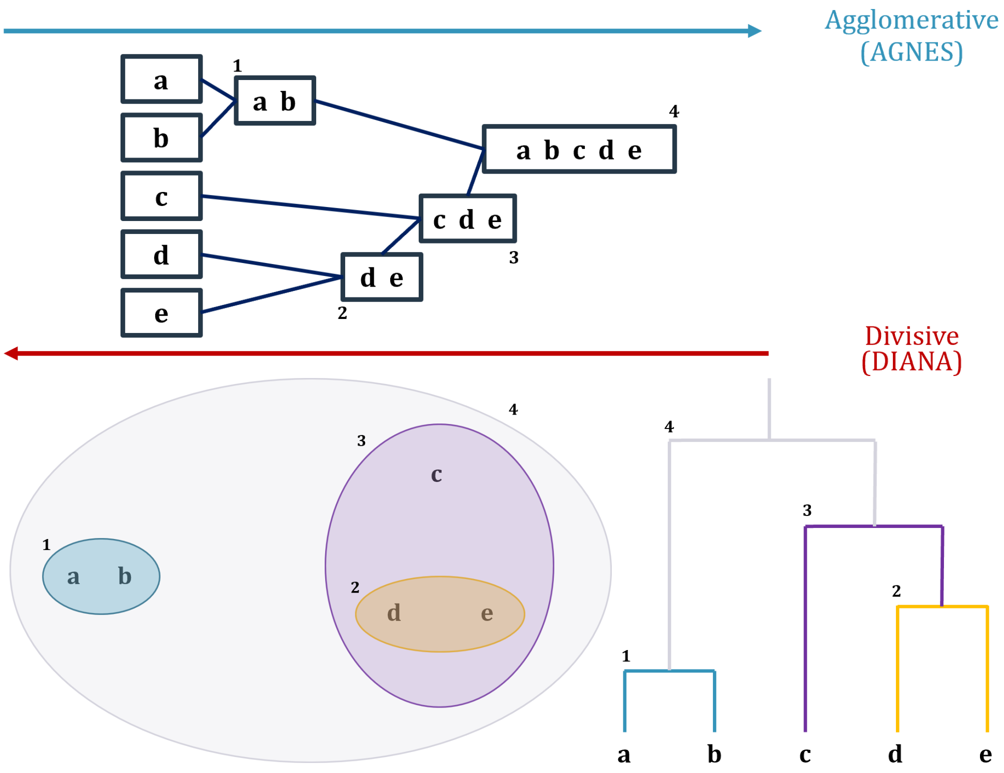


Figure 22.2: Conceptual representation of AGNES and DIANA on a simple artificial dataset.

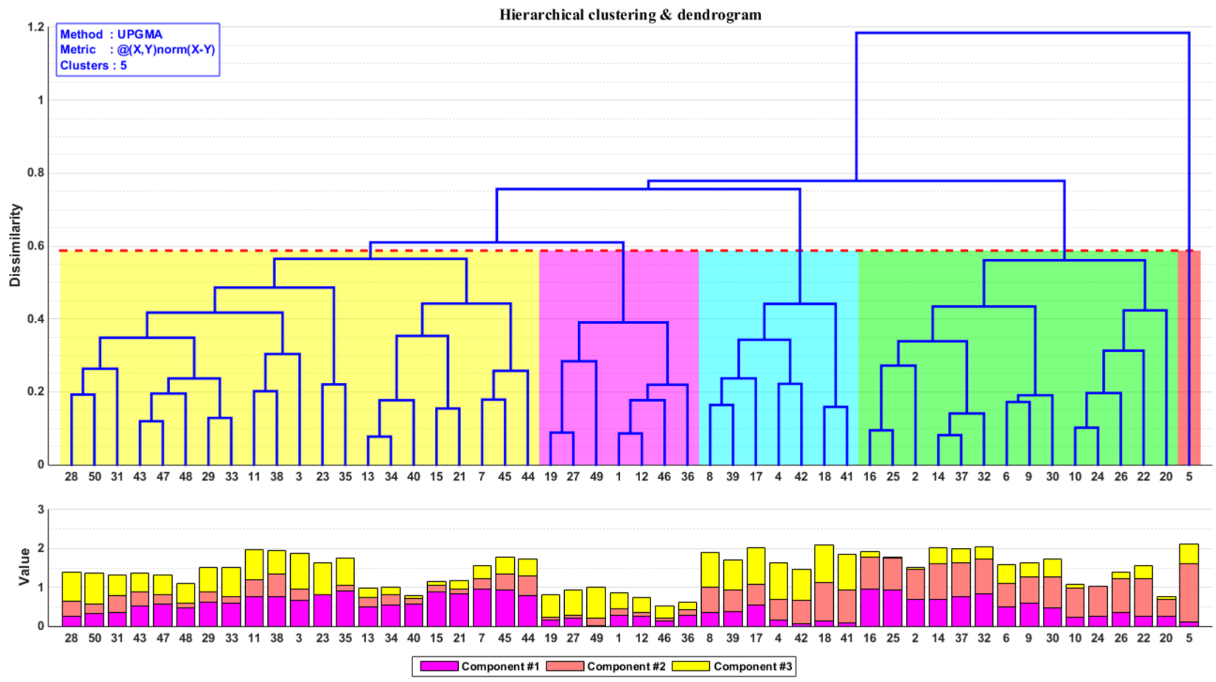


Figure 22.3: Cluster dendrogram for the hierarchical cluster structure of a dataset with 50 observations and 3 variables, with average linkage (UPGMA) and using the Euclidean distance as the dissimilarity measure [author unknown]. The dendrogram is cut at a dissimilarity level ≈ 0.6 so that 5 clusters emerge (ordered and coloured in red, magenta, blue, green, and red); the observations profiles are shown in the stacked bar chart and provide potential descriptions of the clusters (magenta = small total height, with mostly dominant 3rd components, say). Based on the profiles, we might also have elected to cut a slightly lower dissimilarity level (≈ 0.55), so that the yellow and green clusters would have been further split into two clusters apiece (between observations 35 and 13, and 30 and 10, perhaps?).

In Figure 22.3, the dataset is first split into $n = 50$ clusters; observations 13 and 34 are then found to be most similar, and merged into a single cluster, and the 50 observations are grouped into 49 clusters. The next two observations which are most similar are 14 and 37, which are themselves merged, so that there are 48 clusters at that level.

The process is continued until all observations are merged into a single cluster, leading to the global clustering structure (**clustering dendrogram**) for the dataset. In order to obtain actual clusters (as opposed to the global structure), we cut the dendrogram at the selected dissimilarity level, with the resulting groups of observations yielding the dataset clusters (5, in the example).

Increasing the dissimilarity threshold decreases the number of clusters, and *vice-versa*.

Linkage Strategy In the first AGNES stage, we compare all pairs of observations to determine which two are least dissimilar; these are then merged into a cluster.[15]

In the second stage, we must also compare each of the non-merged observations with a **cluster** of two observations to determine their dissimilarity (the other dissimilarities have been computed in the first stage and do not need to be computed anew).

In subsequent stages, we might also need to compare two clusters with any number of observations for overall similarity. How can this be achieved? Let A and B be clusters in the data, with $|A| = n_A$, $|B| = n_B$. The **dissimilarity** between A and B can be computed in multiple ways:

- **complete linkage** – the **largest** dissimilarity among all pairwise dissimilarities between the observations in A and those in B (n_An_B computations);
- **single linkage** – the **smallest** dissimilarity among all pairwise dissimilarities as in complete linkage;
- **average linkage** – the **average** dissimilarity among all pairwise dissimilarities as in complete (or single) linkage;
- **centroid linkage** – the dissimilarity between the **centroids** of A and B (found using whatever method is appropriate for the context);
- **Ward's method** (and its variants) uses any reasonable **objective function** which reflects domain knowledge [264, 265];
- etc.

The choice of a **linkage strategy** (and of a dissimilarity measure) affects not only how clusters are compared and merged, but also the **topology** (shape/structure) of the resulting dendrogram (are the clusters tight, loose, blob-like, etc.). The various linkage strategies are illustrated in Figure 22.4, assuming Euclidean dissimilarity.

Example We show the results of hierarchical clustering on the Gapminder 2011 data, using complete linkage and Euclidean dissimilarity, and Ward D linkage and the maximum dissimilarity. In each case, we consider $k = 2, 3, 4$ clusters.[16] The cluster charts are in Figure 22.5.

[15] With n observations, there are $1 + \cdots + (n-1) = \frac{(n-1)n}{2}$ such pairs.

[16] **Reminder:** we work on the scaled data. Assume that `library(ggplot2)` has already been loaded.

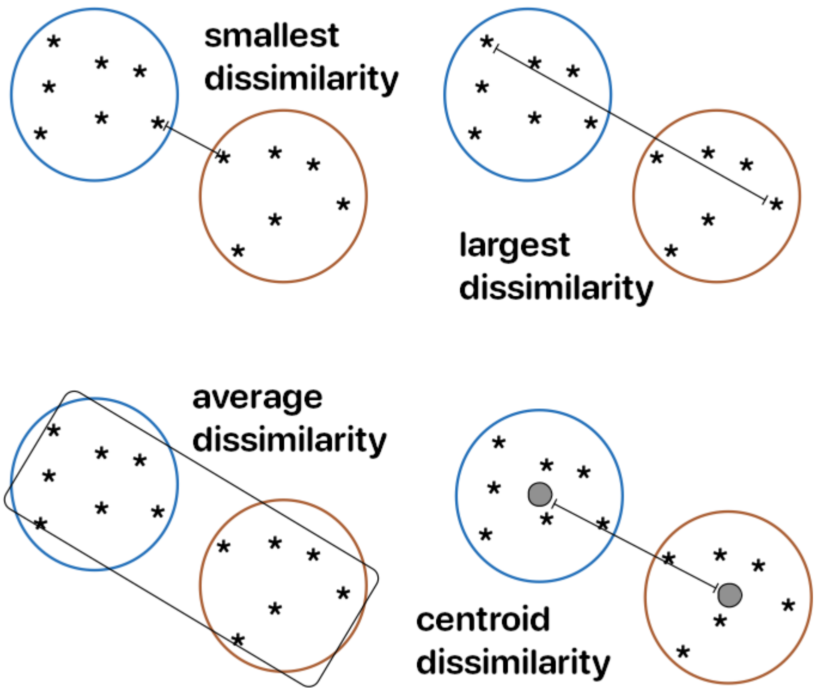


Figure 22.4: Conceptual notions of linkage, assuming Euclidean dissimilarity.

We first need to create the AGNES structure for the data using complete linkage and Euclidean dissimilarity.

```
# global, complete, Euclidean
par(cex=0.45)
hclust.gapminder.SoCL.2011 <- hclust(dist(gapminder.SoCL.2011.s))
plot(hclust.gapminder.SoCL.2011, hang = -1, cex=0.7,
     main = "Gapminder 2011 Data \n HC - Global Structure
     - Euclidean - Complete", ylab="")
```

Let us find the breakdown for $k = 2, 3, 4$ clusters for complete linkage and Euclidean dissimilarity.

```
# k=2, complete, Euclidean
par(cex=0.45)
hclust.gapminder.SoCL.2011 |> as.dendrogram() |>
  dendextend::set("branches_k_color", value = c("red", "blue"), k = 2)  |>
  plot(main = "Gapminder 2011 Data \n HC - 2 clusters - Euclidean - Complete")

hclust.gapminder.SoCL.2011 |> as.dendrogram() |>
    dendextend::rect.dendrogram(k=2, border = 8, lty = 5, lwd = 2, lower_rect=0)

GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
    aes(color=as.factor(cutree(hclust.gapminder.SoCL.2011, k = 2))),
    diag=list(continuous=my_dens))

table(cutree(hclust.gapminder.SoCL.2011, k = 2))
```

```
 1   2
66  118
```

```
# k=3, complete, Euclidean
par(cex=0.45)
hclust.gapminder.SoCL.2011 |> as.dendrogram() |>
  dendextend::set("branches_k_color",
      value = c("red", "blue", "darkgreen"), k = 3)  |>
  plot(main = "Gapminder 2011 Data \n HC - 3 clusters -
      Euclidean - Complete")

hclust.gapminder.SoCL.2011 |> as.dendrogram() |>
    dendextend::rect.dendrogram(k=3, border = 8, lty = 5,
        lwd = 2, lower_rect=0)

GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
    aes(color=as.factor(cutree(hclust.gapminder.SoCL.2011,
        k = 3))), diag=list(continuous=my_dens))

table(cutree(hclust.gapminder.SoCL.2011, k = 3))
```

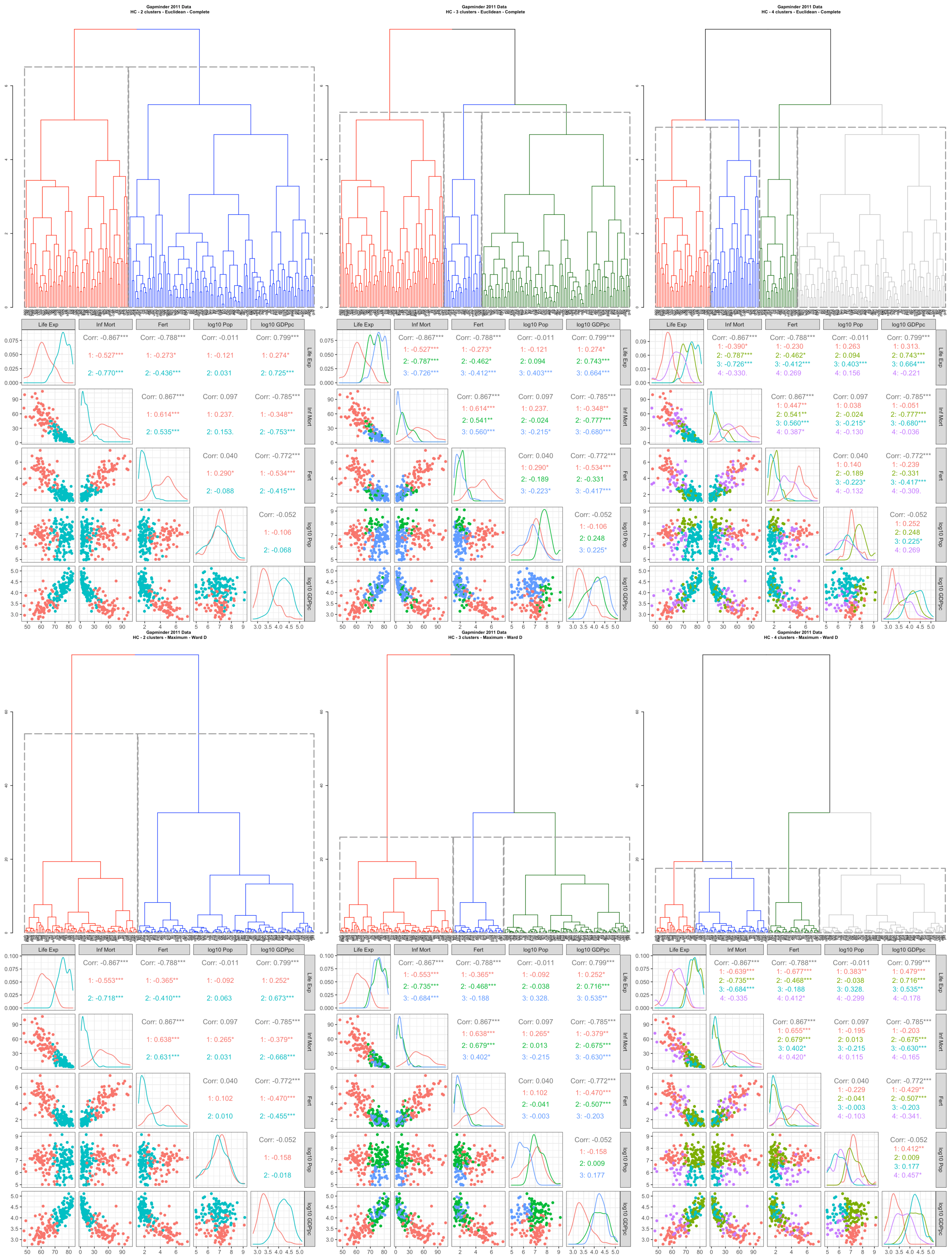
```
 1  2  3
66 24 94
```

```
# k=4, complete, Euclidean
par(cex=0.45)
hclust.gapminder.SoCL.2011 |> as.dendrogram() |>
  dendextend::set("branches_k_color",
      value = c("red", "blue", "darkgreen", "gray"), k = 4)  |>
  plot(main = "Gapminder 2011 Data \n HC - 4 clusters -
      Euclidean - Complete")

hclust.gapminder.SoCL.2011 |> as.dendrogram() |>
    dendextend::rect.dendrogram(k=3, border = 8, lty = 5,
        lwd = 2, lower_rect=0)

GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
    aes(color=as.factor(cutree(hclust.gapminder.SoCL.2011,
        k = 4))), diag=list(continuous=my_dens))

table(cutree(hclust.gapminder.SoCL.2011, k = 4))
```

```
 1  2  3  4
35 24 94 31
```

Notice how the number of observations in each cluster follows a hierarchical structure: when we go from $k = 2$ to $k = 3$ clusters, the new cluster is a subset of one of the old clusters (and similarly when we go from $k = 3$ to $k = 4$).

We can see how the results change when we use a different distance metric (maximum) and a different linkage strategy (Ward D): the line `hclust.gapminder.SoCL.2011 <- hclust(dist(gapminder.SoCL.2011.s))`is substituted throughout by `hclust.gapminder.SoCL.2011.2 <- hclust(dist(gapminder.SoCL.2011.s,method="maximum"), method="ward.D")`.


Gapminder 2011 Data
HC - 2 clusters - Euclidean - Complete

Gapminder 2011 Data
HC - 3 clusters - Euclidean - Complete

Gapminder 2011 Data
HC - 4 clusters - Euclidean - Complete

Gapminder 2011 Data
HC - 2 clusters - Maximum - Ward D

Gapminder 2011 Data
HC - 3 clusters - Maximum - Ward D

Gapminder 2011 Data
HC - 4 clusters - Maximum - Ward D

Figure 22.5: Realizations of hierarchical clustering (AGNES) on the 2011 Gapminder data: complete linkage, Euclidean dissimilarity for $k = 2, 3, 4$ clusters (top 2 rows); Ward D linkage, maximum dissimilarity for $k = 2, 3, 4$ clusters (bottom 2 rows).

22.3 Clustering Evaluation

Hierarchical clustering (HC) and k−means (and its variants) both attempt to separate the data into **natural groups**, using different conceptual approaches; k−means tries to minimize within-cluster variation while HC builds a global clustering structure.

We have discussed some of their shortcomings in the previous section; the fact that they may yield different clustering outcomes depending on the choices made along the way (initialization, similarity/dissimilarity measures, linkage strategy, number of clusters, etc.) reinforces the notion that unsupervised learning is **difficult**.

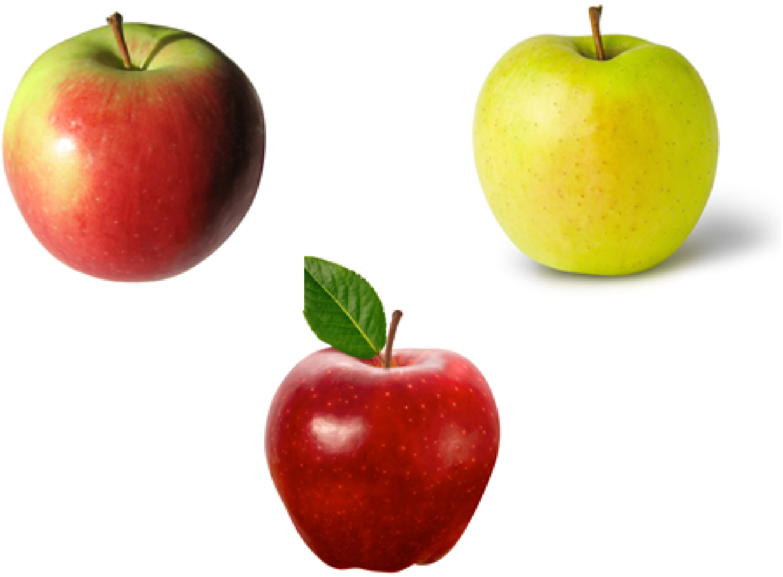
We will discuss advanced algorithms that sidestep some of these issues in Section 22.4, but we make an important observation in the meantime: a hallmark of clustering is that whenever a new approach manages to overcome a previously-identified difficulty, it usually does so at the cost of introducing holes in hitherto sound aspects.

We may thus not be able to ever find a "best" clustering approach/outcome,[17] but is it possible to identify which of several clustering scheme is "**optimal**" (or best-suited for an eventual task)?

22.3.1 Clustering Assessment

In machine learning, clustering is defined as grouping objects based on their over-all similarity (or dissimilarity) to each other.[18] It can be tempting to focus on just one or two attributes (especially for visual or "eyeball" clustering), but keep in mind that even if we were to focus on one or two particular attribute, all of the other attributes must still come along for the ride.

For instance, consider the objects shown below.



[17] An updated take on the No Free Lunch theorem, perhaps? [235]

[18] Note that each object has multiple dimensions, or attributes available for comparison.

What is the same about these objects? What is different? Do they belong in the same group? If not, how many groups are there?

As a start, they are all pictorial representations of apples;[19] they all possess stems, and appear to be of similar size. On the other hand, two of them are (mostly) red while the other is green(ish); one of the stems has a leaf while the other two do not, and one of them is spherical, while the other two are "tapered" at the bottom.

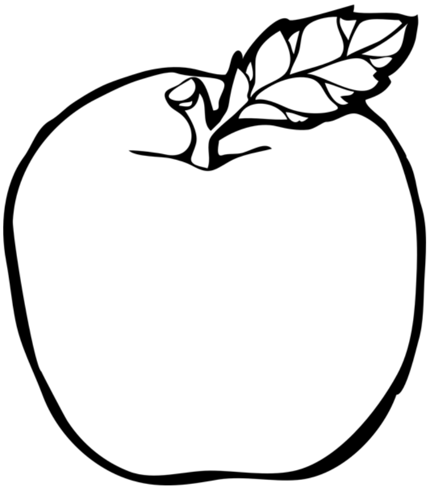
While we do recognize them all as apples, we could make the argument that each of them is unlike the other two (and thus also that each of them is similar to exactly one of the other two).

Fruit Image Dataset In order to appreciate the challenges presented by clustering validation, it will be helpful to relate the concepts to something tangible. We will explore some of these notions through an artificial dataset of 20 fruit images (see Figure 22.6):

- are there right or wrong clusterings of this dataset?
- are there multiple possible 'natural' clusterings?
- could different clusterings be used for different tasks?
- will some clusterings be of (objectively) higher quality than others?

Key Notions At a fundamental level, clustering relies on the notion of **representativeness**; ideally, the essence of **instances** (observations) in a cluster would be faithfully captured by the cluster **concept** (examplar, representative), and differentiated from those of other clusters by the same token.

As an example, the image below is a concept for "apples":



as is the Mirriam-Webster definition:

> "The fleshy, usually rounded red, yellow, or green edible pome fruit of a usually cultivated tree (genus Malus) of the rose family."

[19] While we cannot forget that they are not actual apples, we will assume that this is understood and simply refer to the objects as fruit, or apples.

Of course, this is not *all* that an apple is, but most people who have seen or eaten an apple at some point in the past will have no trouble recognizing what is being alluded to by the concept, even if the corresponding mental image differs from one person to the next.

The cluster concepts offer a **generalized representation** – they capture "something" of their corresponding cluster's instances. For a given cluster, then, can we clearly identify a concept capturing its (and solely its) essence? If so, does that make the entire clustering scheme a good one?

For machine learning purposes, we use **signature vectors** to represent **instance properties**. Each vector element represents an instance **attribute**; the element's value is the **measured value** of the corresponding object's property (for instance, the colour of the apple).

The apple below, as an example, could perhaps be described by the signature vector

$$(12, 9.12, \text{tapered}, \text{golden delicious}),$$

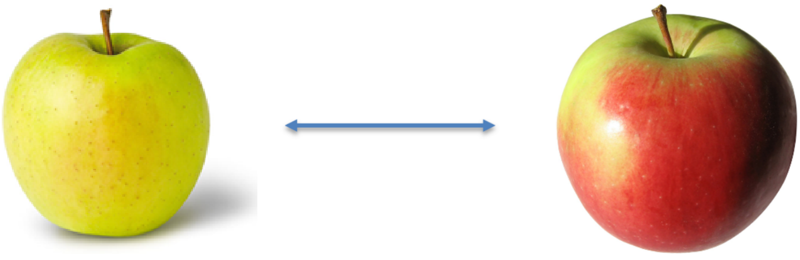
where the components are the instance's **colour** (ordinal), **height** (continuous), **shape**, and **variety** (both categorical).[20]

Signature vectors are used to compare objects (**instance-to-instance relationships**); such comparisons could yield, among others, a measure of **similarity** between instances.

While similarity can be measured against a single **dimension** (component), the comparisons of interest for clustering task require an overall similarity measure, **across all dimensions**. We would compare the two apples below, say, by comparing their signature vectors

$$\mathbf{v}_1 = (12, 9.12, \text{tapered}, \text{golden delicious})$$
$$\mathbf{v}_2 = (2, 10.43, \text{spherical}, \text{macintosh})$$

with the help of some similarity measure $w(\mathbf{v}_1, \mathbf{v}_2)$.[21]



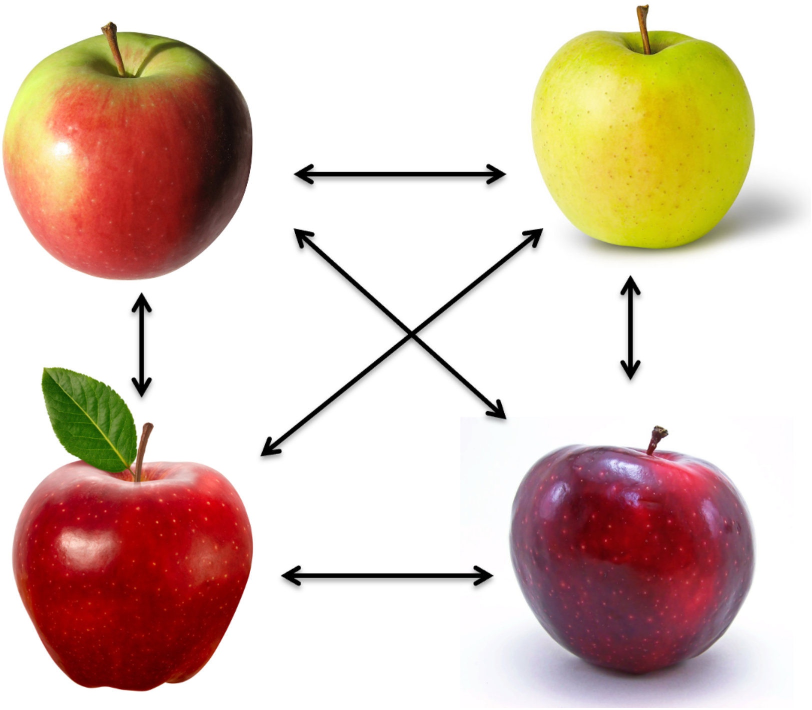
[20] An important consideration, from a general data science perspective, is whether the signature vector provides a **sufficient description** of the associated object or whether it is too crude to be of use. This is usually difficult to ascertain prior to obtaining analysis results, and comparing them to the "reality" of the underling system (see Chapters **??** and 14 for details).

[21] Keep in mind that different similarity measures may yield various results, in some cases showing the two apples to be similar, in others to be dissimilar.

As we have discussed in Section 22.1, the use of a **distance measure** (or metric) is a popular strategy for defining how similar (or dissimilar) two objects are to each other. Importantly, a distance takes into account all of the properties of the objects in question, not just a few of them. Traditionally, only numeric attributes are allowed as input (see Chapter 27 for an in-depth discussion of distance metrics), but it is technically possible to convert categorical attributes to numeric ones, or to define **mixed distances**.[22]

In the **clustering framework**, we are often interested in all pairwise similarities between objects, not just in the similarity between two objects, which is to say that pairs of objects are not solely interesting in and of themselves, but also **in relation to other pairs of objects**.

In a dataset with 4 objects, for instance, we might require the computation of (up to) 6 pairwise similarities (as shown below).

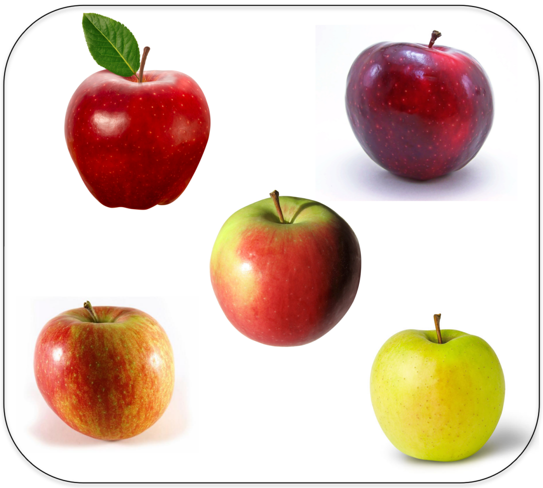


As is the case with objects, **clusters** also have **properties**. These could include:

- the number of instances in a cluster;
- similarity statistics across instances within a cluster (minimum, maximum, average, median, standard deviation, mean absolute deviation, variance, etc.);
- the cluster representative, which may be an actual instance, or an amalgamation of multiple instances (**exemplar**).

How many instances are there in the cluster at the top of the next page, for instance? What pair of observations is most similar? The least similar? What are the similarity values? Which instance is most representative?

[22] While the moniker "distance" harkens back to the notion of Eulidean (physical) distance between points in space, it is important to remember that the measurements refer to the distance between the associated signature vectors, which do not necessarily correspond to their respective physical locations.



We can also define **cluster-to-instance** relationships. A specific instance can be:

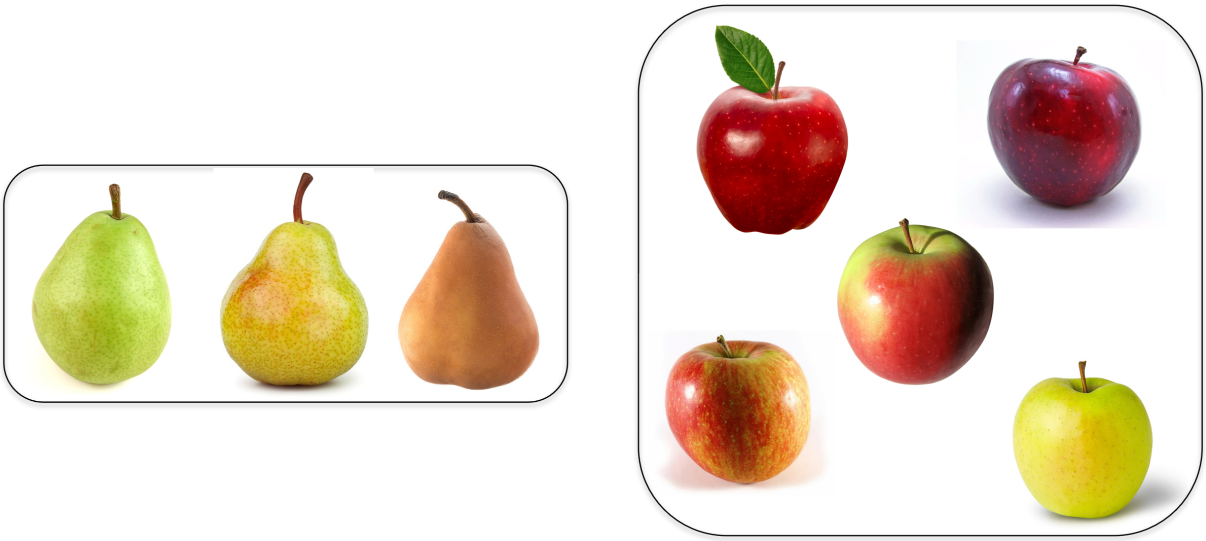
- compared to a cluster representative, and/or
- compared to specific instances in a cluster (as in instance-to-instance relationships), such as the most similar instance or the most distant instance.

This allows for **membership** questions: is the green apple similar to the cluster below? Does it belong in the cluster, or is it most likely to belong to another cluster? Or perhaps to no cluster in particular?

Finally, we might be interested in **cluster-to-cluster** relationships, where we compare cluster-level properties, such as:

- number of instances;
- **within-cluster** similarities;
- cluster representatives.

To these, we can also add **between-cluster** (or across-cluster) similarities, as a way to determine if the instances are notably different from one cluster to the next. This allows for **validity** questions: are the two clusters below significantly different? Should they be joined into a mega-cluster? Does it make sense to have them as separate clusters in the dataset?



How would we qualify the clustering outcome of Figure 22.7, for instance, as it relates to colour, height, and shape? Could there be clusterings of higher quality? Of lower quality? How could this be quantified?

Cluster and instance comparisons can be combined in many different ways, which can then be used to generate a vast number of **clustering validation functions**.

The central cluster validation question becomes: what can these tell us about the quality of a particular clustering outcome relative to some objective criteria about "good" clustering schemes (**internal validation**), to another clustering option (**relative validation**), or to external information (**external validation**)?

Clustering Quality Measures In general, clustering involves two main activities:

- **creating/building** the clusters, and
- **assessing their quality**, individually and as a whole.

From a practical perspective, clustering requires two functions: one which assigns each instance to a cluster,[23] and one which assigns each clustering scheme to a **cluster quality measurement**.[24]

An illustration is provided in Figure 22.8, on an artificial dataset containing two variables, with dissimilarity between instances given by the corresponding Euclidean distance.

[23] Or in the case of soft clustering, assign each instance a "probability" of belonging to each cluster.

[24] The similarity matrix is typically required at both stages.

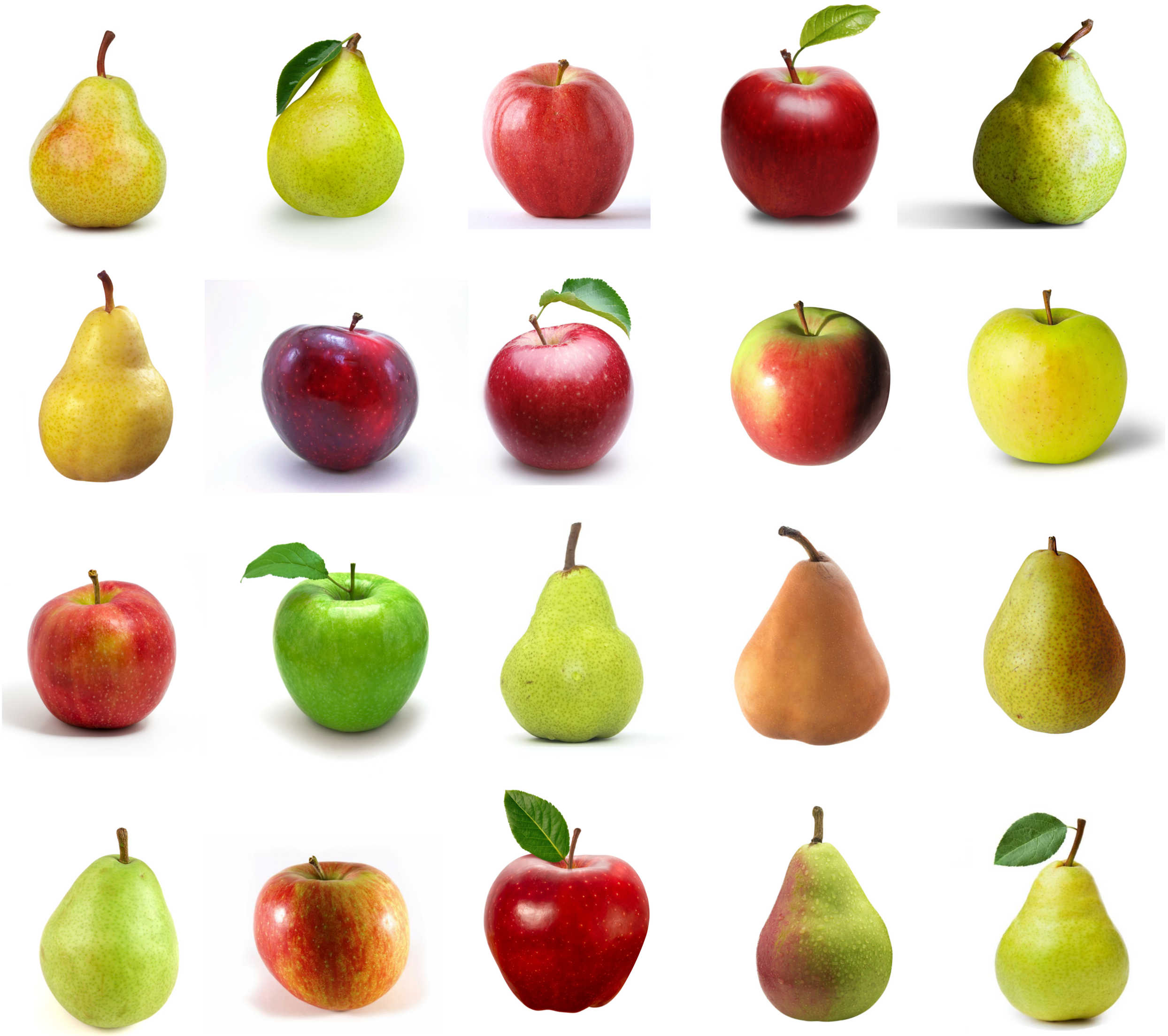


Figure 22.6: Toy dataset with which the key concepts of clustering validation will be illustrated.

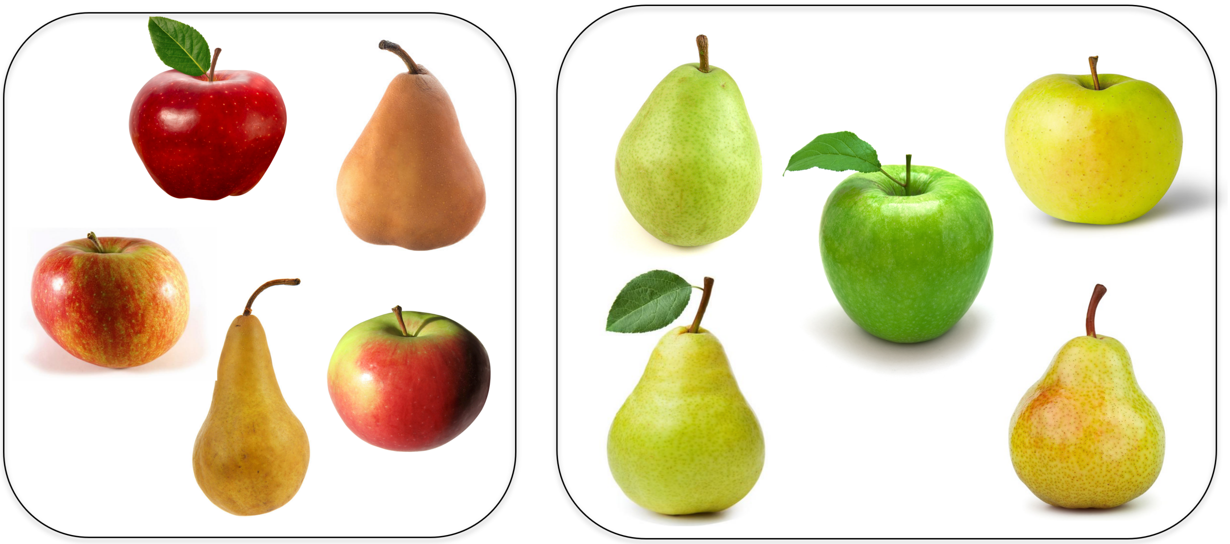


Figure 22.7: Two clusters in a subset of the fruit image toy dataset.

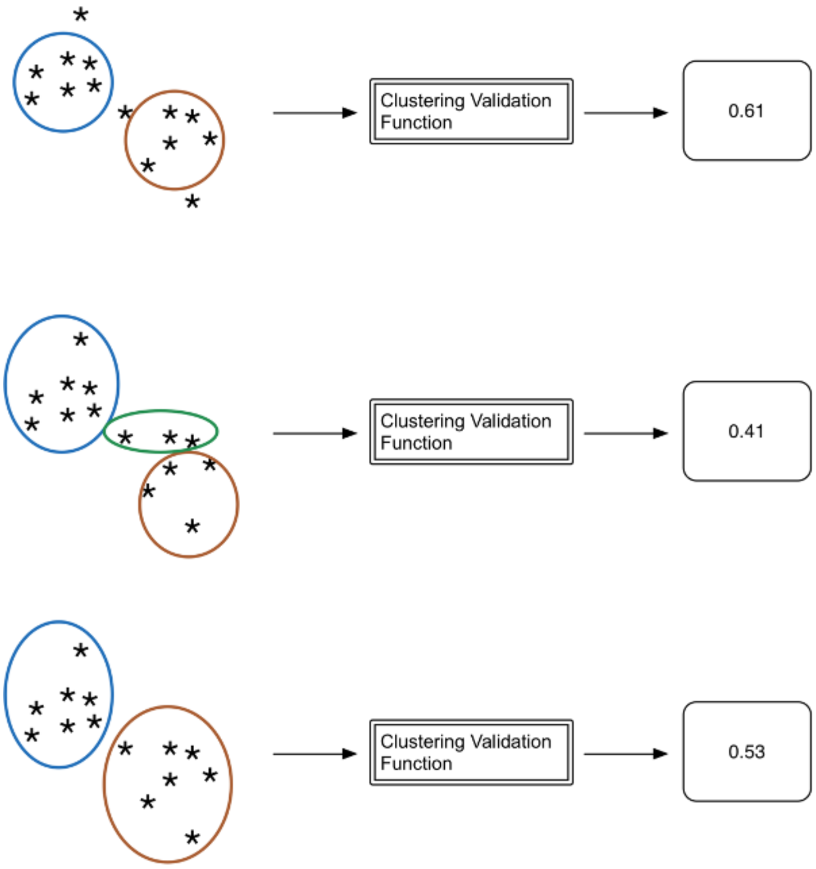

Clustering Validation Function
0.61
Clustering Validation Function
0.41
Clustering Validation Function
0.53

Figure 22.8: Cluster quality measurements on an artificial dataset.

We obtain three different clustering schemes, and their quality is assessed with the help of some clustering validation function:[25]

- **top** – two clusters are found in the data (with outliers), and the quality of the clustering is assessed as 0.61;
- **middle** – three clusters are found (no outliers), with quality assessment at 0.41;
- **bottom** – two clusters are found (no outliers), with quality assessment at 0.53.

With this choice of clustering validation function, the top scheme would be preferred, followed by the bottom scheme; the middle one brings up the rear. We have already mentioned the abundance of clustering algorithms [175]; it will come as no surprise that a tremendous number of clustering validation function in practice as well (for much the same reasons as those discussed in Section 22.1.2).

They are, however, all built out of the basic measures relating to instance or cluster properties we have already reviewed, as illustrated schematically in Figure 22.9:

- **instance properties**;
- **cluster properties**;
- **instance-to-instance relationship properties**;
- **cluster-to-instance relationship properties**, and
- **cluster-to-cluster relationship properties**.

[25] The specifics of that function are not germane to the current discussion and so are omitted.

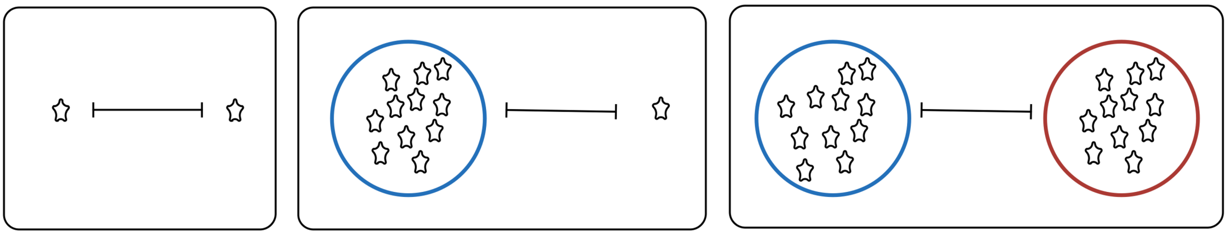
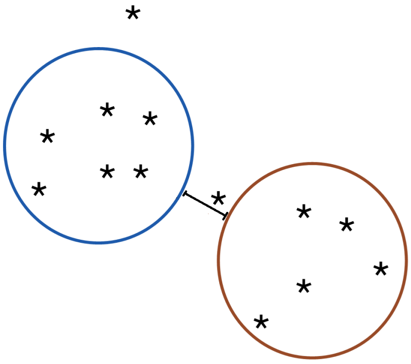


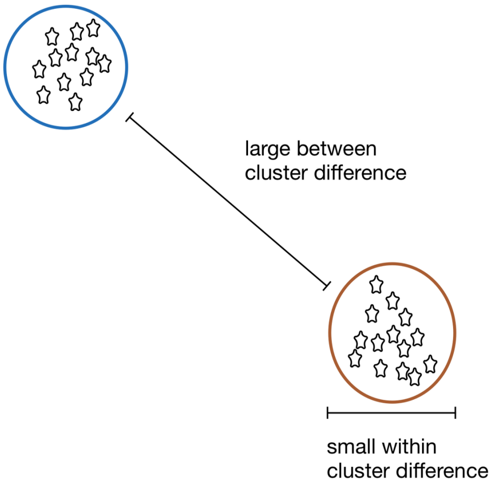
Figure 22.9: Schematic illustrations of various instance/cluster properties and relationships.

Internal Validation Context is quite relevant to the **quality** of a given clustering result. But what if no context is readily available? **Internal validation** methods attempt to measure cluster quality objectively, without context.[26]

We could elect to validate the clustering outcome using only the properties of the obtained clusters, such as, say, the distance between all clusters: if the average between-cluster distance is large, we might feel that there is some evidence to suggest that the resulting clusters provide a good segmentation of the data into **natural groups**, as in the image below.



Alternatively, we might validate cluster quality by tempering the average between-cluster distance against the average within-cluster distance between the instances, which would reward "tight" and "isolated clusters", as opposed to simply "isolated" ones, as below.



[26] "Clustering validation" suggests that there is an ideal clustering result against which to compare the various algorithmic outcomes, and all that is needed is for analysts to determine how much the outcomes depart from the ideal result. "Cluster quality" is a better way to refer to the process.

There are multiple ways of including both between-cluster and within-cluster similarities in a **cluster quality metric** (CQM): both the **Davies-Bouldin index** and **Dunn's index** do so, to name but two examples. The broad objectives of clustering remain universal: instances within a cluster should be similar; instances in different clusters should be dissimilar.

The problem is that there are many ways for clusters to deviate from this ideal: in specific clustering cases, how do we weigh the "good" aspects (such as high within-cluster similarity) relative to the "bad" ones (such as low between-cluster separation)?

Other internal properties and relationships can also be used and combined in various ways, leading to an embarrassment of riches when it comes to CQMs. The following indices are all available in the `R` package `clusterCrit`, for instance [227]:

- Ball-Hall
- Banfeld-Raftery
- C
- Calinski-Harabasz
- *__Davies-Bouldin__
- Det Ratio and LogDetRatio
- *__Dunn__
- Baker-Hubert Gamma
- GDI
- Gplus
- KsqDetW
- McClain-Rao
- PBM
- Point-Biserial
- Ratkowsky-Lance
- Ray-Turi
- Scott-Symons
- SD
- SDbw
- *__Silhouette__
- Tau
- TraceW and TraceWiB
- Wemmert-Gancarski
- *__WSS__
- LogSSRatio
- Xie-Beni

While it is useful to have access to so many CQMs, we should remain aware that the No-Free Lunch theorem is still in play: none of them is universally superior to any of the others.[27]

In practice, certain approaches (**weightings**) may prove more relevant given the eventual specific analysis goals, but what these could prove to be is not always evident from the context; consequently, we recommend assessing the quality of the clustering outcomes using a variety of CQMs. Studies have been performed to compare a large number of internal validation measures, relative to one another and against human evaluation; generally speaking, variants of the **silhouette index** performed reasonably well (but see previous NFLT comment) [266, 267].

One possible interpretation of these results is that internal validation *via* CQMs may be providing information about clustering across all contexts, and that it is usually easier to identify clustering outcomes that clearly miss the mark than it is to objectively differentiate amongst "good" outcomes, for reasons that are not fully understood yet.

Details on the CQMs are readily available from a multitude of sources (see [4, 266, 267]); we provide more information for 4 CQMs:

- (within) sum of squared error;
- Davies-Bouldin;
- Dunn, and
- silhouette;

[27] Given that all of them are supposedly provide context-free assessments of clustering quality, that is problematic (although emblematic of unsupervised endeavours).

Within Sum of Squares Let $\mathscr{C} = \{C_1, \ldots, C_K\}$ be the K clusters obtained from a dataset $\mathbf{X}$ *via* some algorithm $\mathscr{A}$. Denote the **centroid** (or some other central representative) of cluster C_k by $\mathbf{c}_k$. The **(within) sum of error** for $\mathscr{C}$ is

$$\text{WSE} = \sum_{k=1}^{K} \sum_{\mathbf{x} \in C_k} \text{dissimilarity}(\mathbf{x}, \mathbf{c}_k).$$

The dissimilarity is often selected to be the **square of the Euclidean distance** (thence "sum of squared error"), but the choice of the Euclidean distance (and of the square exponent) is arbitrary – it would make more sense, in practice, to use a dissimilarity related to the similarity measure used by $\mathscr{A}$.

Note that WSE decreases with the number of clusters K, and optimality is obtained at **points of diminishing returns** (see the next section for details); from a validation perspective, it might be easier to interpret the **(within) average error**:

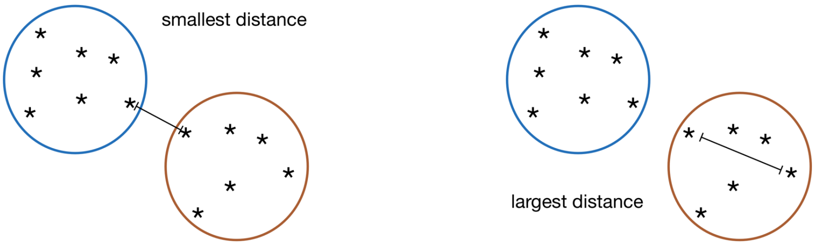
$$\text{WAE} = \text{Avg}_{k=1}^{K} \left\{ \text{Avg}_{\mathbf{x} \in C_k} \{ \text{dissimilarity}(\mathbf{x}, \mathbf{c}_k) \} \right\} = \text{Avg}_{k=1}^{K} \{ s_k \}.$$

Davies-Bouldin Index With s_k, $k = 1, \ldots, K$ as above, the **Davies-Bouldin index** (DBI) is defined as

$$\text{DBI} = \frac{1}{K} \sum_{k=1}^{K} \max_{\ell \neq k} \left\{ \frac{s_k + s_\ell}{\text{dissimilarity}(\mathbf{c}_k, \mathbf{c}_\ell)} \right\}.$$

If the clusters $\{C_k\}$ are **tight** and **dissimilar to one another**, we expect the numerators $s_k + s_\ell$ to be small and the denominators $\text{dissimilarity}(\mathbf{c}_k, \mathbf{c}_\ell)$ to be large, so that the DBI would be **small**.

Dunn's Index With clusters that are **loose** or **somewhat similar to one another**, we expect the DBI to be **large**. There are other ways to assess separation and tightness; **Dunn's index** is the ratio of the **smallest between-cluster dissimilarity** (for pairs of observations not in the same cluster) to the **largest cluster diameter** (within-cluster dissimilarity).



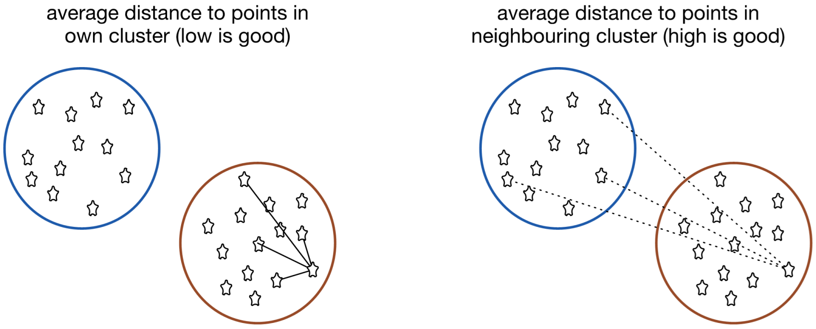
If the clusters $\{C_k\}$ are **tight** and **dissimilar to one another**, we expect the smallest between-cluster dissimilarity to be large and the largest cluster diameter to be small, leasing to a **large** Dunn ratio.

Conversely, with clusters that are **loose** or **somewhat similar to one another**, the Dunn ratio will be **small**. As is the case with the sum of errors and the DBI, the choice of the dissimilarity (or distance) measure leads to different variants of the Dunn index, but all of them take non-negative values.

Silhouette Metric The three previous CQMs provide examples of validation metrics that are computed for the entire clustering outcome; the **silhouette metric** instead assigns a score to each observation, and aggregates the scores at a cluster level and at the dataset level: for a dissimilarity d and a point $\mathbf{x}$ in a cluster C, set

$$b(\mathbf{x}) = \min_{C' \neq C} \left\{ \text{Avg}_{\mathbf{y} \in C'} \{d(\mathbf{x}, \mathbf{y})\} \right\}, \quad a(\mathbf{x}) = \text{Avg}_{\substack{\mathbf{w} \in C \\ \mathbf{w} \neq \mathbf{x}}} \{d(\mathbf{x}, \mathbf{w})\}.$$

Small values of $a(\mathbf{x})$ imply that $\mathbf{x}$ is similar to the instances in its cluster, large values of $b(\mathbf{x})$ imply that it is dissimilar to instances in other clusters.


average distance to points in own cluster (low is good)
average distance to points in neighbouring cluster (high is good)

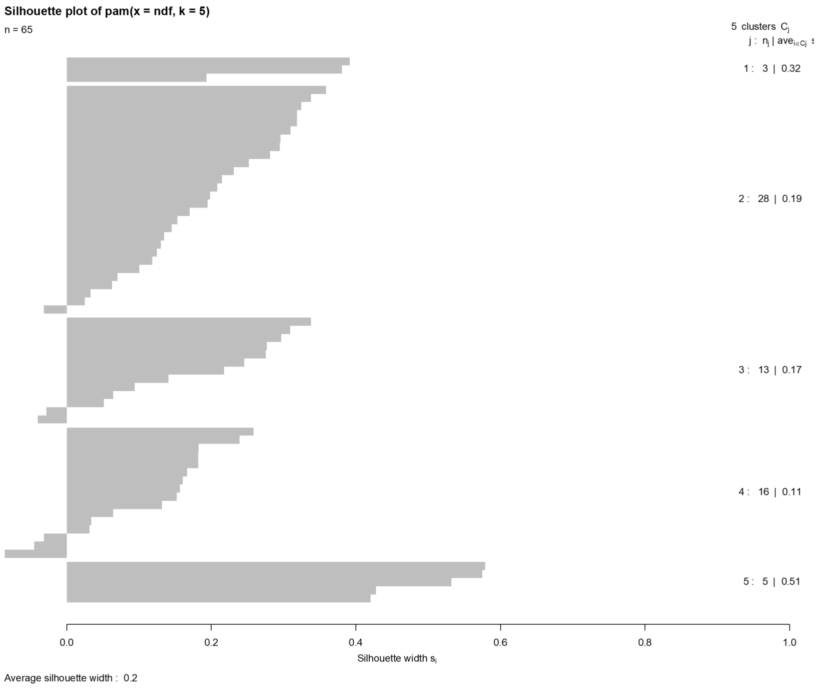
The silhouette metric at $\mathbf{x}$ is

$$\text{silhouette}(\mathbf{x}) = \frac{b(\mathbf{x}) - a(\mathbf{x})}{\max\{a(\mathbf{x}), b(\mathbf{x})\}} = \begin{cases} 1 - a(\mathbf{x})/b(\mathbf{x}) & \text{if } a(\mathbf{x}) < b(\mathbf{x}) \\ 0 & \text{if } a(\mathbf{x}) = b(\mathbf{x}) \\ b(\mathbf{x})/a(\mathbf{x}) - 1 & \text{if } a(\mathbf{x}) > b(\mathbf{x}) \end{cases}$$

Consequently, $-1 \leq \text{silhouette}(\mathbf{x}) \leq 1$ for all $\mathbf{x}$. Thus, when silhouette($\mathbf{x}$) is large (≈ 1), the ratio $a(\mathbf{x})/b(\mathbf{x})$ is small and we interpret $\mathbf{x}$ to be correctly assigned to cluster C (and conversely, when silhouette($\mathbf{x}$) is small (≈ -1), we interpret $\mathbf{x}$'s assignment to C to be "incorrect").

The **silhouette diagram** corresponding to the clustering results displays the silhouette scores for each observation, and averages out the scores in each cluster. The mean over all observations (and the number of instances that have been poorly assigned to a cluster) gives an indication of the appropriateness of the clustering outcome.

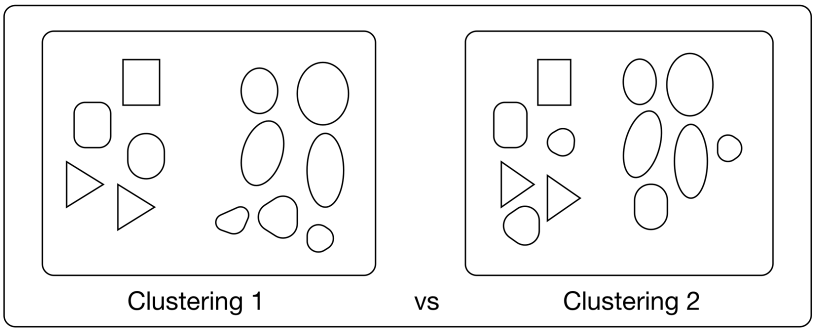
In the example below, 65 observations are separated into 5 clusters: 6 observations are **poorly assigned** (those with negative silhouette scores) and the **average silhouette score** over the entire dataset is 0.2, which suggests that the clustering is more successful than unsuccessful, overall.



Note, however that it could prove difficult to associate intrinsic meaning to a **lone numerical score** – there could be contexts where 0.2 is considered to be a fantastic silhouette score, and others where it is viewed as an abject failure. It is in comparison to the scores obtained by different clustering schemes that this score (and those of the other CQMs) can best serve as an **indicator** of a strong (or a poor) clustering outcome.

Relative Validation Obtaining a **single validation measure** for a **single clustering outcome** is not always that useful – how would we really characterize the silhouette score of 0.2 in the previous example? Could the results be better? Is this the best that can be achieved?

One approach is to compare clustering outcomes across multiple runs to take advantage of non-deterministic algorithms or various parameters' values (see image below, and schematics in Figure 22.10).



If the outcomes of different clustering algorithms on the same dataset are "similar", this provides evidence in favour of the resulting clusters being **efficient**, **natural**, or **valid**, in some sense.

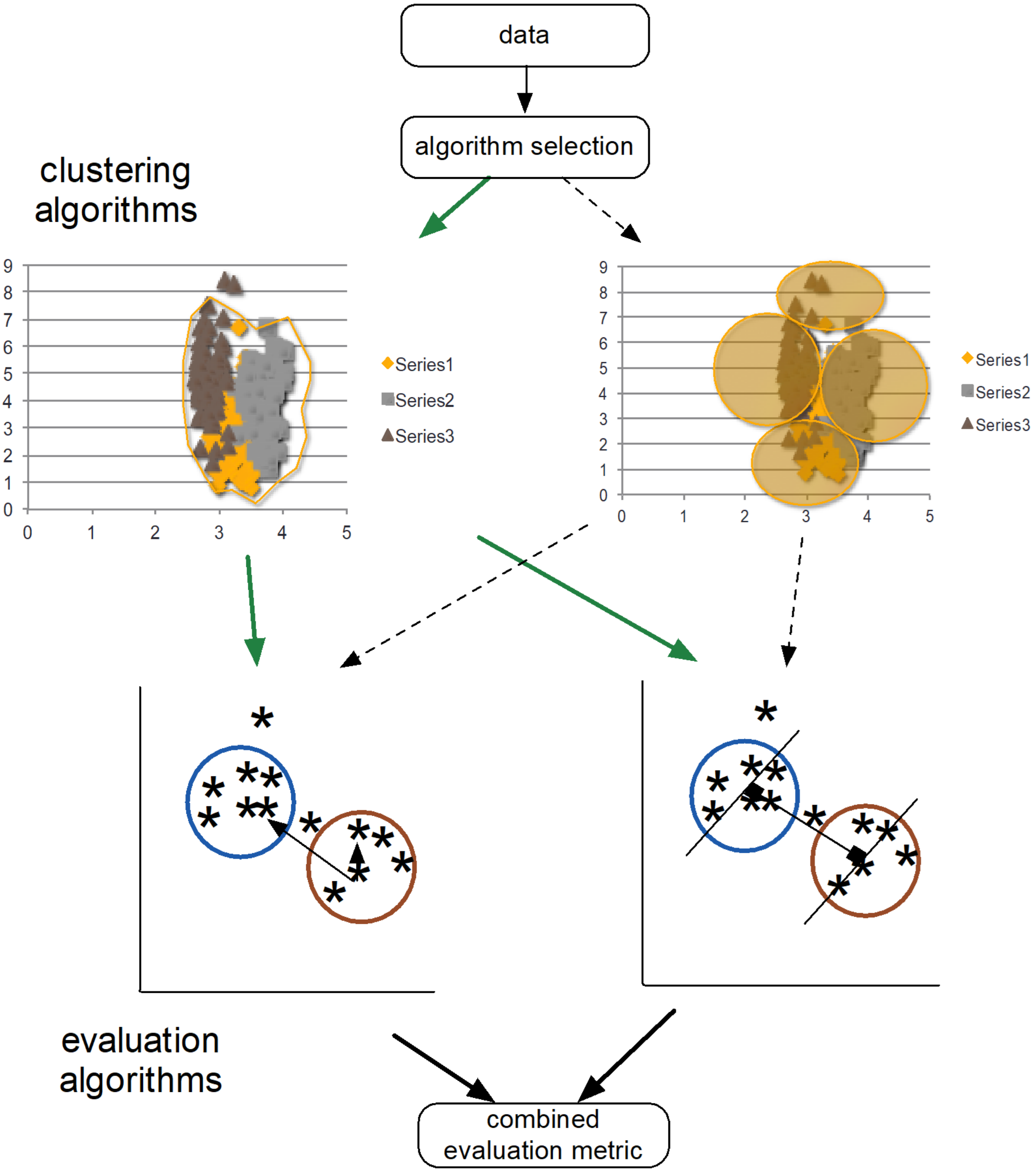
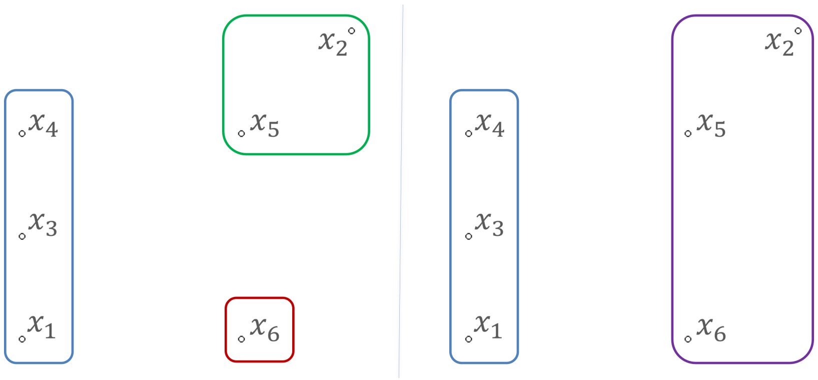


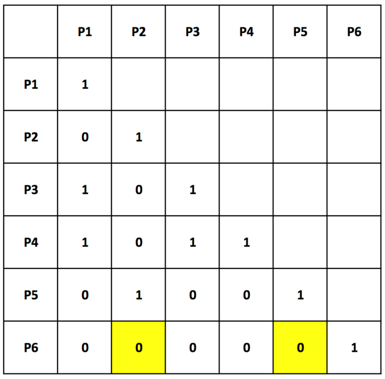
Figure 22.10: Schematics of relative clustering validation.

Consider, for instance, a dataset with 6 instances, which is clustered in two different manners ($\mathscr{A}$ and $\mathscr{B}$, with $|\mathscr{A}| = 3$ and $|\mathscr{B}| = 2$), as shown below. The clusterings are clearly not identical; how similar are they?

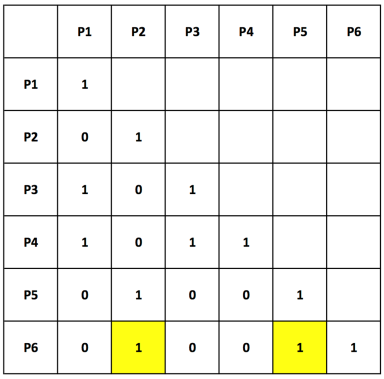


One way to approach relative validation for two outcomes is by computing "**correlations**" between cluster outcomes. Intuitively, we would expect the similarity between both clustering schemes to be high, seeing as the two outcomes are not that different from one another.[28]

This can be represented in the form of matrices:



	P1	P2	P3	P4	P5	P6
P1	1					
P2	0	1				
P3	1	0	1			
P4	1	0	1	1		
P5	0	1	0	0	1	
P6	0	0	0	0	0	1



	P1	P2	P3	P4	P5	P6
P1	1					
P2	0	1				
P3	1	0	1			
P4	1	0	1	1		
P5	0	1	0	0	1	
P6	0	1	0	0	1	1

How can this be quantified? **Correlation measures** include the Rand, Jaccard (see Chapter 27), and Gamma (see [268]) indices.

Let $\mathscr{A} = \{A_1, \ldots, A_k\}$ and $\mathscr{B} = \{B_1, \ldots, B_\ell\}$ be two clusterings of a dataset $\mathbf{X}$. If $\mathbf{X}$ consists of n instances, there are thus

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

pairs of observations in the data. Denote the number of pairs of observations that are in:

- the same cluster in $\mathscr{A}$ **and** the same cluster in $\mathscr{B}$ by ss,
- different clusters in $\mathscr{A}$ **and** different clusters in $\mathscr{B}$ by dd;
- the same cluster in $\mathscr{A}$ **but** different clusters in $\mathscr{B}$ by sd, and
- different clusters in $\mathscr{A}$ **but** the same cluster in $\mathscr{B}$ by ds.

[28] In $\mathscr{B}$, the two smallest clusters of $\mathscr{A}$ have been merged into a single, larger cluster.

Thus,

$$\binom{n}{2} = \text{ss} + \text{dd} + \text{sd} + \text{ds},$$

and we define the **Rand index** of $\mathcal{A}$ and $\mathcal{B}$ as

$$\text{RI}(\mathcal{A}, \mathcal{B}) = \frac{\text{ss} + \text{dd}}{\text{ss} + \text{dd} + \text{sd} + \text{ds}} = \frac{\text{ss} + \text{dd}}{\binom{n}{2}}.$$

Large values of the index are indicative of similarity of clustering outcomes.[29] Unfortunately, the Rand index does not satisfy the **constant baseline property**.[30]

The **adjusted Rand index** (as well as several other pair-counting, set-matching, and information theoretic measures) relies on the **contingency table** between $\mathcal{A}$ and $\mathcal{B}$, a method to summarize the outcomes of two clustering results on the same dataset:

	B_1	$\cdots$	B_ℓ	Total
A_1	$n_{1,1}$	$\cdots$	$n_{1,\ell}$	μ_1
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
A_k	$n_{k,1}$	$\cdots$	$n_{k,\ell}$	μ_k
Total	ν_1	$\cdots$	ν_ℓ	n

In this array, $n_{i,j} = |A_i \cap B_j|$ represents the number of instances that are found in **both** the cluster $A_i \in \mathcal{A}$ and $B_j \in \mathcal{B}$. The adjusted Rand index ($\in [-1, 1]$) is given by

$$\text{ARI}(\mathcal{A}, \mathcal{B}) = \frac{\sum_{ij} \binom{n_{ij}}{2} - \left[\sum_i \binom{\mu_i}{2} \sum_j \binom{\nu_j}{2}\right] \Big/ \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{\mu_i}{2} + \sum_j \binom{\nu_j}{2}\right] - \left[\sum_i \binom{\mu_i}{2} \sum_j \binom{\nu_j}{2}\right] \Big/ \binom{n}{2}},$$

which can also be re-written as

$$\text{ARI}(\mathcal{A}, \mathcal{B}) = \frac{2(\text{ss} \cdot \text{dd} - \text{sd} \cdot \text{ds})}{(\text{ss} + \text{sd})(\text{ds} + \text{dd}) + (\text{ss} + \text{ds})(\text{sd} + \text{dd})}.$$

It is straightforward to compute RI and ARI for the two clusterings of the artificial example with 6 instances. Since $n = 6$, there are $6 \cdot 5/2 = 15$ pairs of observations in the data, and we have:

- ss $= 4$ (x_1 and x_3; x_1 and x_4; x_3 and x_4; x_2 and x_5);
- ds $= 2$ (x_2 and x_6; x_5 and x_6);
- sd $= 0$ (none);
- dd $= 9$ (all remaining pairs).

Thus,

$$\text{RI}(\mathcal{A}, \mathcal{B}) = \frac{4 + 9}{4 + 9 + 0 + 2} = \frac{13}{15} = 0.87$$

$$\text{ARI}(\mathcal{A}, \mathcal{B}) = \frac{2(4 \cdot 9 - 0 \cdot 2)}{(4 + 0)(2 + 9) + (4 + 2)(0 + 9)} = 0.73.$$

Both of these values are indicative of high similarity between the clustering outcomes $\mathcal{A}$ and $\mathcal{B}$.

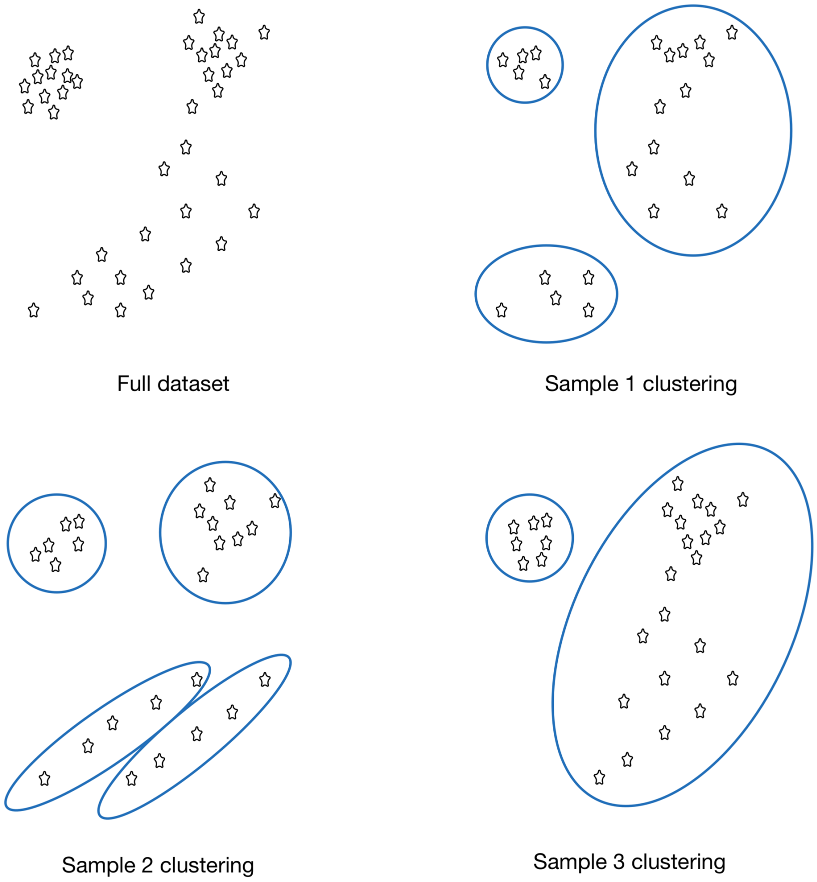
[29] The formula for $\text{RI}(\mathcal{A}, \mathcal{B})$ reminds one of the definition of accuracy, a performance evaluation measure for (binary) classifiers.

[30] In a nutshell, the expected value of $\text{RI}(\mathcal{A}, \mathcal{B})$ for independent, random clusterings $\mathcal{A}$ and $\mathcal{B}$ is not 0 [269].

Cluster **stability** can also be used to assess the appropriateness of the choice of algorithm for the data. Assume that we apply a clustering algorithm $\mathscr{G}$ to a dataset $\mathbf{X}$, resulting in a clustering scheme $\mathscr{C} = \{C_1, \ldots, C_K\}$.

In general, a dataset $\mathbf{X}$ is a **realization** of a (potentially unknown) underlying data generating mechanism related to the situation of interest; a different realization $\mathbf{X}'$, which could arise from the collection of new data, may yield a distinct clustering scheme $\mathscr{C}'$. How **stable** is the clustering outcome of $\mathscr{G}$, relative to the realization $\mathbf{X}$?

We can get a sense for the underlying stability by sampling multiple row subsets from $\mathbf{X}$;[31] however this is achieved, we have obtained ℓ subsets $\mathbf{X}_1, \ldots, \mathbf{X}_\ell \subseteq \mathbf{X}$, on which we apply the algorithm $\mathscr{G}$, with parameters $\mathscr{P}$, yielding the corresponding clustering schemes $\mathscr{C}_1, \ldots, \mathscr{C}_\ell$.



Full dataset

Sample 1 clustering

Sample 2 clustering

Sample 3 clustering

Let $\mathscr{W}$ be the similarity matrix for the various clustering schemes:

$$\mathscr{W} = \begin{pmatrix} w(\mathscr{C}_1, \mathscr{C}_1) & \cdots & (\mathscr{C}_1, \mathscr{C}_\ell) \\ \vdots & \ddots & \vdots \\ w(\mathscr{C}_\ell, \mathscr{C}_1) & \cdots & (\mathscr{C}_\ell, \mathscr{C}_\ell) \end{pmatrix};$$

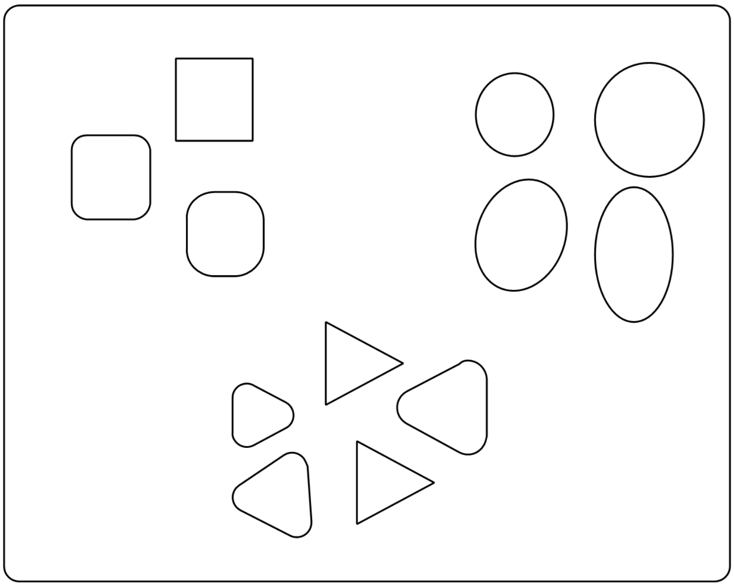
this $\mathscr{W}$ can be used as the basis of a **meta-clustering scheme** in which $\mathscr{C}_1, \ldots, \mathscr{C}_\ell$ play the role of instances, using a graph-based meta-clustering method such as DBSCAN (which we will discuss in Section 22.4.1). If the clustering results obtained from $\mathbf{X}$ by applying $\mathscr{G}$ are **stable**, we might expect the meta-clustering results to yield a **single meta-cluster**.[32]

[31] Alternatively, we could also sample from $\mathbf{X}$'s columns, or sample columns from a variety of sampled rows of $\mathbf{X}$.

[32] If multiple meta-clusters are found from $\mathscr{W}$, this supports the position that $\mathscr{G}$ does not produce stable clusters in $\mathbf{X}$, although this does not necessarily imply **instability** as the number of meta-clusters could be an artefact related to the choice of the meta-clustering method. This approach seems simple in theory, but in practice it often simply pushes the issues and challenges of clustering to the meta-clustering activity; a more sophisticated treatment of the problem of cluster stability assessment is presented in [270].

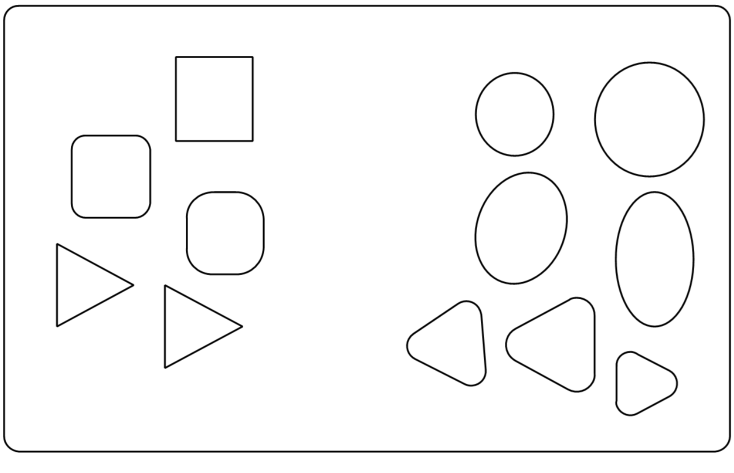
External Validation In everyday applications, we tend to gauge clustering results against some (often implicit) **external expectation**: we cannot help but bring in outside information to evaluate the clusters.

The outside information is typically what is deemed to be the 'correct' groups to which the instances belong.[33] In the example below, for instance, we might be interested in finding **natural groups** in a dataset of objects, but we might hold the pre-conceived notion that the natural groups are linked to the underlying **shape** (square, triangle, circle).



Natural Groupings

If the outcome agrees with this (external) notion, we naturally feel that the clustering was successful; if, the outcome seems to pick up something about the sharpness of the shapes' vertices, say (as in the image below), we might conclude that the algorithm does not do a good job of capturing the essential nature of the objects, or, more rarely, that we need to revisit our pre-conceived notions about the dataset and its natural groups.



Clustering Results

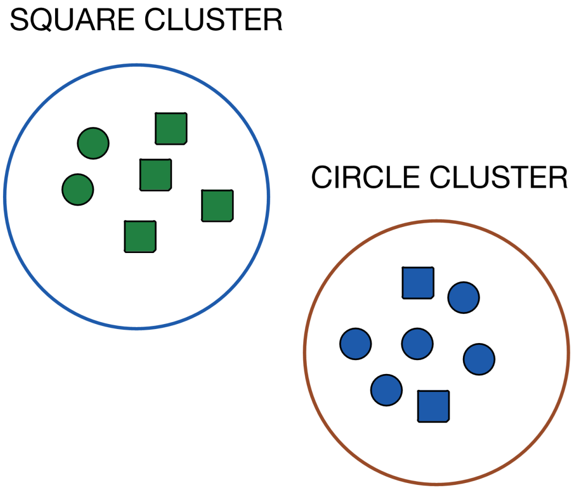
[33] This is starting to look a lot like **classification**, a supervised learning approach.

This validation strategy is often used to **build confidence** in the overall approach, based on preliminary or sample results, but it rests on a huge assumption (which often conflicts with the unsupervised learning framework), namely, that the natural groups in the data are **identifiable**, explicitly or implicitly.

Due to the conceptual similarity to classification,[34] **external validation measures** often resemble classification performance measures. Case in point, the **purity metric**. In the presence of an external classification variable, this metric assign a label to a cluster C according to the most frequent classes of the instances in C, and

$$\text{purity}(C) = \frac{\#\,\text{correctly assigned points in } C}{|C|},$$

as in the example below:



The **clustering purity score** for $\mathscr{C} = \{C_1, \ldots, C_K\}$ is obtained as the average of the cluster purity scores, or as a weighted average of purity scores:

$$\text{weighted purity}(\mathscr{C}) = \frac{1}{n}\sum_{\ell=1}^{K} |C_\ell| \cdot \text{purity}(C_\ell),$$

where n represents the number of instances in the data.

In the image above, the green cluster is labeled as the square cluster (since 4 of its 6 instances are classified as squares), and the blue cluster is labeled as the circle cluster (since 5 of its 7 instances are classified as circles). At the cluster level, the purity scores are thus:

$$\text{purity}(C_\square) = \frac{2}{3}, \quad \text{purity}(C_\bigcirc) = \frac{5}{7};$$

the average and weighted purity scores are

$$\text{average purity}(\mathscr{C}) = \frac{1}{2}\left(\frac{2}{3} + \frac{5}{7}\right) = 69.0\%$$

$$\text{weighted purity}(\mathscr{C}) = \frac{1}{6+7}\left(6 \cdot \frac{2}{3} + 7 \cdot \frac{5}{7}\right) = 69.2\%.$$

[34] Which it is emphatically not, it bears repeating.

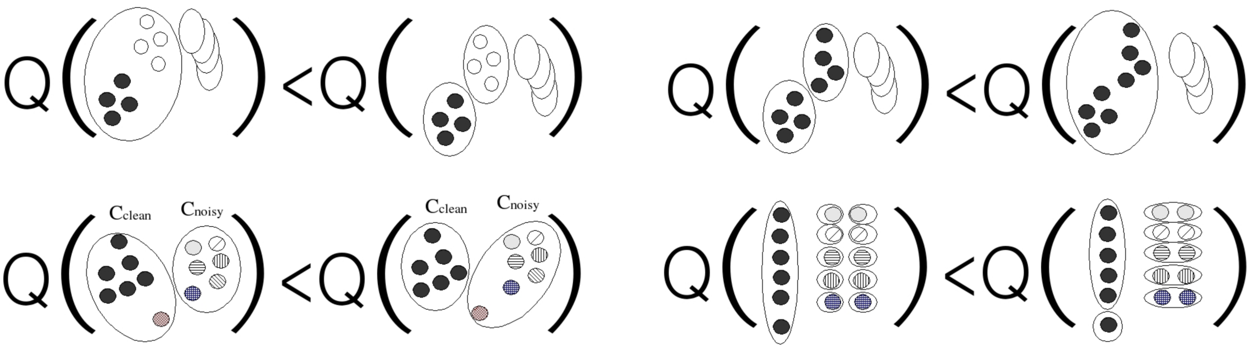


Figure 22.11: Useful external quality metric considerations: homogeneity (top left), completeness (top right), noisy and outlying data (bottom left), size and quantity (bottom right).

These two numbers are very nearly identical because the clusters have roughly the same size; if the size variance is large, the two measurements would be quite different. The purity is an obvious analogue to **accuracy**; other measures based on **precision** and **recall** are also popular [271].

Useful external quality metrics take advantage of the natural classes (if they are aligned with the clustering results), and take into account cluster **homogeneity** (top left, Figure 22.11), **completeness**, (top right), **noisy and outlying data** (bottom left), and **size vs. quantity** considerations (bottom right): the preferred behaviour is shown on the right [271].

Example Let us illustrate some of these notions using various $k-$means and hierarchical clusters of the Gapminder data used in the previous sections. In all instances, we use Euclidean distance on the scaled dataset.

Internal Validation We use the `R` packages `cluster`, `fpc`, and `clusterCrit` to compute 3 CQMs: the Dunn index, the average silhouette width, and the **Calinski-Harabasz** index, which is simply the ratio of the sum of **between-clusters dispersion** to the **inter-cluster dispersion** for all clusters (higher is better).

We start by clustering the data using $4-$means; we then use hierarchical clustering with complete linkage and 3 clusters (the global structure has already been computed).

```
set.seed(123) # for replicability
kmeans.4 = kmeans(gapminder.SoCL.2011.s,4,iter.max=2509,nstart=1)
stats.kmeans.4 <- as.numeric(
    clusterCrit::intCriteria(as.matrix(gapminder.SoCL.2011.s),
    kmeans.4$cluster,
    c("Dunn","Silhouette","Calinski_Harabasz")))

dist.all <- cluster::daisy(gapminder.SoCL.2011.s,metric="euclidean",stand=FALSE)
hc.1.3 <- cutree(hclust.gapminder.SoCL.2011, k = 3)
stats.hc.1.3 <- c(fpc::cluster.stats(dist.all, clustering=hc.1.3, silhouette = TRUE)$dunn,
    fpc::cluster.stats(dist.all, clustering=hc.1.3, silhouette = TRUE)$avg.silwidth,
    fpc::cluster.stats(dist.all, clustering=hc.1.3, silhouette = TRUE)$ch)
```

The results are summarized below:

```
stats <-rbind(stats.kmeans.4,stats.hc.1.3)
colnames(stats) <- c("Dunn",
  "Silhouette",
  "Calinski-Harabasz")
stats
```

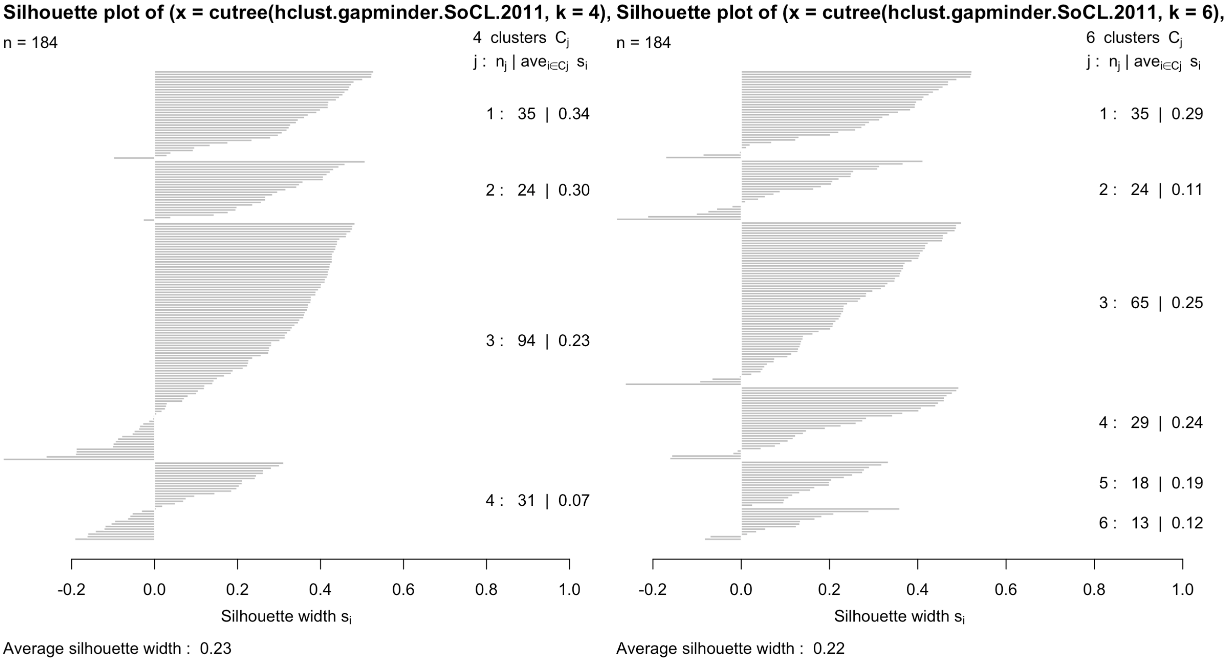
method	Dunn	Avg. Sil.	C.-H.
4−means	0.097	0.315	139.0
HC(comp; 3)	0.091	0.274	125.4

Both of the Dunn values are low, although the 4−means result is marginally superior; while the average silhouette widths are also low, they are least positive in both cases, with a slight advantage in favour of 4−means; the Calinski-Harabasz values are not very indicative on their own, but once again, 4−means comes out ahead of HC.

The average silhouette width is an intriguing metric. On the one hand, we attempt to gauge the quality of the **entire clustering** with a single number, but the average is a fickle summary measurement and potentially affected by outlying values; on the other, we do have access to a silhouette score **for each observation**, and can thus get a better idea of the performance by studying the **silhouette profile**.

We show the silhouette results for hierarchical clustering with complete linkage for 4 (average width= 0.23) and 6 clusters (average width= 0.22).

```
plot(cluster::silhouette(cutree(hclust.gapminder.SoCL.2011, k = 4), dist.all))
plot(cluster::silhouette(cutree(hclust.gapminder.SoCL.2011, k = 6), dist.all))
```



The average silhouette width seems to favour the 4-cluster result, the profile for the 6-cluster result seems more in-line with desirable properties: in both instances, some observations are "mis-clustered" (negative silhouette scores), but these seem to be more broadly distributed in the latter case.[35]

Relative Validation We compute the Rand index (RI) and the adjusted Rand index (ARI) to compare the outcomes of a single run of 2−means ($\mathscr{A}_2$), 3−means ($\mathscr{A}_3$), and 4−means ($\mathscr{A}_4$), respectively.

```
set.seed(1) # for replicability
kmeans.2 = kmeans(gapminder.SoCL.2011.s,2,iter.max=250,nstart=1)
kmeans.3 = kmeans(gapminder.SoCL.2011.s,3,iter.max=250,nstart=1)
kmeans.4 = kmeans(gapminder.SoCL.2011.s,4,iter.max=250,nstart=1)
```

We can compute the Rand index and the adjusted Rand index using the following function:

```
# create a matrix of 1s and 0s depending as to whether
# observations i and j are in the same cluster or not
w2=kmeans.2$cluster
mat2=floor(1 - abs(sqrt(w2%*%t(w2))) %% 1)

w3=kmeans.3$cluster
mat3=floor(1 - abs(sqrt(w3%*%t(w3))) %% 1)

w4=kmeans.4$cluster
mat4=floor(1 - abs(sqrt(w4%*%t(w4))) %% 1)

# build the rand index from these matrices
randindices  <- function(W1,W2) {
  diag(W1) <- -1
  diag(W2) <- -1
  W=table(W1+2*W2)
  W=W[-c(1)]
  dd=W[1]
  sd=W[2]
  ds=W[3]
  ss=W[4]
  RI=(ss+dd)/(ss+dd+sd+ds)
  ARI=2*(ss*dd-sd*ds)/((ss+sd)*(ds+dd)+(ss+ds)*(sd+dd))
  randindices=data.frame(cbind(RI[[1]],ARI[[1]]))
}

# compute RI and ARI for the 3 comparisons
(randindices(mat2,mat3))
(randindices(mat2,mat4))
(randindices(mat3,mat4))
```

```
    RI    ARI        RI    ARI        RI    ARI
0.7327 0.5152   0.6407 0.3285   0.7549 0.4584
```

[35] in the 4-cluster case, half a cluster seems to have been mis-assigned, for instance.

There are $\binom{184}{2} = 16836$ pairs of distinct observations in the 2011 Gapminder dataset; the full pair types and indices break down as below:

Schemes	ss	dd	sd	ds	RI	ARI
$\mathscr{A}_2, \mathscr{A}_3$	5304	7032	3852	648	0.73	0.52
$\mathscr{A}_2, \mathscr{A}_4$	4395	6392	4761	1288	0.64	0.33
$\mathscr{A}_3, \mathscr{A}_4$	3754	8955	2198	1929	0.75	0.46

$\mathscr{A}_2, \mathscr{A}_3$ are relatively similar according to RI, as are $\mathscr{A}_3, \mathscr{A}_4$, but the ARI suggests that $\mathscr{A}_2, \mathscr{A}_3$ are more similar to one another than $\mathscr{A}_3, \mathscr{A}_4$ are; $\mathscr{A}_2, \mathscr{A}_4$ are not as similar, according to both indices, which is not that surprising as the number of clusters in this case jumps from 2 to 4.

External Validation Finally, we compare the clustering results of hierarchical clustering, for 4 and 8 clusters, with a variety of external grouping: 6 world regions, as determined by the Gapminder Foundation, and OECD/G77 membership (see Figure 22.12).

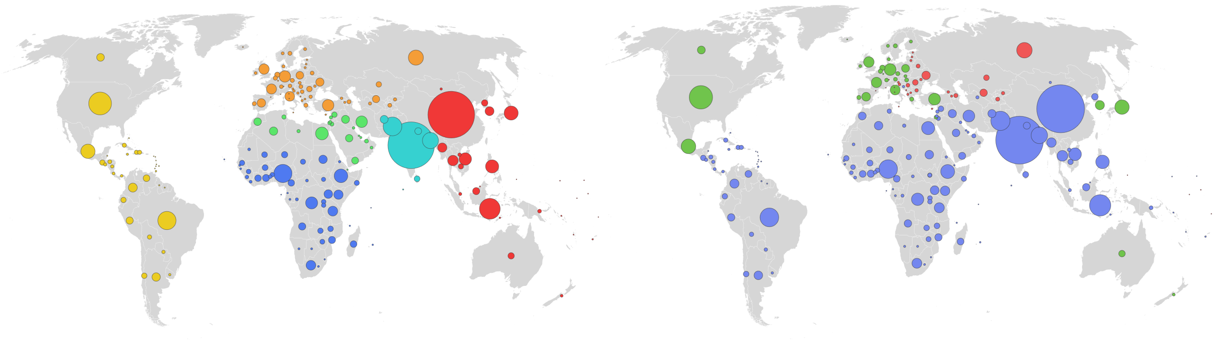


Figure 22.12: 6 world regions (left): America (yellow, 33 countries), East Asia Pacific (red, 26), Europe Central Asia (orange, 49), Middle East North Africa (green, 20), South Asia (turquoise, 8), Sub Saharan Africa (blue, 48); memberships (right): OECD (green, 30), G77 (purple, 128), other (red, 26); bubble size represents population [61].

We start by importing the external groupings.

```
gapminder.regions = read.csv("gapminder_regions.csv",
    stringsAsFactors=TRUE)
colnames(gapminder.regions)[1] <- c("geo")
```

Then, we cluster the data using HC with complete linkage, for $k = 4$ and $k = 8$ clusters (using Euclidean dissimilarity). Recall that the dendrogram structure was originally stored in `hclust.gapminder.SoCL.2011`.

```
hc.4.ce <- as.factor(cutree(hclust.gapminder.SoCL.2011, k = 4)) # complete, Euclidean
hc.8.ce <- as.factor(cutree(hclust.gapminder.SoCL.2011, k = 8)) # complete, Euclidean

geo = names(hc.4.ce)
clusters=data.frame(geo,hc.4.ce,hc.8.ce)
external.results <- merge(gapminder.regions,clusters, by="geo")
```

The first 3 entries of the `external.results` are shown on the next page.

```
head(external.results,3)
```

```
geo name         four_regions  eight_regions       six_regions          members_oecd_g77  hc.4.ce  hc.8.ce
afg Afghanistan  asia          asia_west           south_asia           g77               1        1
ago Angola       africa        africa_sub_saharan  sub_saharan_africa   g77               1        1
alb Albania      europe        europe_east         europe_central_asia  others            3        3
```

The clusters are then labeled with their most frequent cluster assignment, which can be extracted with the following function:

```
mode <- function(x) { names(which.max(table(x))) }

tab <- external.results |>  group_by(hc.4.ce) |>
  summarise(mode = mode(six_regions), n=n())
n.mode <- external.results |>  group_by(hc.4.ce) |>
  count(six_regions) |>
  summarise(n.mode = max(n))
info <- merge(tab,n.mode, by="hc.4.ce")[,2:4]
```

Are there any reasons to suspect that the clusters would be aligned with these external classes? For the 6 world regions classes, the clusters labels (the most frequent class per cluster) for HC(4) are shown below:

Cluster	Label	Size	Frequency
1	Sub Saharan Africa	35	31
2	East Asia Pacific	24	9
3	Europe Central Asia	94	42
4	Sub Saharan Africa	31	14

```
info[,4] <- info$n.mode/info$n
(purity <- mean(info$V4))
(weighted.purity <- weighted.mean(info$V4,info$n))
```

This clustering scheme yields a purity of 0.54 and a weighted purity of 0.52 – the overall score is not great, but the Sub Saharan countries are fairly well captured by clusters 1 and 4.

We repeat the same process for HC(8) (the code is omitted). The clusters labels in that case are found below:

Cluster	Label	Size	Frequency
1	Sub Saharan Africa	35	31
2	America	17	6
3	Europe Central Asia	65	34
4	East Asia Pacific	7	3
5	america	29	10
6	Sub Saharan Africa	18	7
7	East Asia Pacific	10	5
8	Sub Saharan Africa	3	3

This now yields a purity of 0.55 and a weighted purity of 0.54; which is still .... not that great. Perhaps the clusters have little to do with geography, in the end.

Are they aligned with OECD/G77/other membership? The labels for HC(8) with this external grouping are found below:

Cluster	**Label**	**Size**	**Frequency**
1	G77	27	27
2	G77	29	22
3	OECD	28	17
4	G77	20	18
5	G77	12	11
6	OECD	23	11
7	G77	25	24
8	G77	20	10

The purity values are 0.77 and 0.76, respectively: these are better values than the previous external validation attempts, but they might not really be meaningful given the preponderance of G77 countries in the data.

It seems, then, that neither of the external classifications is a good gauge of cluster validity for this data.

For the most part, the cluster validation yields middling results. The few algorithms we have tried with the data suggest that there is some low-level grouping at play, but nothing we have seen so far would suggest that the data segments are all that "natural."

While this result is somewhat disappointing, we should note that this is often the case with real-world data: there is no guarantee that natural groups even exist in the data. However, we have not been directing our choices of algorithms and parameters – up to now, they have been made fairly arbitrary. Can anything be done to help with **model selection**?

22.3.2 Model Selection

How do we pick the number of clusters and the various other parameters (including choice of algorithm) to use for "optimal" results? A common approach is to look at all the outcomes obtained from various parameter runs and replicates (for a given algorithm), and to select the parameter values that optimize a set of QCMs, such as Davies-Bouldin, Dunn, CH, etc.

Optimization is, of course, dependent on each QCM's properties: in some cases, we are searching for parameters that maximizes the index, in other cases, those that minimize it, and yet in other cases, for "knees" or "change points" in various associated plots.

Note, however, that the parameter values that optimize a QCM may not optimize others; when they coincide, this reinforces the support for the existence of natural groups; when they do not, they provide a smaller collection of models from which to select, removing some of the arbitrariness discussed above.

This can also be done for clustering outcomes arising from different algorithms, although in this case we are not selecting parameter values so much as identifying the model that best describes the natural groups in the data among all results, according to some metric(s).

The metrics presented in Section 22.3.1 all provide frameworks to achieve this. There are additional approaches, such as: seeking the clustering $\mathscr{C} = \{C_1, \ldots, C_k\}$, among a list of such outcomes, which minimizes the **quadratic cost**

$$\Lambda_{\mathbf{W}}(\mathscr{C}) = -\text{trace}\left(Z^{\top}(\mathscr{C})\mathbf{W}Z(\mathscr{C})\right),$$

where $z_{i,\ell} = 1$ is $\mathbf{x}_i \in C_\ell$ and 0 otherwise, associated with a **similarity matrix W**; or methods relying on stability assessment [270, 272]. Model assessment and selection remains a popular research topic.

But it remains important to remember that there is a lot of diversity in clustering validation techniques. The various types of validation methods do not always give concordant results; this variation within the types can be demoralizing at times, but it can also be leveraged to extract useful information about the data's **underlying structure**.

In general, we should avoid using a single assessment method; it is preferable to seek "signals of agreement" across a **variety of strategies** (both in the choices of clustering algorithms and evaluation methods). Finally, remember that clustering results may just be 'ok' ... but that is ok too! We can study the situation and decide what is important and what can safely be ignored – as always, **a lot depends on the context**.

Example How many clusters k should we seek when clustering the (scaled) 2011 Gapminder dataset using Euclidean distance? For each $k = 2, \ldots, 15$, we compute the outcome of $m = 40$ runs of k−means, and average the **within sum of squares** (WSS) and a (modified) **Davies-Bouldin** index (DBI) over the runs. The optimal number of parameters is obtained at a DBI maximum or a WSS "knee".

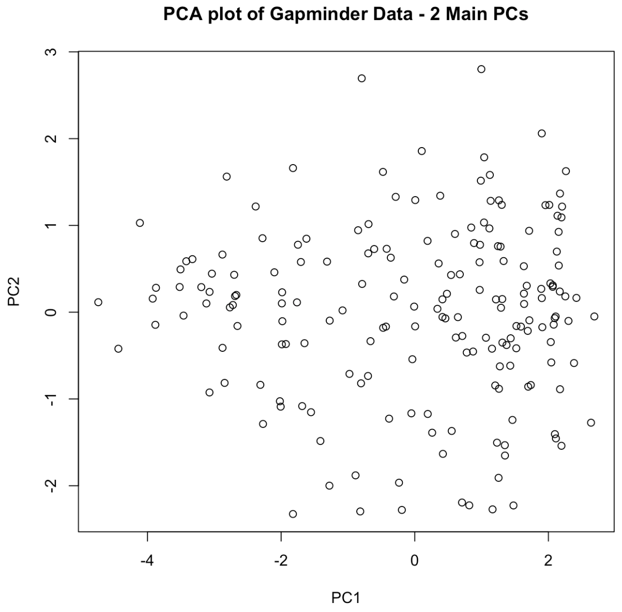
We start by computing the principal components for displaying purposes – although we could also use them to cluster the data, at the cost of some information about the dataset.

Principal component distribution

```
decomposition
pc.agg.data = princomp(gapminder.SoCL.2011.s)
summary(pc.agg.data)
```

```
Importance of components:
                          Comp.1    Comp.2     Comp.3     Comp.4    Comp.5
Standard deviation     1.850624 0.9973737 0.49950729 0.45130387 0.3163521
Proportion of Variance 0.688705 0.2000380 0.05017419 0.04095763 0.0201251
Cumulative Proportion  0.688705 0.8887431 0.93891726 0.97987490 1.0000000
```

```
pc.df.agg.data = cbind(pc.agg.data$scores[,1],
    pc.agg.data$scores[,2])
plot(pc.df.agg.data, xlab="PC1", ylab="PC2")
title('PCA plot of Gapminder Data - 2 Main PCs')
```

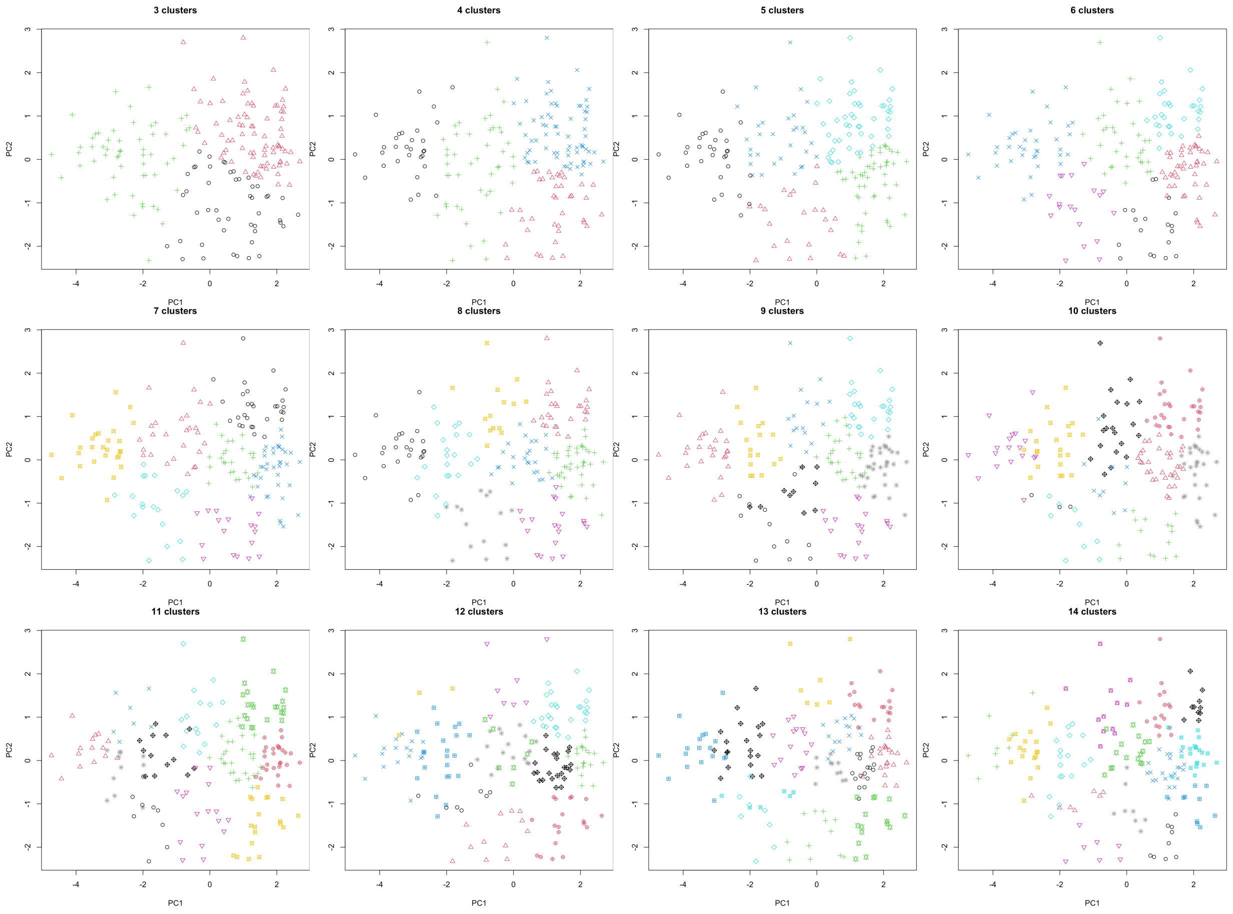


The Davies-Bouldin index is computed using the following formula.

```
Davies.Bouldin <- function(A, SS, m) {
  # A  - the centres of the clusters
  # SS - the within sum of squares
  # m  - the sizes of the clusters
  N <- nrow(A)   # number of clusters
  # intercluster distance
  S <- sqrt(SS/m)
  # Get the distances between centres
  M <- as.matrix(dist(A))
  # Get the ratio of intercluster/centre.dist
  R <- matrix(0, N, N)
  for (i in 1:(N-1)) {
    for (j in (i+1):N) {
      R[i,j] <- (S[i] + S[j])/M[i,j]
      R[j,i] <- R[i,j]
    }
  }
  return(mean(apply(R, 1, max)))
}
```

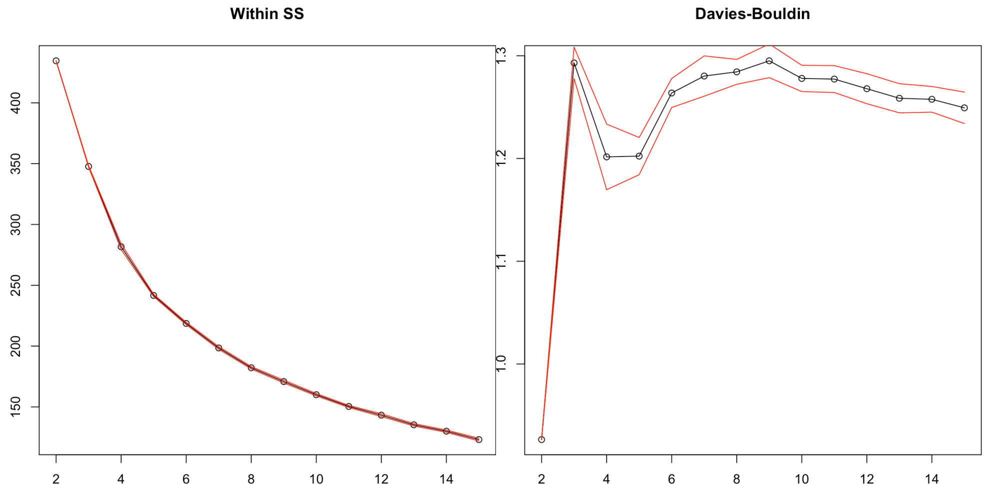
For each $k = 2, \dots, 15$, we run $k-$means $N = 40$ times (using Euclidean dissimilarity). One realization is displayed for each k, as are the DBI curves and the WSS curves (with confidence bands). The clusters are displayed on the first 2 principal components of the dataset, which explain 88% of the variation in the data.

```
N = 40            # Number of repetitions
max.cluster = 15  # Number of maximum number of desired clusters

# initializing values
m.errs = m.DBI = s.errs = s.DBI <- rep(0, max.cluster)

# clustering and plotting
set.seed(1)
for (i in 2:max.cluster) {
  errs = DBI <- rep(0, max.cluster)

  for (j in 1:N) {
    # data, number of internal shifts of the cluster centres, number of clusters
    KM <- kmeans(gapminder.SoCL.2011.s,i,iter.max=2509,nstart=1)
    errs[j] <- sum(KM$withinss)
    DBI[j] <- Davies.Bouldin(KM$centers, KM$withinss, KM$size)
  }

  m.errs[i - 1] = mean(errs)
  s.errs[i - 1] = sd(errs)
  m.DBI[i - 1] = mean(DBI)
  s.DBI[i - 1] = sd(DBI)

  plot(pc.df.agg.data,col=KM$cluster, pch=KM$cluster, main=paste(i,"clusters"),
       xlab="PC1", ylab="PC2")
}
```



The average within sum of squares curve and the average Davies-Bouldin curves are also provided, with 95% confidence intervals.

```
# WSS
MSE.errs_up = m.errs + 1.96 * s.errs / sqrt(N)
MSE.errs_low = m.errs - 1.96 * s.errs / sqrt(N)

plot(2:(max.cluster), m.errs[1:(length(m.errs)-1)], main = "Within SS", xlab="", ylab="")
lines(2:(max.cluster), m.errs[1:(length(m.errs)-1)])
par(col = "red")

lines(2:(max.cluster), MSE.errs_up[1:(length(MSE.errs_up)-1)])
lines(2:(max.cluster), MSE.errs_low[1:(length(MSE.errs_low)-1)])

# DBI
MSE.DBI_up = m.DBI + 1.96 * s.DBI / sqrt(N)
MSE.DBI_low = m.DBI - 1.96 * s.DBI / sqrt(N)

par(col = "black")
plot(2:(max.cluster), m.DBI[1:(length(m.DBI)-1)], main = "Davies-Bouldin", xlab="", ylab="")
lines(2:(max.cluster), m.DBI[1:(length(m.DBI)-1)])
par(col="red")

lines(2:(max.cluster), MSE.DBI_up[1:(length(MSE.DBI_up)-1)])
lines(2:(max.cluster), MSE.DBI_low[1:(length(MSE.DBI_low)-1)])
```



Where is the Davies-Bouldin index maximized?

```
(i_choice <- which(m.DBI==max(m.DBI[1:(length(m.DBI)-1)]))+1)
```

```
[1] 9
```

The WSS curve does not yield much information, but the DBI curve suggests that both $k = 3$ and $k = 9$ could be good parameter choices. With parsimony considerations in mind, we might elect to use $k = 3$, but if the results are too simple or if signs of instability appear,[36] $k = 9$ might prove to be a better choice in the end.

[36] Recall that k−means is a stochastic algorithm.

22.4 Advanced Clustering Methods

In the rest of this chapter, we present representative clustering algorithms from the remaining families.[37]

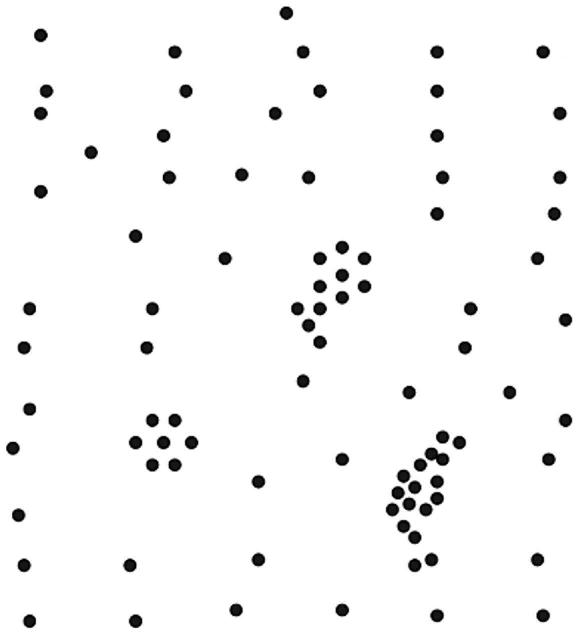
22.4.1 Density-Based Clustering

The assumptions of the $k-$means algorithm imply that the clusters that it finds are usually **Gaussian**.[38] But this is not always a desired outcome.

In **density-based clustering**, it is the **density** of observations and the **connectivity** of the accompanying clustering network that determine the number and location of clusters.[39] Popular density-based clustering algorithms include DBSCAN, DENCLUE, OPTICS, CHAMELEON, etc.

Once density has been defined in a meaningful way, [40] density-based algorithms are straightforward to apply (see [4, 161, 219, 225, 226]).

Density How do we measure density? Intuitively, we can recognize areas of **low density** and **high density** in the (artificial) dataset below.



As the saying goes, “birds of a feather flock together”; it should not come as a surprise that areas of higher density could be viewed as **clusters** in the data. In that context, if $\mathbf{\Psi} \subseteq \mathbb{R}^n$ is an $n-$dimensional **sub-manifold** of $\mathbb{R}^n$, we could define the **density** of $\mathbf{\Psi}$ around $\mathbf{x}$ by, say,

$$\text{density}_{\mathbf{\Psi}}(\mathbf{x}; d) = \lim_{\varepsilon \to 0^+} \frac{\text{Vol}_n(B_d(\mathbf{x}, \varepsilon) \cap \mathbf{\Psi})}{\text{Vol}_n(B_d(\mathbf{x}, \varepsilon))},$$

where

$$B_d(\mathbf{x}, \varepsilon) = \{\mathbf{y} \in \mathbb{R}^n \mid d(\mathbf{x}, \mathbf{y}) < \varepsilon\}$$

and

$$\text{Vol}_n(A) = n - \text{volume of } A \text{ in } \mathbf{R}^n.$$

[37] Substantially more information on the topic can be found in [2, 4, 175].

[38] That is, blob-like.

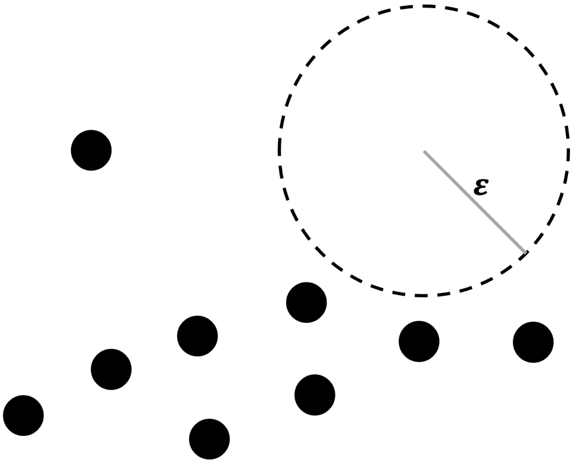
[39] We will discuss these further in the next section.

[40] Which depends on a number of contextual factors.

DBSCAN In practice, the dataset $\mathbf{X}$ is usually a **discrete subset** of $\mathbb{R}^n$, and the limit definition above cannot apply. **Density-based spatial clustering of applications with noise** (DBSCAN) estimates the density at an observations $\mathbf{x} \in \mathbf{X}$ as follows: we pick a "reasonable" value of $\varepsilon^* > 0$ and set

$$\text{density}_{\mathbf{X}}(\mathbf{x}; d) = |B_d(\mathbf{x}, \varepsilon^*) \cap \mathbf{X}|\,.$$

The outcome depends, of course, on the choice of ε^* and the distance d.



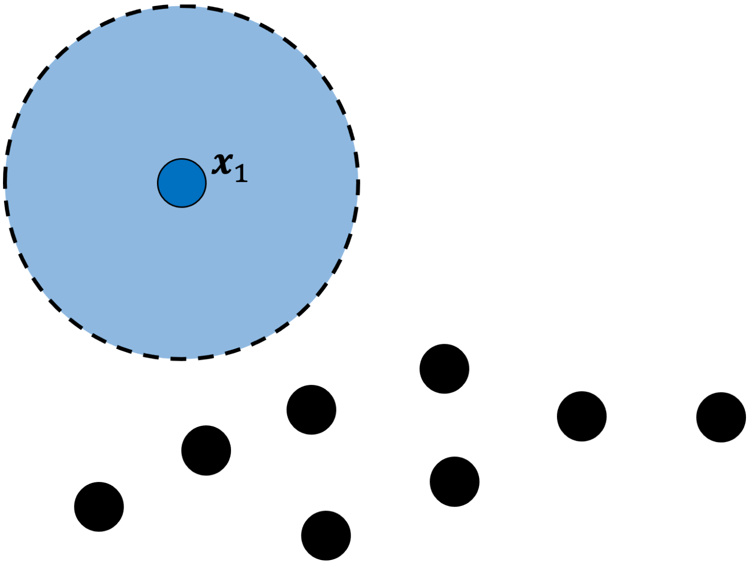
DBSCAN also requires a connectivity parameter: the **minimum number of points** *minPts* in

$$V_{\mathbf{x}} = B_d(\mathbf{x}, \varepsilon^*) \cap [\mathbf{X} \setminus \{\mathbf{x}\}]$$

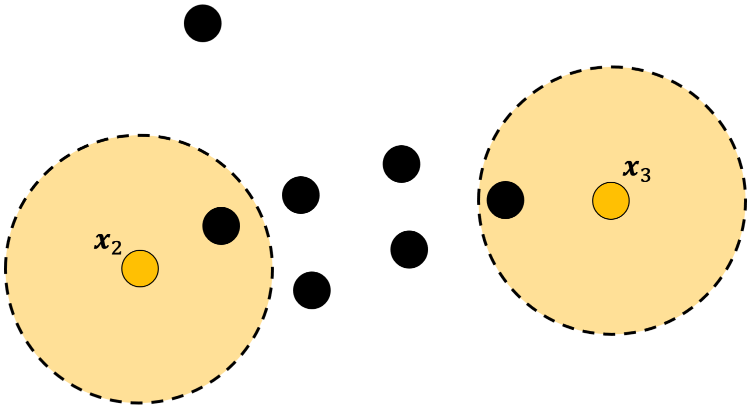
(excluding $\mathbf{x}$). If $|V_{\mathbf{x}}| \geq minPts$, the observations in $V_{\mathbf{x}}$ are said to be **within reach of** (or reachable from) $\mathbf{x}$.

In other words, for a given choice of d, ε^*, and *minPts*, there are three **types of observations** in $\mathbf{X}$:

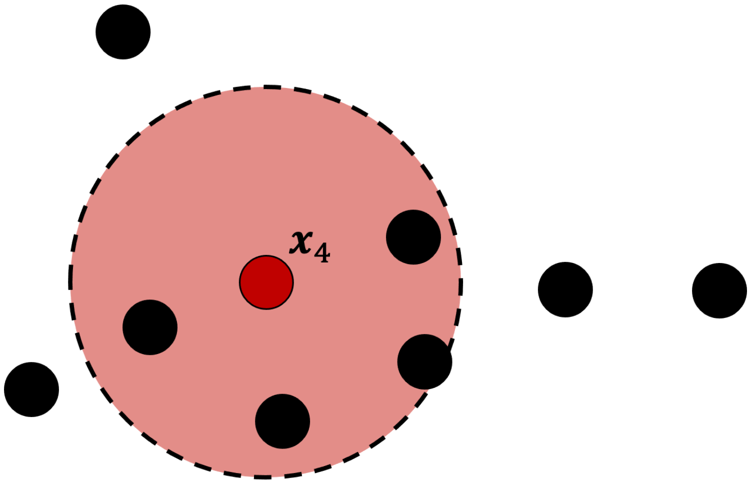
- **outliers** are observations that are not within reach of any of the other observations, such as $\mathbf{x}_1$ below:



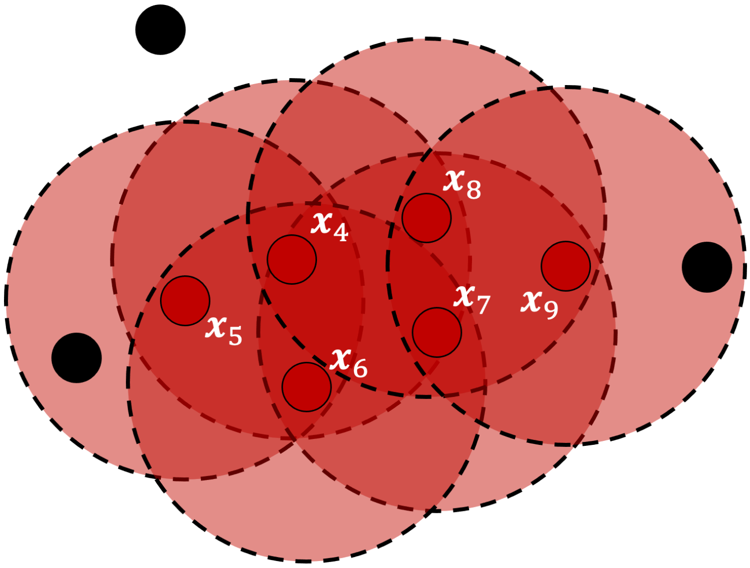
- **reachable** (non-core) **observations** are observations that are within reach of fewer than *minPts* other observations, such as $\mathbf{x}_2$ and $\mathbf{x}_3$ below (with *minPts* = 3):



- **core observations** are within reach of at least *minPts* other observations, such as $\mathbf{x}_4$ below (with *minPts* = 3):



There are other core points: $\mathbf{x}_5$, $\mathbf{x}_6$, $\mathbf{x}_7$, $\mathbf{x}_8$, and $\mathbf{x}_9$.



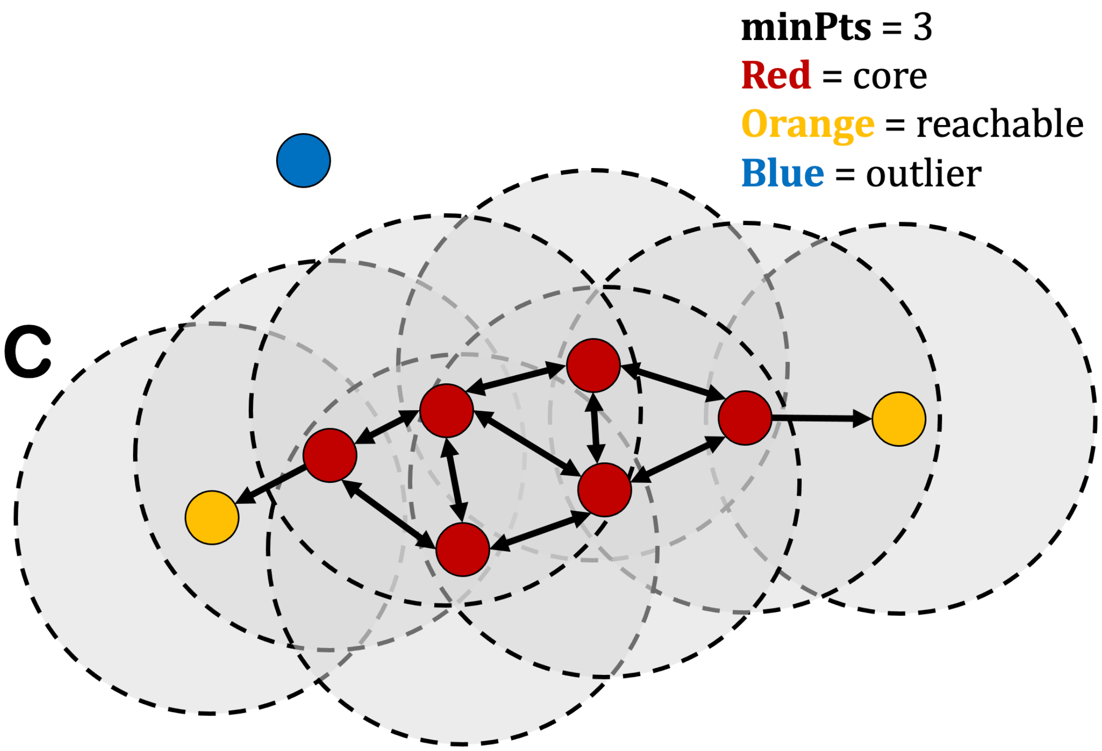


Figure 22.13: Density path connection in a DBSCAN cluster C.

Reachability is **not a symmetric relation**: no observation is reachable from a non-core point (a non-core point may be reachable, but nothing can be reached from it).

We can build a new symmetric relation on non-outlying observations on the basis of reachability, however:

$$\mathbf{p}, \mathbf{q} \in \mathbf{X} \setminus \{\text{outliers}(\mathbf{X})\}$$

are said to be **density-connected** for $\varepsilon^* > 0$ and d if there is an observation $\mathbf{o} \in \mathbf{X}$ such that $\mathbf{p}, \mathbf{q} \in V_{\mathbf{o}}$, with $|V_{\mathbf{o}}| \geq \mathit{minPts}$.

The same $\mathbf{p}, \mathbf{q}$ are said to be **density-connected in a path** if either they are density-connected or if there is a sequence of observations

$$\mathbf{p} = \mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{k-1}, \mathbf{r}_k = \mathbf{q}$$

such that $\mathbf{r}_{i-1}, \mathbf{r}_i$ is density-connected for all $i = 1, \ldots, k$.

That the latter is a relation on $\mathbf{X} \setminus \{\text{outliers}(\mathbf{X})\}$ is clear:

- it is **reflexive** as every $\mathbf{x} \in \mathbf{X} \setminus \{\text{outliers}(\mathbf{X})\}$ is either reachable or a core observation, so that $\exists \mathbf{o}_\mathbf{x} \in \mathbf{X}$ with $\mathbf{x} \in V_{\mathbf{o}_\mathbf{x}}$ and $|V_{\mathbf{o}_\mathbf{x}}| \geq \text{minPts}$, and so $\mathbf{x}$ is density-connected to itself;
- it is **symmetric** and **transitive** by construction.

DBSCAN clusters are, essentially, composed of observations that are density-connected in a path.

In the image above, arrows represent density-connection: each orange observation is within reach of a red one, but no observation can be reached from the orange points.

Algorithm DBSCAN clusters are grown as follows:

1. select an observation **at random** that has yet to be assigned to a cluster, from the list of not previously selected observations;
2. determine the selected observation's type (outlier, non-core, core);
3. if the observation is an outlier or a non-core point, assign it to the **noise cluster**;
4. else, build its network of density-connected paths;
5. assign all observations in the network to a **stand-alone cluster**;
6. repeat steps 1 to 5 until all points have been assigned to a cluster.

All points within a cluster are **mutually density-connected in a path**. If a point is reachable from any point of the cluster, it is part of the cluster as well. An illustration of the DBSCAN algorithm is provided in Figure 22.14.

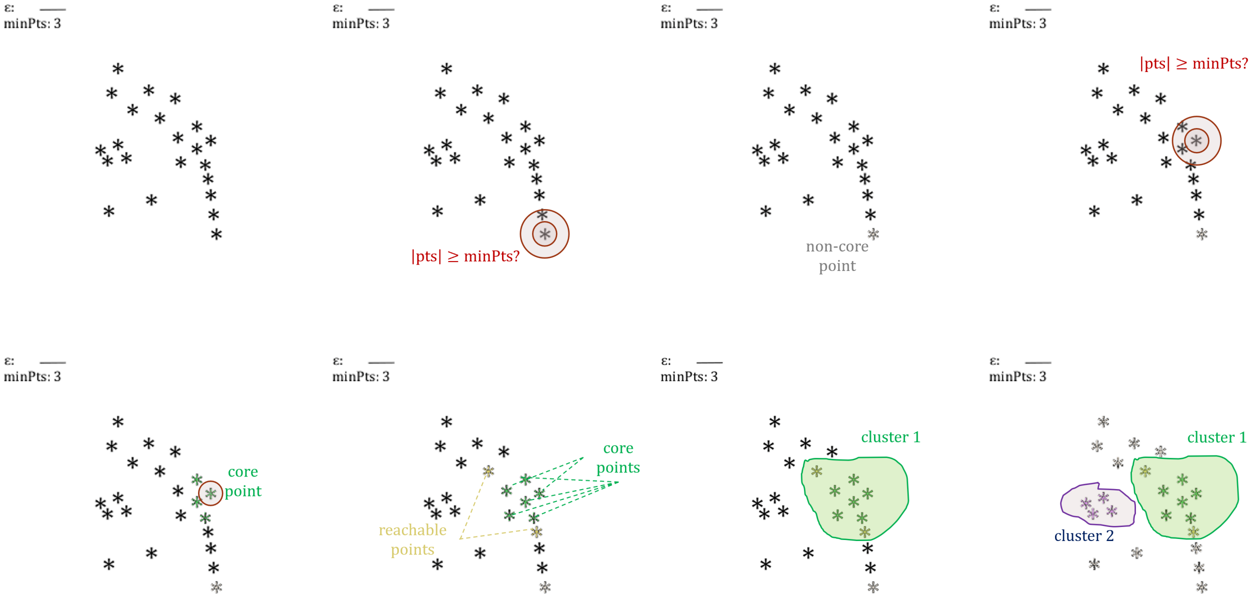


Figure 22.14: Illustration of DBSCAN on an artificial dataset (top, left). The parameters ε and *minPts* are shown in each display. We select a point at random (second image, top row); it is not a core point as its $\varepsilon-$neighbourhood does not contain more than *minPts* observations (excluding the selected point itself); it is assigned to the noise cluster. We select another point at random (top, right); that one is core point, as its $\varepsilon-$neighbourhood contains 4 observations. All its density-connected observations are shown in green (bottom, left). Its network of density-connected paths is shown in green, for the core observations, and in light green, for the reachable observations (bottom row, second image); they make up cluster 1 (bottom row, third image). Continuing on this way, we obtain 2 clusters and noisy observations (bottom, right).

The observations in the noise cluster are typically identified as **outliers**, making DBSCAN a reasonable unsupervised learning approach for anomaly detection (see Chapter 27).

Note that clusters, by definition, must contain **at least** one core point. Small groups of observations that are not density-connected to any core points will then also be assigned to the noise cluster. A non-core point that has been assigned to the noise cluster may end up being assigned to a stand-alone cluster at a later stage (but the opposite cannot occur).

It is possible for two clusters to **share** non-core points, in which case the points in question are randomly assigned (the random order of selection in step 1 may affect the results); consequently, some clusters may end up containing **fewer than** *minPts* observations.

Comments The main **advantages** of DBSCAN are that:

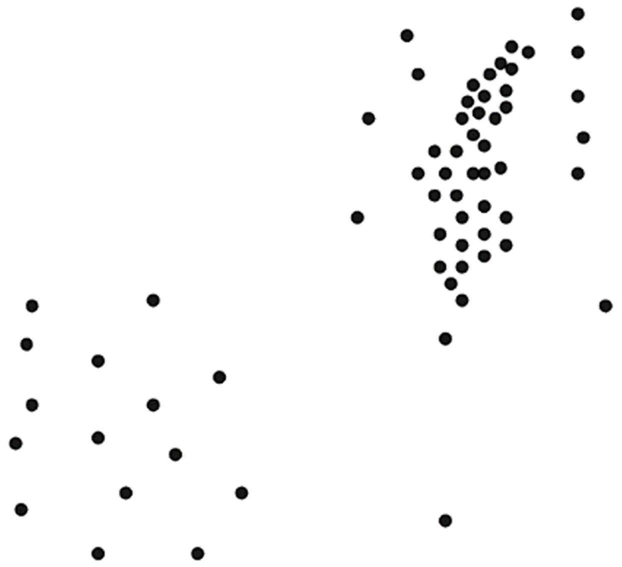
- there is no need to specify the **number** of clusters to find in the data;
- clusters of **arbitrary shapes** can be discovered;
- observations that are "noisy"/outlying are not forced into a cluster;
- the clusters are **robust** with respect to outliers, and
- it only requires two parameters ($\varepsilon^* > 0$ and *minPts*) to run properly, which can be set by domain experts if the data is well understood.

In general, it is suggested to use $minPts \geq p + 1$, with larger values being preferable for **noisy** datasets, or $minPts \geq 2p$ for large datasets or sets with duplicates. Meanwhile, the choice of $\varepsilon^* > 0$ should take into account that if it is too small, a large portion of the observations will be **assigned to the noise cluster**; but if it is too large, a majority of observations will be found in a **single cluster**. Small values are preferable, but how small is too small?

The parameter choices have a large impact on the DBSCAN results, as does the choice of the distance function, which should take place **before** ε^* is selected to avoid **data dredging** and "begging the question". Given that DBSCAN can handle globular clusters as well as non-globular clusters, why would we not always use it?

One important reason relates to **computational efficiency**. For a dataset **X** with n observations, the basic $k-$means algorithm has order $O(nk)$, whereas the most efficient versions of DBSCAN algorithm has order $O(n \log n)$. Thus, when n increases, the DBSCAN runtime increases faster than the $k-$means runtime.

Another reason is that DBSCAN works well when the density of clusters is assumed to be **constant**.



Most of us would agree that there are **two clusters** in the image above – a loose one in the bottom/left corner, and a tight one in the top/right corner – as well as some **outliers** around the tight cluster, but no combination of $\varepsilon^* > 0$ and *minPts* can allow DBSCAN to discover this structure: either it finds no outliers, or it only finds the one tight cluster.

Example We re-visit the (scaled) 2011 Gapminder dataset: we use Euclidean dissimilarity in this example, but the `dbscan()` function from the `fpc` package in `R` can accommodate other metrics: we first compute the corresponding distance matrix and specify `method="dist"` instead of `method="raw"` in the function call.

We will use 9 combinations of parameters

$$(\varepsilon^* \in \{0.75, 1, 1.25\}) \times (minPts \in \{6, 10, 15\}).$$

```
set.seed(0) # for replicability
dbscan1 <- fpc::dbscan(gapminder.SoCL.2011.s, eps = 0.75, MinPts = 6)
dbscan2 <- fpc::dbscan(gapminder.SoCL.2011.s, eps = 1.0, MinPts = 6)
dbscan3 <- fpc::dbscan(gapminder.SoCL.2011.s, eps = 1.25, MinPts = 6)
dbscan4 <- fpc::dbscan(gapminder.SoCL.2011.s, eps = 0.75, MinPts = 10)
dbscan5 <- fpc::dbscan(gapminder.SoCL.2011.s, eps = 1.0, MinPts = 10)
dbscan6 <- fpc::dbscan(gapminder.SoCL.2011.s, eps = 1.25, MinPts = 10)
dbscan7 <- fpc::dbscan(gapminder.SoCL.2011.s, eps = 0.75, MinPts = 15)
dbscan8 <- fpc::dbscan(gapminder.SoCL.2011.s, eps = 1.0, MinPts = 15)
dbscan9 <- fpc::dbscan(gapminder.SoCL.2011.s, eps = 1.25, MinPts = 15)
```

No doubt there are more efficient ways to go through the 9 combinations, but this will do for the purpose of illustration.

```
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(dbscan1$cluster)),
                diag=list(continuous=my_dens))
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(dbscan2$cluster)),
                diag=list(continuous=my_dens))
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(dbscan3$cluster)),
                diag=list(continuous=my_dens))
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(dbscan4$cluster)),
                diag=list(continuous=my_dens))
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(dbscan5$cluster)),
                diag=list(continuous=my_dens))
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(dbscan6$cluster)),
                diag=list(continuous=my_dens))
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(dbscan7$cluster)),
                diag=list(continuous=my_dens))
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(dbscan8$cluster)),
                diag=list(continuous=my_dens))
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(dbscan9$cluster)),
                diag=list(continuous=my_dens))
```

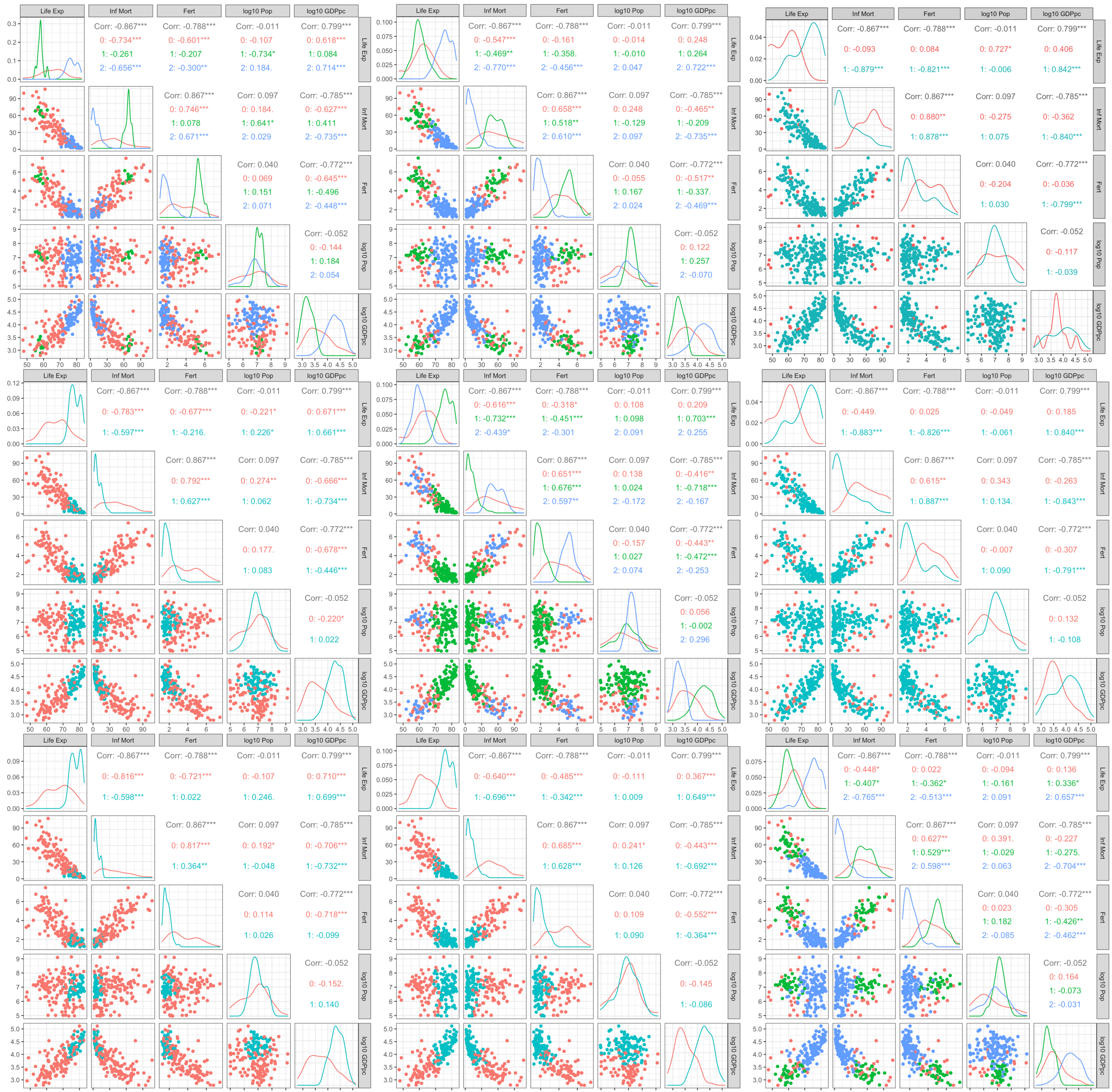


Figure 22.15: Realizations of DBSCAN on the (scaled) 2011 Gampinder data: $\varepsilon = 0.75$ (first column), 1 (second column), 1.25 (third column); $minPts = 6$ (first row), 10 (second row), 15 (third row).

The noisy observations are shown in red: one immediate insight is that the number of outlying observations **decreases** as ε^* increases, which is as expected. Another insight is that the number of noisy observations **increases** as *minPts* increases, which is again not surprising.

If we compare the shape of the DBSCAN clusters with those of the $k-$means and HC clusters, we notice that the option of identifying observations as noisy – coupled with the "right" combination of parameters – creates "reasonable" clusters, that is to say, clusters for which we do not have to stretch our ideas about what clusters ought to look like: the problematic observations[41] are simply explained away as outliers.

The various runs find either 1 or 2 stand-alone clusters (as well as noisy observations), but that can change if we use different parameter values.

We can also determine if the cluster observations are core or non-core observations. In the realization with $\varepsilon^* = 1$ and $minPts = 6$, we have:

	noise	**cluster 1**	**cluster 2**
outlier	34	–	–
reachable	–	10	17
core	–	20	103
total	**34**	**30**	**120**

22.4.2 Spectral Clustering

At a fairly coarse level, clustering algorithms are divided along those focusing on **compactness** and those focusing on **connectivity**.

Compactness methods are usually variants of k **Nearest Neighbours** (kNN) methods (see Section 21.1.3), and are effective when there are distinct clumps in the data. We can make specific assumptions about the distribution of the different clusters ahead of time (as in the next section), but compact methods struggle to achieve meaningful results in scenarios where groups are not **linearly separable**.

In cases where we have little to no knowledge of the dataset, making assumptions about the distributions of clusters can lead to invalid clustering schemes; in such cases, connectivity-based methods have been shown to work reasonably well [2, 273].

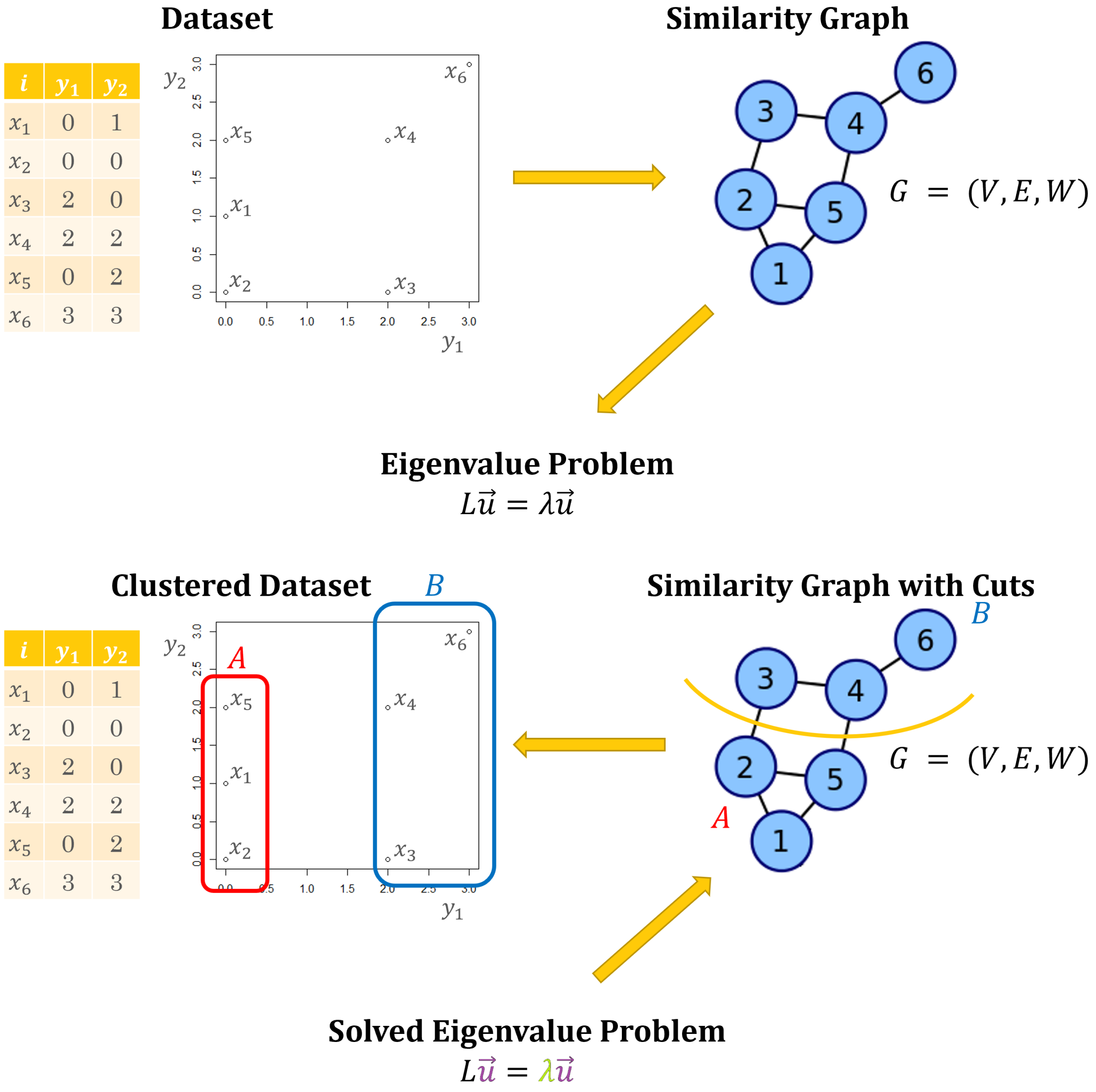
Connectivity methods, such as DBSCAN, focus on dividing observations into groups based on their **similarity graphs**; observations that are quite different in their features[42] may end up in the same cluster if there is a chain of sufficiently similar observations linking them.

Connectivity methods require fewer initial assumptions, but their use can be harder to justify mathematically. The validity of such methods can only be determined *post hoc*.

Spectral clustering is a connectivity method that has become quite popular in practice; in a nutshell, we transform the dataset into its **similarity graph** and convert the latter into an **eigenvalue problem**. We then solve the eigenvalue problem, convert the solution into a **graph cut**, and then translate the cut back into dataset **clusters** (as illustrated in Figure 22.16).

[41] Like China and India in regards to population, say.

[42] And as such would be differentiated using **compactness** methods.



i	y_1	y_2
x_1	0	1
x_2	0	0
x_3	2	0
x_4	2	2
x_5	0	2
x_6	3	3

Dataset
y_2
y_1
0.0 0.5 1.0 1.5 2.0 2.5 3.0
x_1 x_2 x_3 x_4 x_5 x_6
Similarity Graph
1 2 3 4 5 6
$G = (V, E, W)$
Eigenvalue Problem
$L\vec{u} = \lambda\vec{u}$
Clustered Dataset
A
B
Similarity Graph with Cuts
Solved Eigenvalue Problem
$L\vec{u} = \lambda\vec{u}$

Figure 22.16: Schematics of spectral clustering. We extract the similarity graph of a dataset, which gives rise to an eigenvalue problem (top). The eigenenvalue problem is then solved, which suggests an 'optimal' graph cut, which in turns leads to data clusters (bottom).

Before we start delving into the spectral clustering algorithm, we must discuss a few concepts relating to graphs and linear algebra.[43]

Graphs and Cuts A **graph** is an object which connects **nodes** (or **vertices**) together through **edges**. The edges have **weights** and can also be **directed**. In certain cases, we may assume that all edge weights are identical and bidirectional, which is equivalent to saying that the edges just represent that a relationship exists.

Airports (vertices) and flight paths (edges) form a graph in transportation networks, as do people (vertices) and relationships (edges) in social networks; the edges can be weighted according to flight frequency and/or directed according to their origin and destination, say, in the transportation example.

In the social network example, they could be weighted according to frequency of communication and/or directed according to who follows who on some app.

The link with clustering is that once a similarity measure w has been selected, a dataset can be represented by a **similarity graph** $G = (V, E, W)$:

1. observations $\mathbf{x}$ correspond to **vertices** $v \in V$;
2. if $i \neq j$, vertices $v_i, v_j \in V$ are connected by an **edge** $e_{i,j} = 1$ if the **similarity weight** $w_{i,j} = w(\mathbf{x}_i, \mathbf{x}_j) > \tau$ for a predetermined **threshold** $\tau \in [0, 1)$, and by no edge ($e_{i,j} = 0$) otherwise;[44]
3. the edges $(e_{i,j})$ form the **adjacency matrix** E;
4. the weights $(w_{i,j})$ form the **similarity matrix** W;
5. the **(diagonal) degree matrix** D provides information about the number of edges attached to a vertex: $d_{i,i} = \sum_{j=1}^{n} e_{i,j}$.

As an example, we could use the **Gower similarity measure**

$$w(\mathbf{x}_i, \mathbf{x}_j) = 1 - \frac{1}{p}\sum_{k=1}^{p} \frac{|x_{i,k} - x_{j,k}|}{\text{range of } k\text{th feature in } \mathbf{X}}$$

on the dataset found in Figure 22.16; the ranges of X_1 and X_2 are both $r_1 = r_2 = 3$, so that

$$\begin{aligned} w_{3,4} = w_{4,3} = w(\mathbf{x}_3, \mathbf{x}_4) &= 1 - \frac{1}{2}\left(\frac{|x_{3,1} - x_{4,1}|}{r_1} + \frac{|x_{3,2} - x_{4,2}|}{r_2}\right) \\ &= 1 - \frac{1}{2}\left(\frac{|2-2|}{3} + \frac{|0-2|}{3}\right) = 1 - \frac{1}{2}\cdot\frac{2}{3} = \frac{2}{3}; \end{aligned}$$

the similarity matrix as a whole is

$$W = \begin{pmatrix} 0 & 5/6 & 1/2 & 1/2 & 5/6 & 1/6 \\ 5/6 & 0 & 2/3 & 1/3 & 2/3 & 0 \\ 1/2 & 2/3 & 0 & 2/3 & 1/3 & 1/3 \\ 1/2 & 1/3 & 2/3 & 0 & 2/3 & 2/3 \\ 5/6 & 2/3 & 1/3 & 2/3 & 0 & 1/3 \\ 1/6 & 0 & 1/3 & 2/3 & 1/3 & 0 \end{pmatrix}.$$

[43] These concepts are covered in just enough depth to provide an intuition about the algorithm.

[44] Note that, by convention, $w_{i,i} = 0$ for all i.

If we use a threshold value of $\tau = 0.6$, say, then the adjacency matrix is

$$E = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix},$$

and the degree matrix is

$$D = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

The degree matrix can also be read directly from the similarity graph (which depends on the threshold τ), by counting the number of edges at each node (see Figure 22.16).

A **graph cut** is the process by which we remove edges from the graph and separate the vertices into into groups (or **sub-graphs**).

The clustering task is to separate the nodes into multiple groups by **minimizing the total weight of the edges** we have to break in the process (i.e., making sure that the groups are as dissimilar as possible). This is also known as the **minimum cut problem** (MinCut).[45]

This task is NP-Hard, which means that there is no theoretically guaranteed efficient way to do so, in comparison to simply testing every possible cut and finding the minimum weight. This is problematic: for datasets with n observations, the number of cuts is **bounded below** by 2^n (when we only consider $2-$cuts); when n is relatively small, the overall number of cuts to consider remains manageable, but for nearly all reasonable datasets, the size of n turns this task into an exercise in futility.

The clustering approach generalizes the MinCut problem (or any of the other problems) by imposing some properties on the similarity graph to ensure that we can approximate the true MinCut solution in a **computationally efficient** manner.[46]

Formally, the MinCut problem involves finding a **partition** $\{A_1, ..., A_k\}$ of G which minimizes the objective function

$$\text{Cut}(A_1, ..., A_k) = \frac{1}{2}\sum_{i=1}^{k} \mathcal{W}(A_i, \overline{A_i}),$$

where

$$\mathcal{W}(A, B) = \sum_{i\in A, j\in B} w_{i,j}$$

and $\overline{A}$ is the (set-theoretic) complement of A. The factor $\frac{1}{2}$ is there to remove double-counted edges.

The spectral clustering approach instead solves the **Normalized Cut** (NCut) problem, which is similar to the MinCut problem except that we

[45] 45: This cannot be the entire story, however, as we can minimize the total weight of broken edges by simply ... not cutting any edges. Indeed, there are other approaches: **Normalized Cut** (actually used in practice), Ratio Cut, Min-Max Cut, etc.

[46] 46: The spectral MinCut solution is not guaranteed to be the true MinCut solution, but it usually is close enough to be an acceptable approximation.

are minimizing the weight of edges escaping a cluster relative to the total weights in the cluster.[47]

In the NCut problem, the **objective function** is

$$J_{\text{NCut}}(A,B)=\text{Cut}(A,B)\left(\frac{1}{\text{Vol}(A)}+\frac{1}{\text{Vol}(B)}\right),$$

where

$$\text{Vol}(C)=\sum_{i\in C} w_{i,*};$$

in a first pass, we seek to **minimize** J_{NCut} against the set of **all possible partitions** (A,B) of G. The procedure can be repeated as often as necessary on the cluster sub-graphs.

Intuitively, J_{NCut} is small when the observations **within** each sub-graph are **similar** to one another ($\text{Vol}(A)$, $\text{Vol}(B)$ are large) and the observations **across** are dissimilar to one another ($\text{Cut}(A,B)$ is small).

On the plus side, takes into consideration the **size of the partitioned groups** and **intra-group variance**, and tends to avoid isolating vertices, but it is not any easier to solve than the MinCut problem. So why do we even bring it up in the first place? As it happens, we can provide an approximation to the NCut solution using **purely algebraic means**.

Similarity, Degree, and Laplacian Matrices There are different ways to construct a **graph** representing the relationships between the dataset's observations. We can use:

- **fully connected** graphs, where all vertices having non-zero similarities are connected to each other;
- $r-$**neighbourhood** graphs, where each vertex is only connected to the vertices falling inside a ball of radius r (according to some distance metric d), where r has to be tuned to capture the local structure of data;
- k **nearest neighbours** graphs (and variants), where each vertex is connected to its k nearest neighbours (again, according to some distance metric d), with k pre-selected, and
- **mixtures of** $r-$**neighbourhood and** k**NN** graphs, to better capture sparsity in the data.

The similarity measure w is usually picked from a list that includes: Gaussian (most common), cosine, fuzzy, Euclidean, Gower, etc. The similarity matrix W is symmetric and has zeros along the diagonal; its non-diagonal entries represent the **similarity strength** between the corresponding graph vertices.[48]

We have discussed previously how to build the adjacency matrix E from W and a threshold $\tau\in[0,1)$. The only component of a graph that similarity matrices do not directly capture are the **degrees** of each vertex, the number of edges that enter it.[49]

The **diagonal** of the degree matrix D holds that information for each vertex. We can combine W and D (or E and D) to create a matrix L known as the **Laplacian**, which has properties linked to the topology of the similarity graph.

[47] For more information about this abstraction, which actually links a variant of Kernel PCA to spectral clustering, consult [274].

[48] And so beteween the corresponding observations in the dataset.

[49] We are viewing the similarity graph as **undirected**.

The **Laplacian** of a graph is defined by

$$L_0 = D - \Theta, \quad \Theta \in \{E, W\};$$

the **symmetric Laplacian** by

$$L_S = D^{-1/2} L D^{-1/2} = \mathbf{I}_n - D^{-1/2} \Theta D^{-1/2},$$

and the **asymmetric Laplacian** by

$$L_A = D^{-1} L = \mathbf{I}_n - D^{-1} \Theta.$$

In all cases, the **off-diagonal entries** are non-positive, and the **diagonal entries** contain the degree of each node.

The Laplacians have the following useful properties:

- L_0, L_S are symmetric; L_A is not necessarily so;[50]
- all their eigenvalues are real and non-negative;
- every row and column adds up to 0, which means that $\lambda_0 = 0$ is the smallest eigenvalue of each Laplacian (hence they are singular and cannot be inverted);
- the number of connected components in the graph is the **dimension of the nullspace** of the Laplacian associated to $\lambda_0 = 0$ (which may provide a first approximation to the number of clusters in $\mathbf{X}$), and
- the second smallest eigenvalue gives the graph's **sparsest cut**.[51]

Algorithm In the case of two clusters, the objective function J_{NCut} is minimized when finding the eigenvector $\mathbf{f}$ corresponding to the smallest **positive** eigenvalue of L, also known as the **spectral gap**.[52]

The clustering in the original data is recovered by sending $\mathbf{x}_i$ to A when $f_i > 0$ and $\mathbf{x}_j$ to B otherwise. This deterministic algorithm is a special case of the **spectral clustering algorithm** [275].

To split $\mathbf{X}$ into k clusters, we follow the steps below:

1. form a similarity matrix W and a degree matrix D using a threshold $\tau \in [0, 1)$;
2. construct a Laplacian L_ξ, $\xi \in \{0, S, A\}$, using $\Theta = W$;
3. compute the first k eigenvectors $\{\mathbf{u}_1, ..., \mathbf{u}_k\}$ of L_ξ corresponding to the k **smallest positive** eigenvalues of L_ξ;
4. construct the $n \times k$ matrix $\mathbf{U}$ containing the vectors $\{\mathbf{u}_1, ..., \mathbf{u}_k\}$ as **columns**;
5. normalize the rows of $\mathbf{U}$ into a matrix $\mathbf{Y}$ with rows $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$ having unit length;
6. cluster the rows of $\mathbf{Y}$ into k clusters;
7. assign $\mathbf{x}_i$ to cluster j of $\mathbf{X}$ if $\mathbf{y}_i$ was assigned to cluster j in the preceding step.

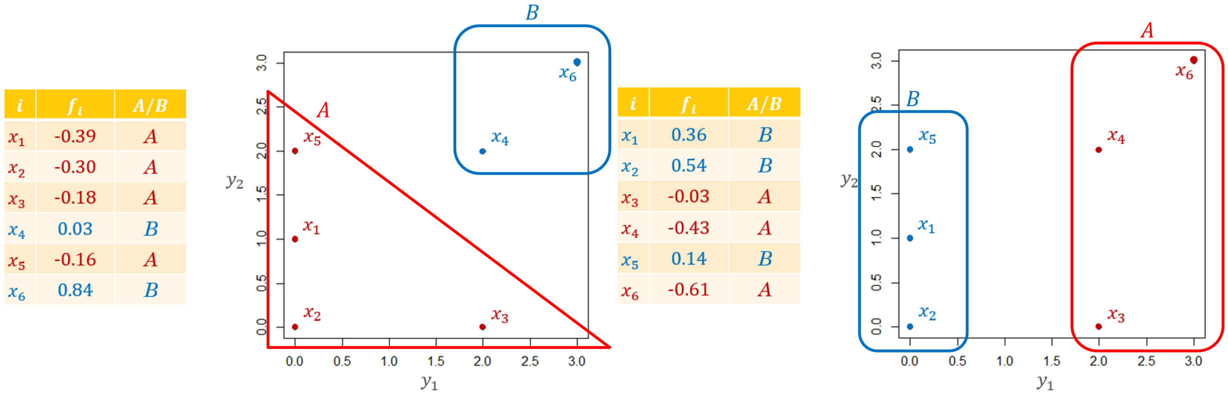
Spectral clusters for the dataset of Figure 22.16, computed using the Laplacian and symmetric Laplacian, are shown in Figure 22.17.

From an experimental perspective, spectral clustering provides an attractive approach because it is easy to implement and reasonably fast, especially for sparse datasets: it is a **graph partitioning problem** that makes no initial assumptions on the form of the data clusters.

[50] Since the product of symmetric matrices is not necessarily symmetric.

[51] This is not the same as the minimum cut which represents the cut that minimizes the number of edges separating two vertices, but instead represents the minimum ratio of edges across the cut divided by the number of vertices in the smaller half of the partition.

[52] This notion will also play a role in Section 23.4.3.



i	f_i	A/B
x_1	-0.39	A
x_2	-0.30	A
x_3	-0.18	A
x_4	0.03	B
x_5	-0.16	A
x_6	0.84	B

i	f_i	A/B
x_1	0.36	B
x_2	0.54	B
x_3	-0.03	A
x_4	-0.43	A
x_5	0.14	B
x_6	-0.61	A

Figure 22.17: Two clusters for the artificial dataset: simple Laplacian (left); symmetric Laplacian (right).

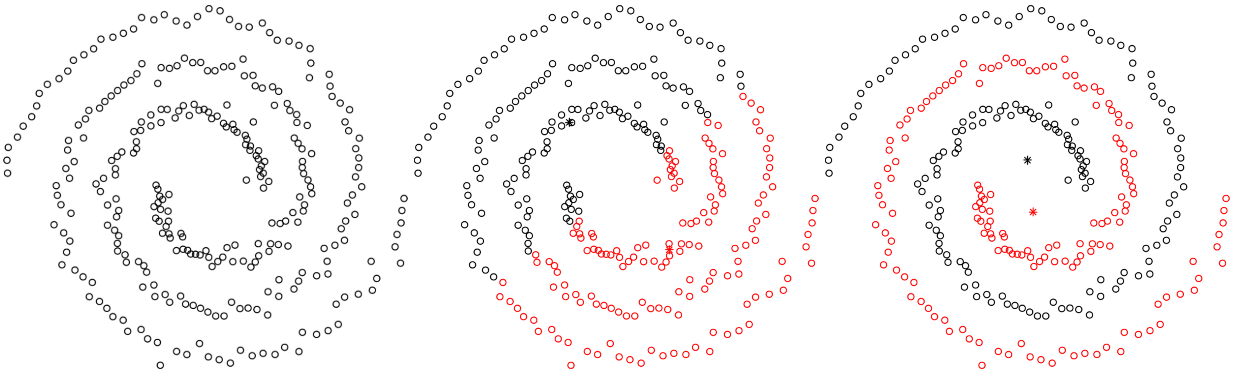


Figure 22.18: Comparing 2−means (middle) and spectral clustering with $k = 2$ (right) on the spirals dataset (left).

Spectral clustering has variants, which depend on the many choices that can be made at various points in the process:

1. **pre-processing** (choice of: number of cluster k, similarity measure w, threshold τ);
2. **spectral representation** (choice of Laplacian);
3. **clustering algorithm** (choice of compact-based, potentially non-deterministic algorithm to unleash on the rows of the representation $\mathbf{Y}$).

The **NJW algorithm** uses L_S for the spectral representation and k−means as a clustering approach. It can be interpreted as **kernalized k−means**: if we select a kernel which transforms the points to their mapped value in the Laplacian of the graph, then we (almost directly) recover spectral clustering [274].[53]

In Figure 22.18, the different outcomes of k−means and NJW are illustrated on the `spirals` dataset (available in `R`).

Practical Details and Limitations The most obvious practical detail in the implementation of spectral clustering is related to the construction of the similarity graph. In general, there is virtually no theoretical justification for determining what type of clustering approach to use; even after an approach has been selected, it can be quite difficult to choose appropriate parameter values.

[53] 53: DBSCAN can also fit within that framework, by picking a similarity method based on the radius that allows the graph to separate into different components. Then the multiplicity of $\lambda_0 = 0$ in the Laplacian gives the number of graph components, and these can be further clustered, as above.

In spectral clustering, there are considerations in favour of using **sparse similarity/adjacency matrix**: we seek to strike a balance between a Laplacian which is too densely connected, and one for which almost all of the observations are seen as dissimilar to one another. Another issue relates to the computational challenge of **finding the eigenvalues** of the Laplacian.

This can be done relatively efficiently if the matrix is sparse enough, however, which suggests using a relatively-high threshold τ; there are methods which help spectral clustering automatically tune for the best parameter values (including τ), but they take up a significant amount of resources [275].

Spectral clustering methods are extremely effective because they do not require assumptions about **distributions** and **centres**, are fairly **easy to implement**, and are **transparent** and **interpretable**.

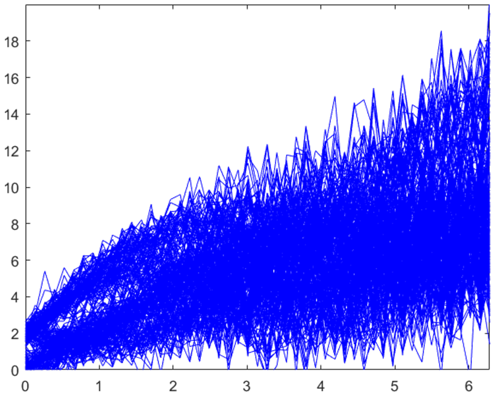
However, they suffer from some of the same drawbacks as other clustering methods, namely when it comes to:

- selecting **initial parameter values**,
- run-times that **do not scale** with larger datasets, and
- determining optimal ways to visualize the results.

As in all clustering scenarios, analysts are faced with decisions at various levels of the process; they must be prepared to run multiple algorithms, in multiple configurations, in order to get a sense for the data structure.[54]

Examples In a first example, spectral clustering is used to segment greyscale images into different segments based on contrasting colours [276]. Figure 22.19 shows instances with high contrast, with fairly decent segmentation performance using NCut, Self-Tuning SC [277], and a proposed SC algorithm [276]; Figure 22.20 shows other instances with less contrast (resulting in a poorer segmentation with the same methods); Figure 22.21 shows the comparison in segmentations using the proposed SC algorithm when the same image is presented at different resolutions.

In the second example, consider a dataset of $n = 250$ times series, with $N = 60$ entries each (see below).



[54] Some strategies specific to spectral clustering are presented in [275].

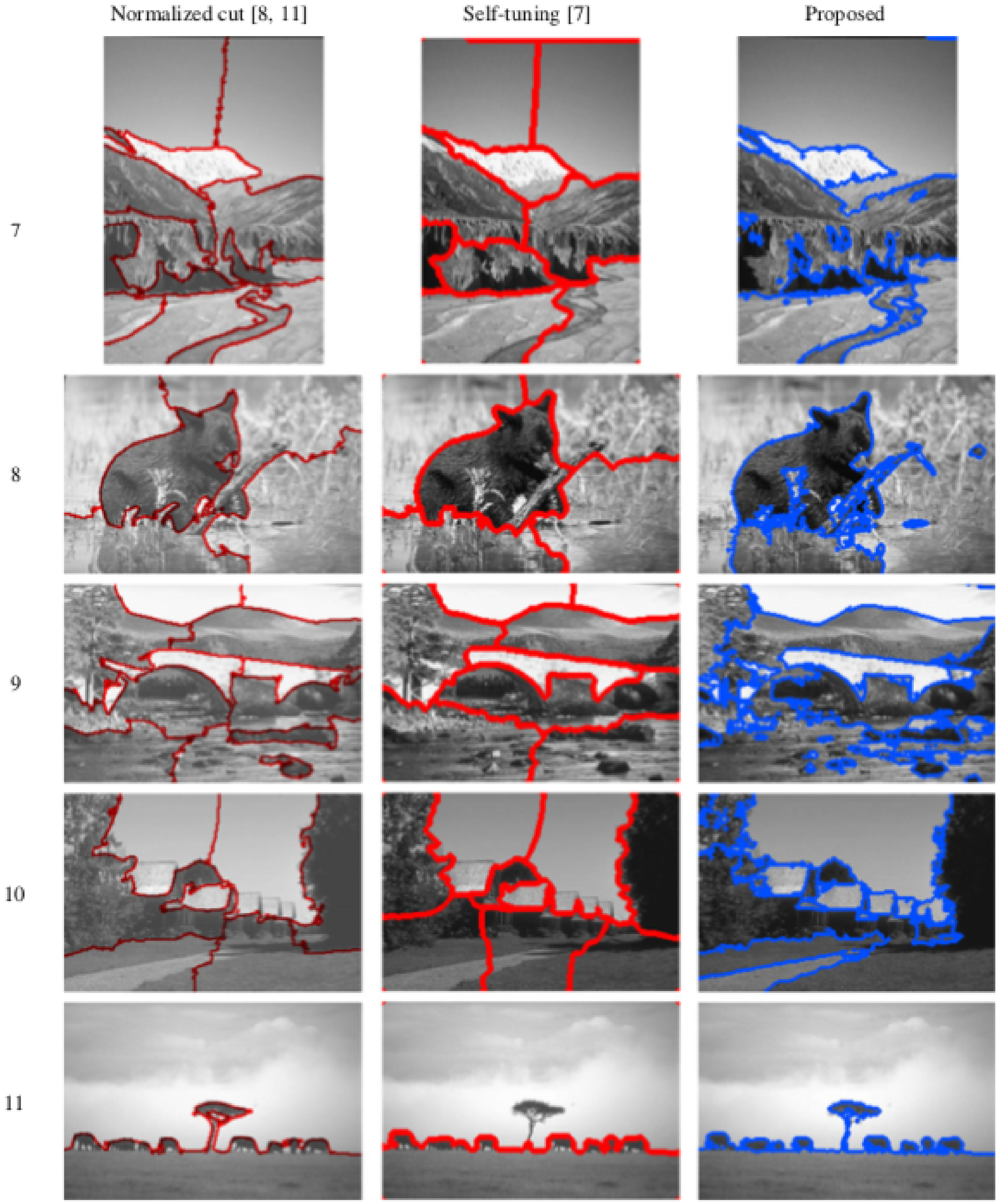


Figure 22.19: High-contrast image segmentation with spectral clustering [276].

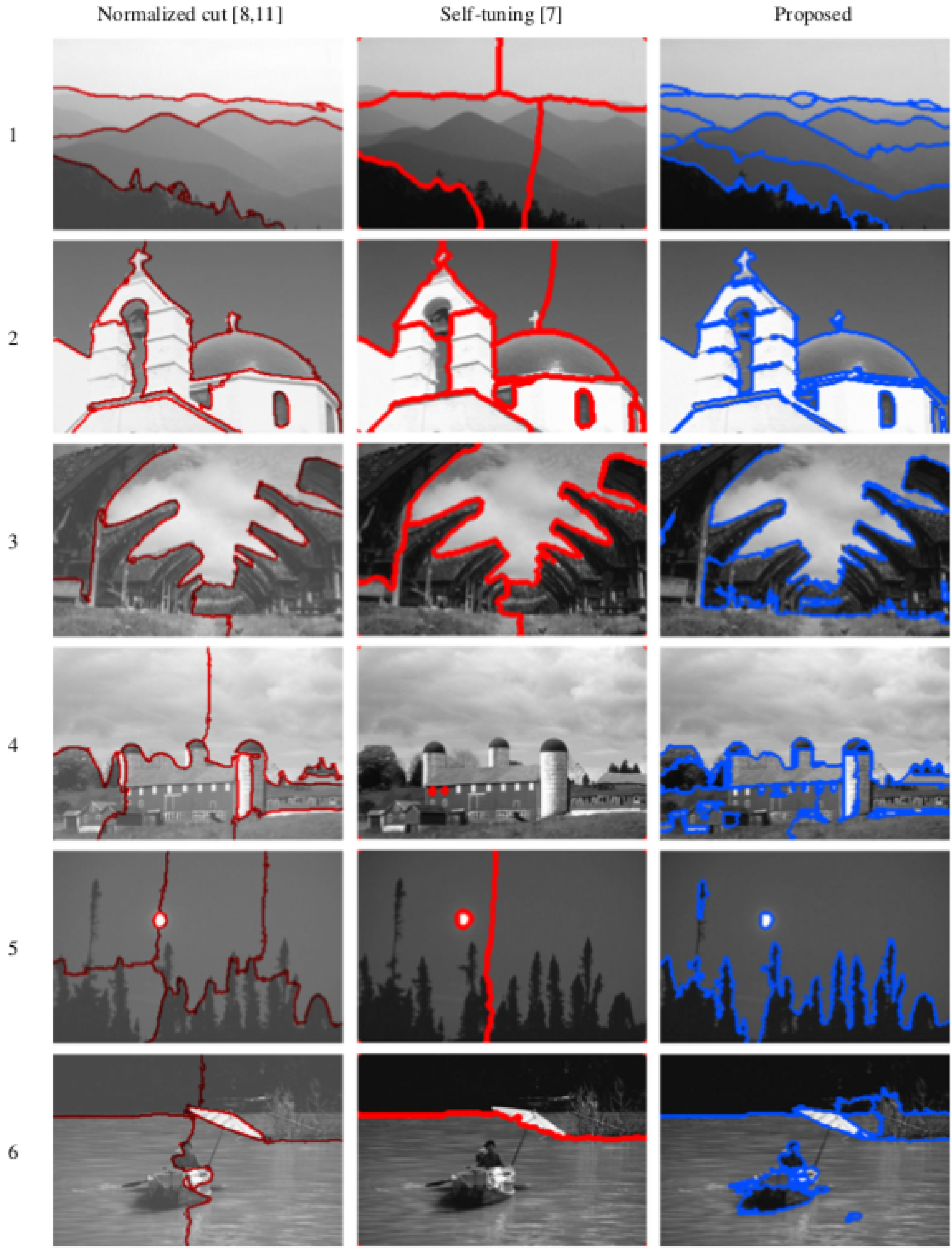


Figure 22.20: Low-contrast image segmentation with spectral clustering [276].

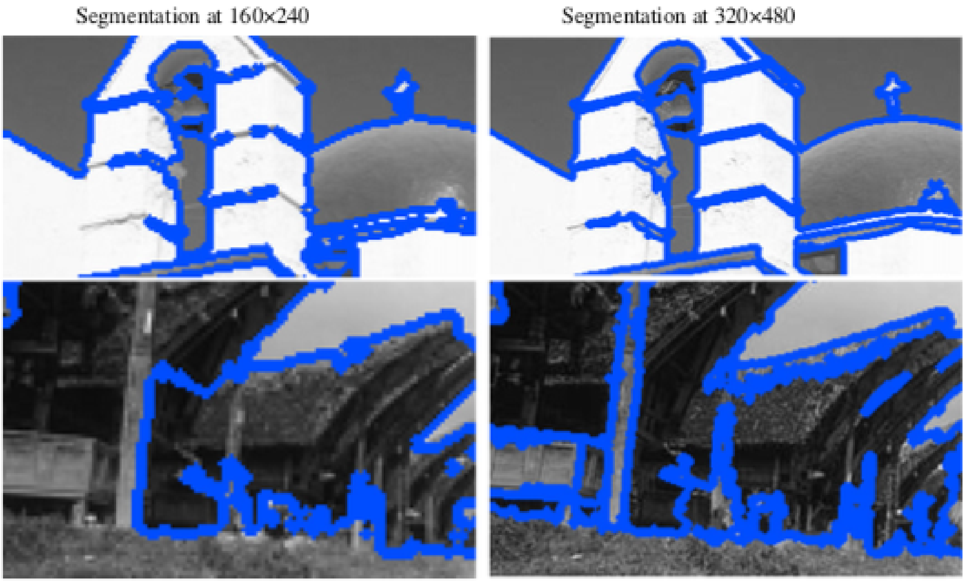


Figure 22.21: Spectral clustering image segmentation of images at different resolutions [276].

We use the **average absolute gap** as distance d:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{60}\sum_{\ell=1}^{60} |x_{i,\ell} - x_{j,\ell}|.$$

We build the **Gaussian similarity** measure

$$w(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{d^2(\mathbf{x}_i, \mathbf{x}_j)}{2\sigma^2}\right),$$

and we use the following parameter values

$$\sigma^2 = 300, \quad \tau = 0.9, \quad k = 5.$$

The spectral clustering results are quite appealing, as can be seen in the first realization of the NJW algorithm with $k = 5$ clusters. Note however that not every run of the algorithm yields an outcome that we would consider meaningful (see Figure 22.22).

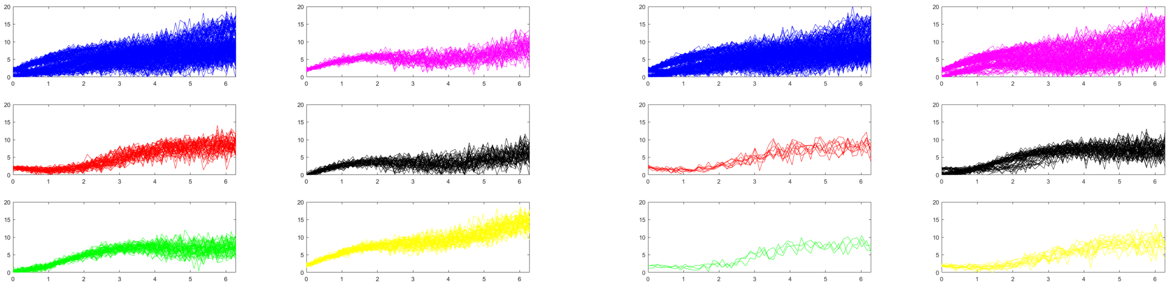
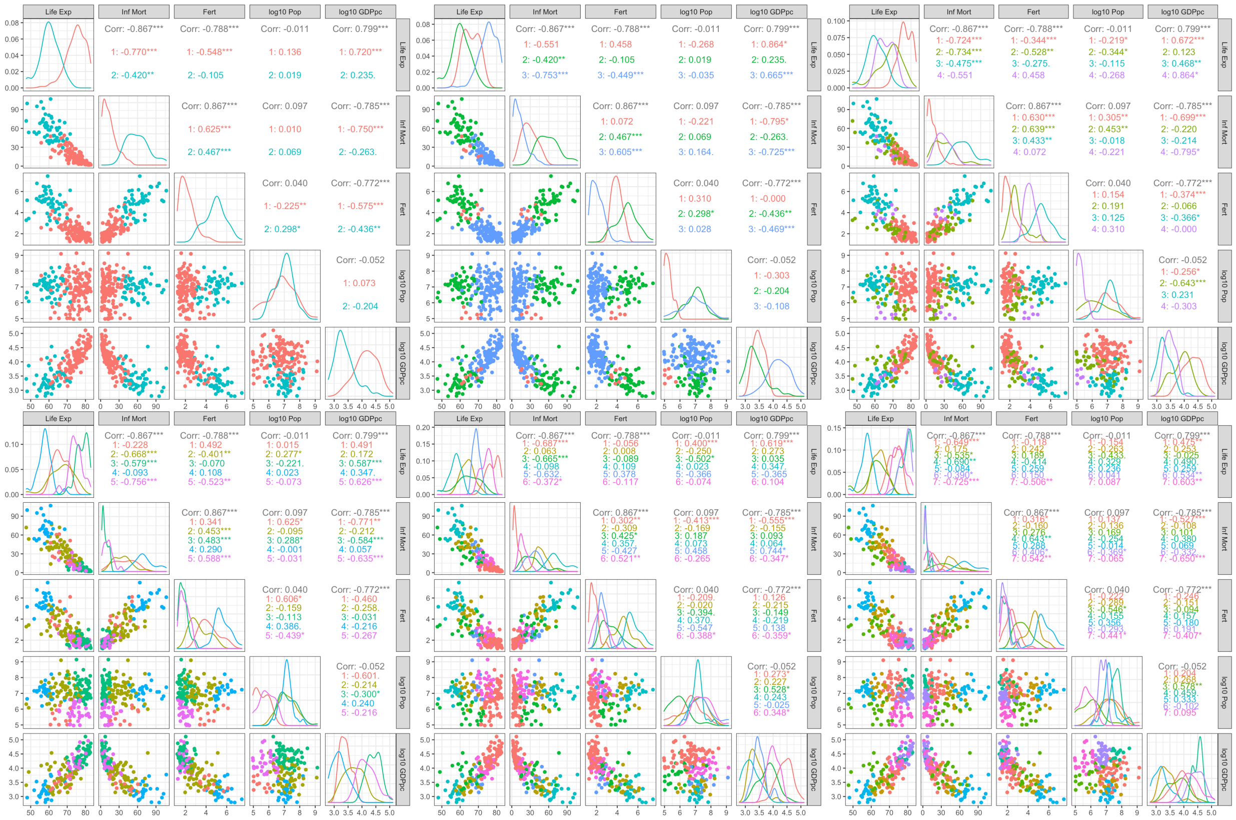


Figure 22.22: Two realizations of spectral clustering, using the NJW algorithm with $k = 5$; the original dataset is shown in blue. We see that the NJW algorithm has captured 5 clusters with different times series characteristics, which is an encouraging result (two leftmost columns); the $k-$means portion of the algorithm leads to different clusters, which appear to be of lower quality (two rightmost columns).

In the final example, we once again revisit the (scaled) 2011 Gapminder dataset using Euclidean dissimilarity. We use the `kernlab` implementation of the NJW algorithm found in `specc()`, with the default settings. We run one instance of the algorithm for $k = 2$ to $k = 7$ clusters.[55]

[55] 55: Assume that the libraries `ggplot2` and `GGally` hgave already been loaded.

```
sc.gapminder.2 <- kernlab::specc(as.matrix(gapminder.SoCL.2011.s), 2)
ggpairs(gapminder.SoCL.2011[,c(3:7)],
        aes(color=as.factor(sc.gapminder.2)), diag=list(continuous=my_dens))

sc.gapminder.3 <- kernlab::specc(as.matrix(gapminder.SoCL.2011.s), 3)
ggpairs(gapminder.SoCL.2011[,c(3:7)],
        aes(color=as.factor(sc.gapminder.3)), diag=list(continuous=my_dens))

sc.gapminder.4 <- kernlab::specc(as.matrix(gapminder.SoCL.2011.s), 4)
ggpairs(gapminder.SoCL.2011[,c(3:7)],
        aes(color=as.factor(sc.gapminder.4)), diag=list(continuous=my_dens))

sc.gapminder.5 <- kernlab::specc(as.matrix(gapminder.SoCL.2011.s), 5)
ggpairs(gapminder.SoCL.2011[,c(3:7)],
        aes(color=as.factor(sc.gapminder.5)), diag=list(continuous=my_dens))

sc.gapminder.6 <- kernlab::specc(as.matrix(gapminder.SoCL.2011.s), 6)
ggpairs(gapminder.SoCL.2011[,c(3:7)],
        aes(color=as.factor(sc.gapminder.6)), diag=list(continuous=my_dens))

sc.gapminder.7 <- kernlab::specc(as.matrix(gapminder.SoCL.2011.s), 7)
ggpairs(gapminder.SoCL.2011[,c(3:7)],
        aes(color=as.factor(sc.gapminder.7)), diag=list(continuous=my_dens))
```



None of our clustering attempts have found what one might call **natural groups** in the 2011 Gapminder data. We might not have hit on the right method yet... but at what point do we decide that the task is futile and no such groups exist in the first place?

22.4.3 Probability Clustering

In contrast with the model-free approach of density-based clustering and spectral clustering, **probabilistic-based clustering** attempts to optimize the fit between the observed data and some **mathematical model** of clustering, with the assumption that the data is generated *via* a number of underlying probability distributions.

In practice, we assume that clusters are represented by **parametric probability distributions**, and the objective is to **learn the parameters** for each of these distributions. This assumption allows us to use probability theory to derive learning formulas for the parameters.[56]

Mixture Models The main underlying assumption of **mixture models** is that each observation is drawn (or generated) from one of several mechanisms (or components). In **model-based clustering**, we learn the parameters that provide the optimal fit to the data; in other words, we make a series of predictions about which component(s) generated each of the observations.

This naturally leads to **clusters**, all observations generated by a given component belonging to the same cluster. Formally, we let

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \in M_{n,p}(\mathbb{R}).$$

Assume that there are k mechanisms that generate data, and that each of them is determined by a vector of parameters $\boldsymbol{\theta}_\ell$, $1 \leq \ell \leq k$.

For $1 \leq j \leq n$, denote the probability of $\mathbf{x}_j$ being **generated by the ℓ−th mechanism**, $1 \leq \ell \leq k$, by

$$P(\mathbf{x}_j \mid \boldsymbol{\theta}_\ell).$$

The mixture vector $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_k)$ is a vector such that $\pi_\ell \in [0,1]$ for all $1 \leq \ell \leq k$ and $\pi_1 + \cdots + \pi_k = 1$.

If $P(z_j = \ell) = \pi_\ell$, for $1 \leq \ell \leq k$, and if

$$P(\mathbf{x}_j \mid z_j = \ell) = P(\mathbf{x}_j \mid \boldsymbol{\theta}_\ell) \quad \forall j, \ell,$$

then the probability of observing $\mathbf{x}_j$ is

$$P(\mathbf{x}_j) = \sum_{\ell=1}^{k} \pi_\ell P(\mathbf{x}_j \mid \boldsymbol{\theta}_\ell) = \sum_{\ell=1}^{k} P(z_j = \ell) P(\mathbf{x}_j \mid z_j = \ell),$$

according to the **Law of Total Probability**.

In this set-up, we interpret z_j as the **cluster label** for $\mathbf{x}_j$. Alternatively, we could use

$$\mathbf{z}_j \in \{0,1\}^k, \quad \|\mathbf{z}_j\|_2 = 1$$

to denote the **cluster signature** of $\mathbf{x}_j$. The norm condition implies that exactly one of the components of $\mathbf{z}_j$ is 1; all others are 0. For instance,

[56] We borrow extensively from Deng and Han's *Probabilistic Models for Clustering* chapter in [4].

if there are $k = 5$ mechanisms (clusters) in the data and $\mathbf{x}_j \in C_4$, then $\mathbf{z}_j = (0, 0, 0, 1, 0)$.[57]

If we write

$$P(\mathbf{z}_j) = \pi_1^{z_{j,1}} \times \cdots \times \pi_k^{z_{j,k}} = \prod_{\ell=1}^{k} \pi_\ell^{z_{j,\ell}}$$

and

$$P(\mathbf{x}_j \mid \mathbf{z}_j) = P(\mathbf{x}_j \mid \boldsymbol{\theta}_1)^{\,z_{j,1}} \times \cdots \times P(\mathbf{x}_j \mid \boldsymbol{\theta}_k)^{\,z_{j,k}} = \prod_{\ell=1}^{k} P(\mathbf{x}_j \mid \boldsymbol{\theta}_\ell)^{\,z_{j,\ell}},$$

we recover the **mixture model**:

$$P(\mathbf{x}_j) = \sum_{\ell=1}^{k} \pi_\ell P(\mathbf{x}_j \mid \boldsymbol{\theta}_\ell) = \sum_{\ell=1}^{k} P(\mathbf{z}_j \in C_\ell) P(\mathbf{x}_j \mid \mathbf{z}_j \in C_\ell).$$

Generative Process In practice, then, we can imagine that the dataset $\mathbf{X}$ is generated as follows. For $1 \leq j \leq n$:

1. draw a cluster signature $\mathbf{z}_j \sim \mathcal{G}_k(\boldsymbol{\pi}) = \text{Mult}_k(\boldsymbol{\pi})$, and
2. draw an observation $\mathbf{x}_j$ from the corresponding mechanism according to $P(\mathbf{x}_j \mid \mathbf{z}_j)$.

But we usually do not have access to this **generative process**; instead, we are given $\mathbf{X}$ and the clustering task is to determine how likely it is that component C_ℓ, $1 \leq \ell \leq k$, is **responsible** for observation $\mathbf{x}_j$, $1 \leq j \leq n$.

To do so, we need to compute the probabilities

$$\gamma(z_{j,\ell}) = P(\mathbf{z}_j \in C_\ell \mid \mathbf{x}_j), \quad \forall j, \ell.$$

This is difficult to do directly; we use **Bayes' Theorem** to provide an easier handle on the computations:

$$\begin{aligned}\gamma(z_{j,\ell}) = P(\mathbf{z}_j \in C_\ell \mid \mathbf{x}_j) &= \frac{P(\mathbf{z}_j \in C_\ell) P(\mathbf{x}_j \mid \mathbf{z}_j \in C_\ell)}{P(\mathbf{x}_j)} \\ &= \frac{P(\mathbf{z}_j \in C_\ell) P(\mathbf{x}_j \mid \mathbf{z}_j \in C_\ell)}{\sum_{\nu=1}^{k} P(\mathbf{z}_j \in C_\nu) P(\mathbf{x}_j \mid \mathbf{z}_j \in C_\nu)} = \frac{\pi_\ell P(\mathbf{x}_j \mid \boldsymbol{\theta}_\ell)}{\sum_{\nu=1}^{k} \pi_\nu P(\mathbf{x}_j \mid \boldsymbol{\theta}_\nu)}.\end{aligned}$$

The **clustering objective** is to infer $\{\pi_\ell\}_{\ell=1}^{k}$, $\{\boldsymbol{\theta}_\ell\}_{\ell=1}^{k}$ from $\mathbf{X}$ for a fixed k, to obtain the desired probabilities $\gamma(z_{j,\ell})$.

Denote

$$\boldsymbol{\Theta} = \{\pi_1, \ldots, \pi_k, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_\ell\}.$$

If we further assume that the $\mathbf{x}_j$ are **independently** drawn by the generative process, then, by construction:

$$P(\mathbf{X} \mid \boldsymbol{\Theta}) = \prod_{j=1}^{n} \sum_{\ell=1}^{k} \pi_k P(\mathbf{x}_j \mid \boldsymbol{\theta}_\ell),$$

or

$$\text{LL}(\boldsymbol{\Theta}) = \ln P(\mathbf{X} \mid \boldsymbol{\Theta}) = \sum_{j=1}^{n} \ln \left(\sum_{\ell=1}^{k} \pi_k P(\mathbf{x}_j \mid \boldsymbol{\theta}_\ell) \right).$$

[57] This notation can be generalized to **fuzzy clusters**: the cluster signature of $\mathbf{x}_j$ is

$$\mathbf{z}_j \in [0,1]^k, \quad \|\mathbf{z}_j\|_2 = 1;$$

if $\mathbf{z}_j = (0, 0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0)$, say, then we would interpret $\mathbf{x}_j$ as belonging equally to clusters C_3 and C_4 or as having probability $1/2$ of belonging to either C_3 or C_4.

The **maximum likelihood estimator** (MLE) of $\mathbf{\Theta}$ is

$$\mathbf{\Theta}_{\text{MLE}} = \arg\max_{\mathbf{\Theta}} \left\{ \ln P(\mathbf{X} \mid \mathbf{\Theta}) \right\};$$

if we have information about the **prior** $P(\mathbf{\Theta})$, then we may use the **maximum a posteriori estimator** (MAP) instead:

$$\mathbf{\Theta}_{\text{MAP}} = \arg\max_{\mathbf{\Theta}} \left\{ \ln P(\mathbf{X} \mid \mathbf{\Theta}) + \ln P(\mathbf{\Theta}) \right\}.$$

Whether we use MLE or MAP depend, in large part, on the form taken by the component distributions.

Gaussian Mixture Models A standard assumption is that all clusters are generated by Gaussian mechanisms, which is to say that $P(\mathbf{x}_j \mid \boldsymbol{\theta}_\ell)$ arises from a **multivariate Gaussian** distribution (GMM):

$$\mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_\ell, \mathbf{\Sigma}_\ell) = \frac{1}{\sqrt{(2\pi)^p |\mathbf{\Sigma}_\ell|}} \exp\left(-\tfrac{1}{2}(\mathbf{x}_j - \boldsymbol{\mu}_\ell)^\top \mathbf{\Sigma}_\ell^{-\mathbf{1}}(\mathbf{x}_j - \boldsymbol{\mu}_\ell)\right),$$

where $\boldsymbol{\mu}_\ell \in \mathbb{R}^p$ and $\mathbf{\Sigma}_\ell$ is a symmetric positive semi-definite matrix. Thus, if there are k components, then

$$P(\mathbf{x}_j \mid \mathbf{\Theta}) = \sum_{\ell=1}^{k} \pi_\ell \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_\ell, \mathbf{\Sigma}_\ell)$$

and

$$\text{LL}(\mathbf{\Theta}) = \ln P(\mathbf{X} \mid \mathbf{\Theta}) = \sum_{j=1}^{n} \ln \left(\sum_{\ell=1}^{k} \pi_\ell \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_\ell, \mathbf{\Sigma}_\ell) \right).$$

It is straightforward to show that

$$\nabla \text{LL}_{\boldsymbol{\mu}_\ell}(\mathbf{\Theta}) = \mathbf{\Sigma}_\ell^{-1} \sum_{j=1}^{n} \gamma(z_{j,\ell})(\mathbf{x}_j - \boldsymbol{\mu}_\ell),$$

so that the **MLE estimators** for the **mean vectors** are

$$\hat{\boldsymbol{\mu}}_\ell = \frac{\displaystyle\sum_{j=1}^{n} \gamma(z_{j,\ell})\, \mathbf{x}_j}{\displaystyle\sum_{j=1}^{n} \gamma(z_{j,\ell})}.$$

Thus $\hat{\boldsymbol{\mu}}_\ell$ is a weighted mean of the observations of $\mathbf{X}$, with weights corresponding to the posterior probability $\gamma(z_{j,\ell})$ that the $\ell-$th component was responsible for generating $\mathbf{x}_j$.

Simultaneously, we can show that

$$\nabla \text{LL}_{\mathbf{\Sigma}_\ell}(\mathbf{\Theta}) = \sum_{j=1}^{n} \frac{\pi_\ell}{P(\mathbf{x}_j \mid \mathbf{\Theta})} \cdot \frac{\partial \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_\ell, \mathbf{\Sigma}_\ell)}{\partial \mathbf{\Sigma}_\ell};$$

slightly more complicated manipulations show that the **MLE estimators** for the **covariance matrices** are also weighted averages:

$$\hat{\mathbf{\Sigma}}_\ell = \frac{\sum_{j=1}^{n} \gamma(z_{j,\ell})(\mathbf{x}_j - \hat{\boldsymbol{\mu}}_\ell)(\mathbf{x}_j - \hat{\boldsymbol{\mu}}_\ell)^\top}{\sum_{j=1}^{n} \gamma(z_{j,\ell})}.$$

Finally, to obtain the mixture probabilities π_ℓ, we must maximize $\text{LL}(\mathbf{\Theta})$ with respect to $\boldsymbol{\pi}$, subject to $\pi_\ell \in [0,1]$ and $\pi_1 + \cdots + \pi_k = 1$; we can use **Lagrange multipliers** to show that the MLE estimates of the mixture probabilities are also an average:

$$\hat{\pi}_\ell = \frac{1}{n}\sum_{j=1}^{n} \gamma(z_{j,\ell}).$$

So we have nice expressions for the MLE estimates $\hat{\mathbf{\Theta}}$.[58]

Expectation-Maximization Algorithm While there is no **closed-form solution** allowing us to express the cluster signatures directly in terms of the observed data $\mathbf{X}$, there is a simple iterative solution based on the **Expectation-Maximization** algorithm for GMM.

Input: $\mathbf{X}$
Output: $\mathbf{\Theta}^*$ which maximizes $\text{LL}(\mathbf{\Theta})$

0. Initialize $\mathbf{\Theta}^{[0]} = \left\{\boldsymbol{\mu}_\ell^{[0]}, \mathbf{\Sigma}_\ell^{[0]}, \pi_\ell^{[0]}\right\}_{\ell=1}^{k}$ and set

$$\text{LL}^{[0]} = \text{LL}(\mathbf{\Theta}^{[0]});$$

For $i = 0$ to `max_step`, do:

1. **E**(xpectation)**-step**: compute the responsibilities

$$\gamma(z_{j,\ell}^{[i]}) = \frac{\pi_\ell^{[i]} \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_\ell^{[i]}, \mathbf{\Sigma}_\ell^{[i]})}{\sum_{\nu=1}^{k} \pi_\nu^{[i]} \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_\nu^{[i]}, \mathbf{\Sigma}_\nu^{[i]})}, \quad \forall j, \ell;$$

2. **M**(aximization)**-step**: update the parameters

$$\boldsymbol{\mu}_\ell^{[i+1]} = \frac{\sum_{j=1}^{n} \gamma(z_{j,\ell}^{[i]})\, \mathbf{x}_j}{\sum_{j=1}^{n} \gamma(z_{j,\ell}^{[i]})}, \quad \forall \ell;$$

$$\mathbf{\Sigma}_\ell^{[i+1]} = \frac{\sum_{j=1}^{n} \gamma(z_{j,\ell}^{[i]})(\mathbf{x}_j - \boldsymbol{\mu}_\ell^{[i]})(\mathbf{x}_j - \boldsymbol{\mu}_\ell^{[i]})^\top}{\sum_{j=1}^{n} \gamma(z_{j,\ell}^{[i]})}, \quad \forall \ell,$$

$$\pi_\ell^{[i+1]} = \frac{1}{n}\sum_{j=1}^{n} \gamma(z_{j,\ell}^{[i]}), \quad \forall \ell;$$

[58] There is a problem, however: we need the clustering probabilities $\gamma(z_{j,\ell})$ in order to provide the MLE estimates ... but the former depend on the MLE estimates!

3. Set $\text{LL}^{[i+1]} = \text{LL}(\mathbf{\Theta}^{[i]})$ and check for convergence according to some **convergence criterion**

$$(\|\mathbf{\Theta}^{[i]} - \mathbf{\Theta}^{[i+1]}\| < \varepsilon, \quad \text{say}) :$$

if satisfied, set $\mathbf{\Theta}^* = \mathbf{\Theta}^{[i+1]}$; otherwise, set $i := i+1$ and repeat steps 1 to 3.

There are two main limitations to using EM for GMM:

- EM is **costlier** (has a longer run-time) than $k-$means, and depending on the initialization, the algorithm may converge to a **local critical point** which is not necessarily the global maximizer;
- as the algorithm iterates, two (or more) GMM clusters can **collapse** into a single GMM cluster.

Note that the EM algorithm can be sped-up by **first running** k**−means** and using the mean vector, covariance matrix, and proportion of observations in the $k-$means cluster C_ℓ for the initialization of $\boldsymbol{\mu}_\ell^{[0]}$, $\mathbf{\Sigma}_\ell^{[0]}$, and π_ℓ for $1 \leq \ell \leq k$.

The collapsing of clusters can be mitigated by monitoring $\|\mathbf{\Sigma}_\ell^i\|_2$ and randomly resetting $\boldsymbol{\mu}_\ell^{[i]}$, $\mathbf{\Sigma}_\ell^{[i]}$ when some threshold is reached.

Special Cases and Variants In a GMM with k components, if $\mathbf{\Sigma}_\ell = \mathbf{\Sigma} = \sigma^2\mathbf{I}_n$ for all ℓ, then

$$P(\mathbf{x}_j \mid \boldsymbol{\mu}_\ell, \mathbf{\Sigma}) = \frac{1}{\sqrt{(2\pi)^p}\sigma} \cdot \exp\left(-\frac{1}{2\sigma^2}\|(\mathbf{x} - \boldsymbol{\mu}_\ell)\|_2^2\right);$$

the EM algorithm applied to this special case leads to

$$\textbf{E-step:} \quad \gamma(z_{j,\ell}^{[i]}) = \frac{\pi_\ell^{[i]} \exp\left(-\|\mathbf{x}_j - \boldsymbol{\mu}_\ell^{[i]}\|_2^2/2\sigma^2\right)}{\sum_{\nu=1}^k \pi_\nu^{[i]} \exp\left(-\|\mathbf{x}_j - \boldsymbol{\mu}_\nu^{[i]}\|_2^2/2\sigma^2\right)}$$

$$\textbf{M-step:} \quad \boldsymbol{\mu}_\ell^{[i+1]} = \frac{\sum_{j=1}^n \gamma(z_{j,\ell}^{[i]})\,\mathbf{x}_j}{\sum_{j=1}^n \gamma(z_{j,\ell}^{[i]})}$$

$$\pi_\ell^{[i+1]} = \frac{1}{n}\sum_{j=1}^n \gamma(z_{j,\ell}^{[i]}).$$

When $\sigma \longrightarrow 0$, we can show that

$$\gamma(z_{j,\ell}) \longrightarrow \begin{cases} 1 & \text{if } \ell = \arg\min_\nu \left\{\|\mathbf{x}_j - \boldsymbol{\mu}_\nu\|_2^2\right\} \\ 0 & \text{otherwise} \end{cases}$$

which is simply the formulation for $k-$means. Note that the components do not need to be multivariate Gaussians; there is a **general EM algorithm** that takes advantage of the concavity of the ln function [4].

If the dataset of observations is **binary**, as may occur in image datasets (each pixel taking on the values 0 or 1, depending as to whether the pixel is white or black, say), we can modify GMM so that $P(\mathbf{x}_j \mid \boldsymbol{\mu}_\ell)$ arises from a **multivariate Bernoulli** distribution:

$$\mathcal{B}(\mathbf{x}_j \mid \boldsymbol{\mu}_\ell) = \prod_{\nu=1}^{p} \mu_{\ell,\nu}^{x_{j,\nu}} (1 - \mu_{\ell,\nu})^{1-x_{j,i}},$$

where $\boldsymbol{\mu}_\ell \in [0,1]^p$. Thus, if there are k components, then

$$P(\mathbf{x}_j \mid \boldsymbol{\Theta}) = \sum_{\ell=1}^{k} \pi_\ell \mathcal{B}(\mathbf{x}_j \mid \boldsymbol{\mu}_\ell)$$

and

$$\text{LL}(\boldsymbol{\Theta}) = \ln P(\mathbf{X} \mid \boldsymbol{\Theta}) = \sum_{j=1}^{n} \ln \left(\sum_{\ell=1}^{k} \pi_\ell \prod_{\nu=1}^{p} \mu_{\ell,\nu}^{x_{j,\nu}} (1 - \mu_{\ell,\nu})^{1-x_{j,i}} \right).$$

We can find $\boldsymbol{\Theta}^*$ that maximizes $\text{LL}(\boldsymbol{\Theta})$ by using the EM algorithm for the Bernoulli Mixture Model (BMM): the EM algorithm applied to this special case leads to

$$\textbf{E-step:} \quad \gamma(z_{j,\ell}^{[i]}) = \pi_\ell^{[i]} \prod_{\nu=1}^{p} \left(\mu_{\ell,\nu}^{[i]} \right)^{x_{j,\nu}} (1 - \mu_{\ell,\nu}^{[i]})^{1-x_{j,i}}$$

$$\textbf{M-step:} \quad \boldsymbol{\mu}_\ell^{[i+1]} = \frac{\sum_{j=1}^{n} \gamma(z_{j,\ell}^{[i]})\, \mathbf{x}_j}{\sum_{j=1}^{n} \gamma(z_{j,\ell}^{[i]})}$$

$$\pi_\ell^{[i+1]} = \frac{1}{n} \sum_{j=1}^{n} \gamma(z_{j,\ell}^{[i]}),$$

with initialization $\pi_\ell^{[0]} = \frac{1}{k}$ and

$$\boldsymbol{\mu}_\ell^{[0]} \sim \prod_{\nu=1}^{p} \mathcal{U}(0.25, 0.75)$$

for $1 \leq \ell \leq k$.

Other variants include **Generalized EM**, **Variational EM**, and **Stochastic EM** [4]. Note that the essence of EM methods remains the same for all algorithms: we attempt to "guess" the value of the "hidden" cluster variable $z_{j,\ell}$ in the **E-step**, and we update the model parameters in the **M-step**, based on the approximated responsibilities found in the E−step.

Interestingly, EM can detect overlapping clusters (unlike k−means). But most variants share the same limitations: convergence to a global maximizer is not guaranteed; it may be quite slow even when it does converge, and the correct number of components is assumed to be known prior to analysis.

Example: Gapminder Dataset We cluster the 2011 Gapminder dataset using the `mclust` implementation of EM in `R`; no parameters need be specified (unless we want to use a different dissimilarity measure).[59]

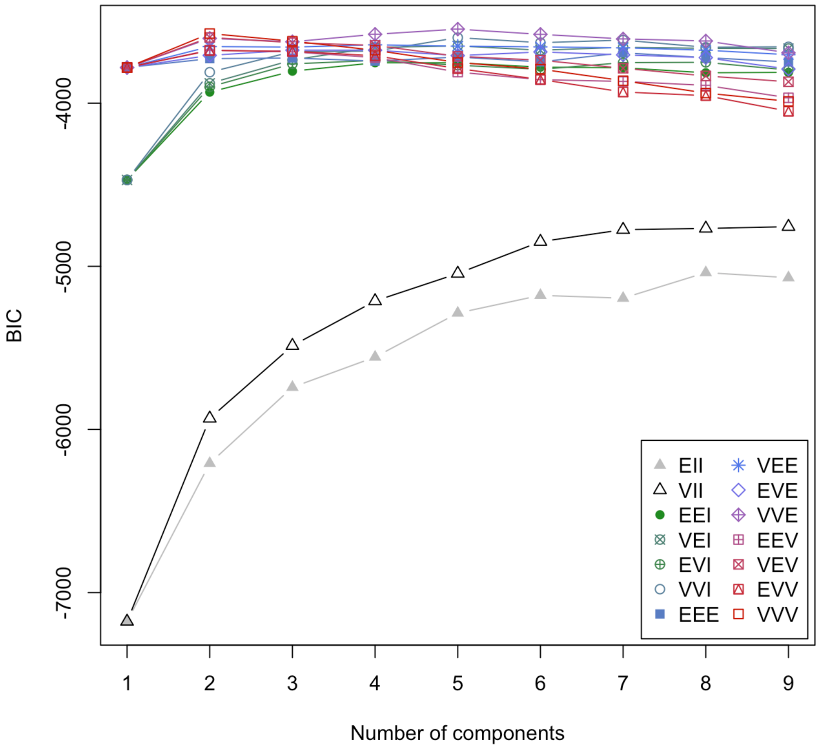
[59] The `mclust` vignette contains more information.

This implementation determines the optimal number of clusters using BIC (see Section 20.4.3).

We cluster both the **raw** data and the **scaled** data, to showcase the impact scaling can have.

We start by determining the number of sources in the raw data:

```
library(mclust)
set.seed(0)
BIC <- mclustBIC(gapminder.SoCL.2011[,c(3:7)])
plot(BIC)
```

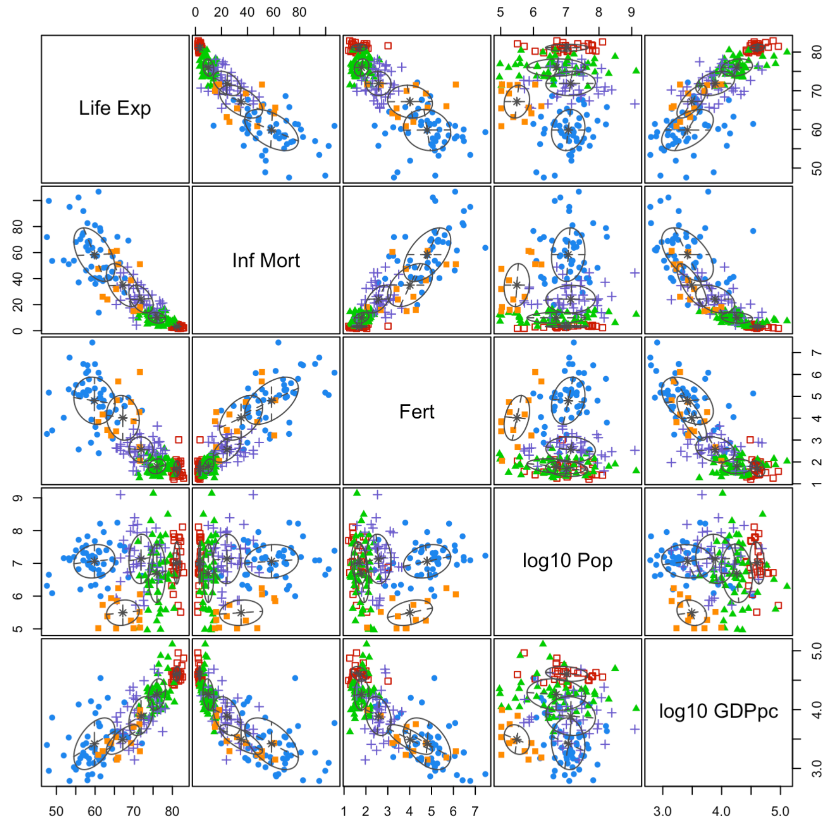


```
summary(BIC)
```

```
Best BIC values:
             VVE,5        VVV,2        VVE,4
BIC      -3545.915 -3573.00451 -3577.50705
BIC diff     0.000   -27.08943   -31.59198
```

This suggests that there are 5 clusters, which we display on the next page.

```
mod1 <- Mclust(gapminder.SoCL.2011[,c(3:7)], x = BIC)
plot(mod1, what = "classification")
```



The MLE estimators for the mean vectors and the covariance matrices for each cluster are computed as below.

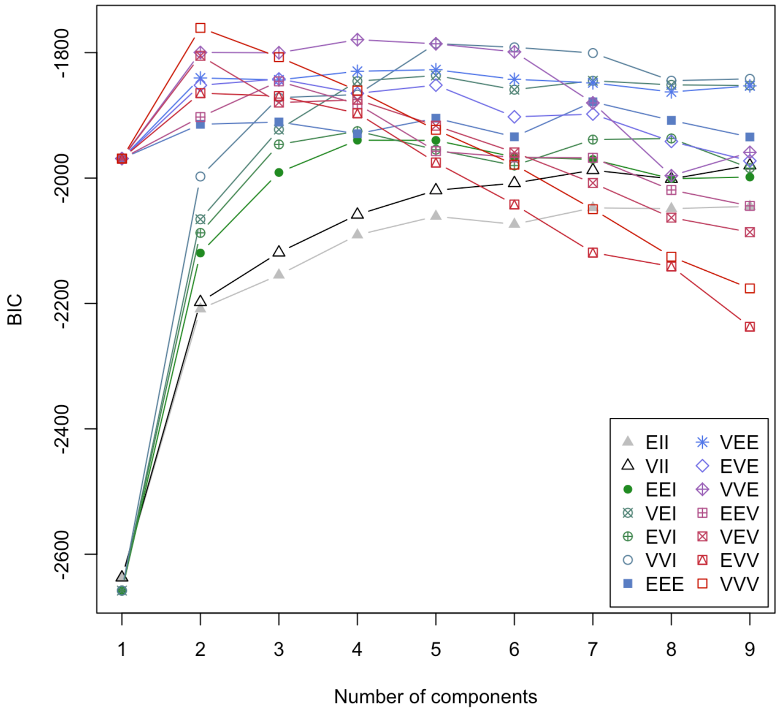
```
summary(mod1, parameters = TRUE)
```

```
Clustering table:
 1  2  3  4  5 
51 27 52 41 13 

Mixing probabilities:
         1          2          3          4          5 
0.28320179 0.14590868 0.27317375 0.22804391 0.06967186 

Means:
                 [,1]      [,2]      [,3]      [,4]      [,5]
Life Exp    59.834937 81.134281 75.998817 71.888971 67.179530
Inf Mort    58.426776  3.254371  9.913182 24.238438 35.027409
Fert         4.798189  1.691375  1.800049  2.566219  4.014588
log10 Pop    7.061637  7.016423  6.695346  7.133957  5.492939
log10 GDPpc  3.419639  4.606936  4.257569  3.889928  3.491324

Variances:
[,,1]
               Life Exp   Inf Mort       Fert   log10 Pop log10 GDPpc
Life Exp     28.3344610 -55.700455 -0.7444078  0.38472919  1.15550574
Inf Mort    -55.7004554 423.909389 11.8241909  1.11117987 -4.78143371
Fert         -0.7444078  11.824191  1.1663137  0.13060543 -0.21270286
log10 Pop     0.3847292   1.111180  0.1306054  0.26127270 -0.01805256
log10 GDPpc   1.1555057  -4.781434 -0.2127029 -0.01805256  0.19296195
```

```
[,,2]
              Life Exp     Inf Mort        Fert    log10 Pop  log10 GDPpc
Life Exp    0.77045283  0.012104972  0.02774975  0.011407709  0.019766044
Inf Mort    0.01210497  0.679738470  0.01905169  0.003200837 -0.004476642
Fert        0.02774975  0.019051687  0.12771916 -0.042927224 -0.012328032
log10 Pop   0.01140771  0.003200837 -0.04292722  0.416389572 -0.018157675
log10 GDPpc 0.01976604 -0.004476642 -0.01232803 -0.018157675  0.015998859
[,,3]
              Life Exp    Inf Mort         Fert   log10 Pop  log10 GDPpc
Life Exp     4.9940940 -1.72216863  0.142340294  0.10257591  0.137637940
Inf Mort    -1.7221686 17.19416942  0.499861409  0.05750027 -0.180062410
Fert         0.1423403  0.49986141  0.115694181 -0.09747328  0.001937248
log10 Pop    0.1025759  0.05750027 -0.097473276  0.79668779 -0.027754402
log10 GDPpc  0.1376379 -0.18006241  0.001937248 -0.02775440  0.071168246
[,,4]
                Life Exp    Inf Mort        Fert   log10 Pop log10 GDPpc
Life Exp      9.91288695 -14.4994297 -0.09127317  0.15574758  0.36288775
Inf Mort    -14.49942972 112.8663448  3.16128196  0.30356930 -1.26453815
Fert         -0.09127317   3.1612820  0.33684903 -0.03319199 -0.04501104
log10 Pop     0.15574758   0.3035693 -0.03319199  0.57090101 -0.02068610
log10 GDPpc   0.36288775  -1.2645382 -0.04501104 -0.02068610  0.10900700
[,,5]
               Life Exp    Inf Mort       Fert   log10 Pop log10 GDPpc
Life Exp     18.0275970 -35.9634327 -0.5031561  0.24042845  0.74329472
Inf Mort    -35.9634327 273.4386404  7.6150659  0.71441466 -3.08374929
Fert         -0.5031561   7.6150659  1.0499241  0.12825306 -0.19073273
log10 Pop     0.2404284   0.7144147  0.1282531  0.15122047 -0.02103882
log10 GDPpc   0.7432947  -3.0837493 -0.1907327 -0.02103882  0.06307893
```

Let us repeat the procedure on the **scaled** dataset.

```
BIC.s <- mclustBIC(scale(gapminder.SoCL.2011[,c(3:7)]))
plot(BIC.s)
```

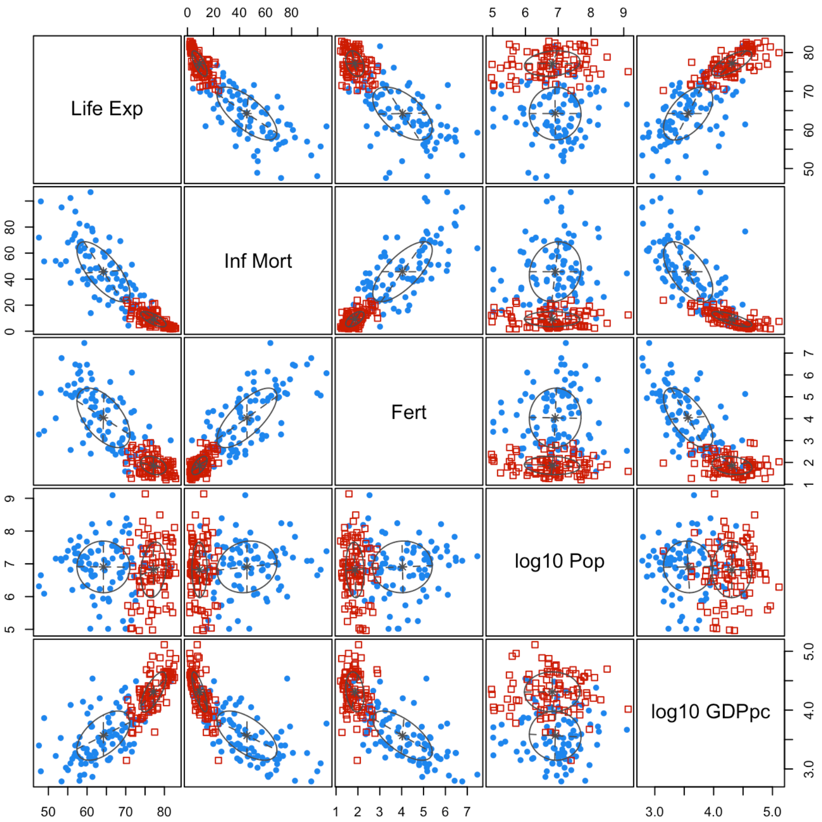


```
summary(BIC.s)
```

```
Best BIC values:
             VVV,2       VVE,4       VVI,5
BIC      -1760.55 -1779.31339 -1785.64061
BIC diff     0.00   -18.76374   -25.09096
```

This suggests that there are 2 clusters, as seen below.

```
mod2 <- Mclust(gapminder.SoCL.2011[,c(3:7)], x = BIC.s)
plot(mod2, what = "classification")
```



The MLE estimators for the mean vectors and covariance matrices for each cluster are computed as below.

```
summary(mod2, parameters = TRUE)
```

```
Clustering table:
 1  2
91 93

Mixing probabilities:
        1         2
0.5059039 0.4940961
```

```
Means:
                 [,1]      [,2]
Life Exp    64.226346 77.164028
Inf Mort    45.696249  9.268183
Fert         4.038171  1.860683
log10 Pop    6.906989  6.816294
log10 GDPpc  3.566453  4.310365

Variances:
[,,1]
                 Life Exp    Inf Mort        Fert   log10 Pop log10 GDPpc
Life Exp      46.94600319 -112.051411 -5.17027793 -0.06217152  1.55664144
Inf Mort    -112.05141081  531.145010 22.59825954  2.37700481 -5.67438362
Fert          -5.17027793   22.598260  1.84974433  0.03491542 -0.37634844
log10 Pop     -0.06217152    2.377005  0.03491542  0.62342443 -0.01356435
log10 GDPpc    1.55664144   -5.674384 -0.37634844 -0.01356435  0.17508305
[,,2]
               Life Exp     Inf Mort        Fert    log10 Pop  log10 GDPpc
Life Exp     10.7636945 -13.79843667 -0.40969404  0.508375039  0.789463554
Inf Mort    -13.7984367  34.33063934  1.47575086 -0.094412049 -1.497455669
Fert         -0.4096940   1.47575086  0.17525844 -0.037372594 -0.032046730
log10 Pop     0.5083750  -0.09441205 -0.03737259  0.719418611  0.002308201
log10 GDPpc   0.7894636  -1.49745567 -0.03204673  0.002308201  0.115062583
```

22.4.4 Affinity Propagation

Affinity propagation (AP) is a fairly recent arrival on the clustering stage [278, 279]; it takes a somewhat novel perspective on clustering although, as might be expected, there are still similarities to other clustering methods, in particular, DBSCAN and $k-$means.

AP takes the $k-$**medoids** algorithm as a jumping off point. Unlike $k-$means or EM, this algorithm does not operate on statistical principles; rather, it selects existing observations to act as the **exemplar** for a particular cluster (rather than a mean vector, as in $k-$means; see Figure 22.23 for an illustration).

The $k-$mediods algorithm refines the selection of these exemplars so that in the final (stable) configuration, the observations assigned to an exemplar are quite similar to it, relative to other exemplars. As the name suggests, the number of clusters k must be selected prior to running the algorithm; as is the case with $k-$means, $k-$medoids is **non-deterministic** and is sensitive to the **initial choice** of exemplars and similarity metric.

The AP algorithm attempts to overcome the issues arising with $k-$medoids, using Bayesian network theory (in particular, belief propagation networks and factor graphs), and treats observations as a **connected graph**. In this approach, each graph vertex can:

- **communicate** with any other vertex, and
- act as a **possible** exemplar for other observations.

The selection of exemplars is determined by exchanging real-valued **messages** between points. Eventually, sets of exemplars and data points associated with each exemplar are generated from this iterative process, forming clusters. Messages are updated on the basis of fairly simple formulae. As in all clustering contexts, a similarity measure s must first

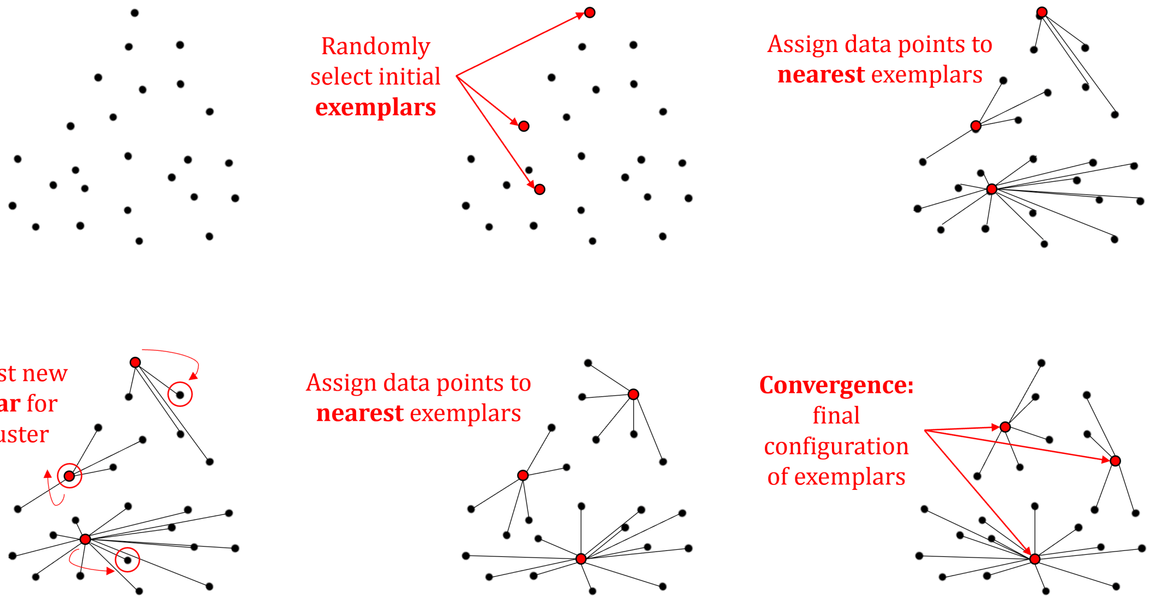


Figure 22.23: Illustration of 3−mediods on an artificial dataset (modified from [278]).

be selected prior to clustering: for distinct pairs (i, k), $s(i, k)$ represents initially the **suitability of k as an exemplar of i** (this suitability will be updated as "messages" are passed between observations).

Each observation k is further assigned a **preference** $s(k, k)$ that it be chosen as an exemplar. The preference can be constant, to indicate no particular initial preference.

Two types of messages get sent:

- the **availability** $a(i, k)$ sent from k to i, which reports on the suitability of k to be an exemplar of i;
- the **responsibility** $r(i, k)$ sent from i to k, which reports on the suitability of i to be represented by k.

The availabilities are initialized to $a(i, k) \leftarrow 0$, the responsibilities to

$$r(i, k) \leftarrow s(i, k) - \max_{k' \neq k}\{a(i, k') + s(i, k')\}.$$

This calculation allows eligible exemplars of an observation to "**compete**" for each observations, in a sense, so they can become that observation's exemplar.[60] After the initial assignment, an availability $a(i, k) = 0$ means that observation i has no affinity for k as its exemplar).

Subsequently, the focus switches back and forth between the exemplar and the observation perspective, with observations looking for available exemplars:

$$a(i,k) \leftarrow \begin{cases} \min\left\{0, r(k,k) + \sum_{i' \notin \{i,k\}} \max\{0, r(i,k)\}\right\} & i \neq k \\ \sum_{i' \neq k} \max\{0, r(i',k)\} & i = k \end{cases}$$

The case $i = k$ is intended to reflect **current evidence** that observation k is an exemplar. The responsibilities and availabilities are updated,

[60] As candidate exemplars are themselves observations, we can also compute **self-responsibility**: $r(k, k) \leftarrow s(k, k) - \max_{k \neq k'}\{s(k, k')\}$.

reflecting the **current affinity** that one observation has for choosing another observation as its exemplar (hence the name), until the quantities converge to $r(i,k)$ and $a(i,k)$, respectively, for all pairs of observations (i,k).

This leads to the **cluster assignment** $\{c_1,\ldots,c_n\}$, where

$$c_i = \arg\max_k \{a(i,k)+r(i,k)\}, \quad 1 \leq i \leq n;$$

if i is an observation with associated exemplar k, then $c_i = c_k = k$.

The fact that any observation can become an exemplar when the quantities are updated, and thus that the number of clusters is not an algorithm parameter, is an important distinction between AP and $k-$medoids (and other segmentation clustering approaches). The process is illustrated below.

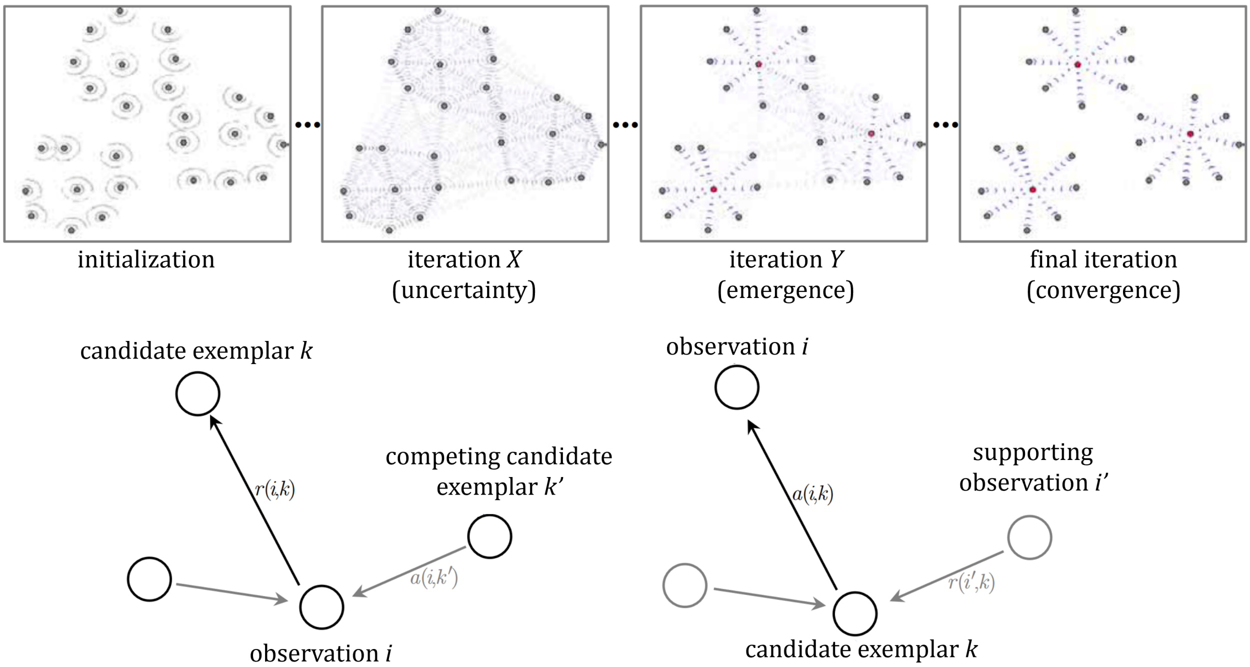

initialization
iteration X (uncertainty)
iteration Y (emergence)
final iteration (convergence)
candidate exemplar k
$r(i,k)$
competing candidate exemplar k'
$a(i,k')$
observation i
observation i
$a(i,k)$
supporting observation i'
$r(i',k)$
candidate exemplar k

Figure 22.24: Illustration of affinity propagation on an artificial dataset (top); illustration of availability and responsibility (bottom); modified from [278].

Setting Algorithm Parameters Two parameters impact AP's clustering behaviour: the **input preference** (which influences the eventual number of clusters) and the **dampening parameter**.

The input preference determines the suitability of each observation to act as an exemplar; this is often set as the **median similarity** in the data, but it can be tweaked. In principle, certain observations could be assigned preference values in a different manner, perhaps relating to domain knowledge (or previous results).

The **dampening parameter** is slightly more technical. Because affinity propagation creates a directed graph to generate clusters, it can become vulnerable to **graph loops**, which could result in algorithmic oscillations (the algorithm may not converge to a particular solution). The dampening factor acts to control this oscillation problem.

Comparison with Other Algorithms Performance of clustering algorithms can be considered both in general (e.g., based on best/worst cases of an implemented algorithm) or in the context of applications in particular domains – one major **drawback** of AP is the calculation cost of the similarity matrix, which is $O(n^2)$.

Once the similarity matrix has been calculated, the number of scalar computations scales linearly in the number of similarities or quadratically in the number of observations if all possible pairwise similarities are used [278]. In other words, AP is slow on larger datasets.

Arguably, one of the key **advantages** of AP (other than not having to specify the number of clusters up front) is its ability to use any similarity measure. As a result, we do not need to alter the dataset to 'fit' with a distance/similarity framework.[61]

Example We once again re-visit the 2011 (scaled) Gapminder dataset. We use the AP implementation found in the R package apcluster, with similarity $s(i,k) = -\|\mathbf{x}_i - \mathbf{x}_k\|^2$. We start by setting the input preference as the median similarity and obtain 14 clusters.

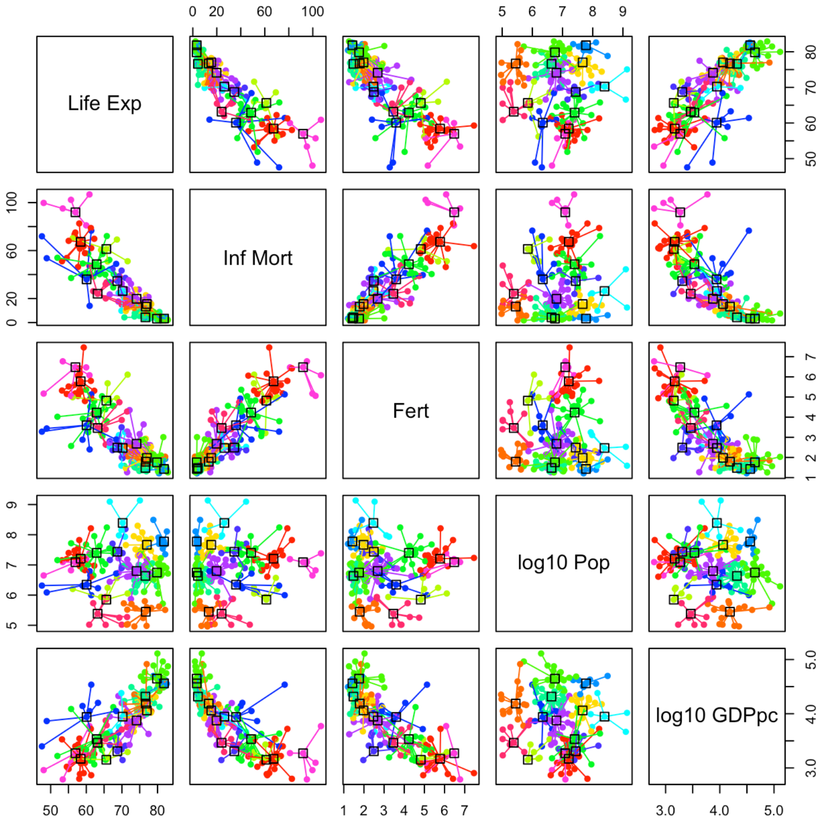
```
library(apcluster)
ap.gap.1 <- apcluster(negDistMat(r=2),
    scale(gapminder.SoCL.2011[,c(3:7)]))
ap.gap.1
```

```
Number of clusters    =  14

Exemplars:
   bfa brb col com dnk gha hrv idn ita nam npl pry tcd vut
Clusters:
   Cluster 1, exemplar bfa:
      afg bdi ben bfa civ cmr gin lbr moz mwi ner nga ssd tgo uga zmb cod
   Cluster 2, exemplar brb:
      bhs blz brb cpv isl mdv mlt sur brn lca mne vct atg grd syc
   Cluster 3, exemplar col:
      arg bra chl col dza irn lka mar mex mys per rou tha tur ukr ven vnm
   Cluster 4, exemplar com:
      com dji gmb gnb mrt tls
   Cluster 5, exemplar dnk:
      aut bel che dnk fin grc irl isr lux nld nor nzl omn prt qat sgp swe are
      bhr cyp kwt sau
   Cluster 6, exemplar gha:
      eri eth gha hti ken lao mdg pak png rwa sdn sen tza yem zaf zwe irq
   Cluster 7, exemplar hrv:
      bgr blr cri cub cze est hrv hun lbn ltu lva mus srb svk svn tto ury alb
      mkd bih
   Cluster 8, exemplar idn:
      chn egy idn ind phl rus
   Cluster 9, exemplar ita:
      aus can deu esp fra gbr ita jpn kor pol usa
   Cluster 10, exemplar nam:
      bwa cog gab gnq lso nam swz lby tkm
   Cluster 11, exemplar npl:
      bgd khm mmr npl tjk uzb prk
```

[61] Such as by changing categorical variables into numeric variables in some way, or ignoring categorical variables altogether.

```
   Cluster 12, exemplar pry:
      arm aze bol btn dom ecu geo gtm hnd jam kaz kgz mda mng nic pan pry slv
      tun jor pse syr
   Cluster 13, exemplar tcd:
      ago caf mli sle som tcd
   Cluster 14, exemplar vut:
      fji guy stp slb ton vut wsm kir fsm
```

```
plot(ap.gap.1, gapminder.SoCL.2011[,c(3:7)])
```



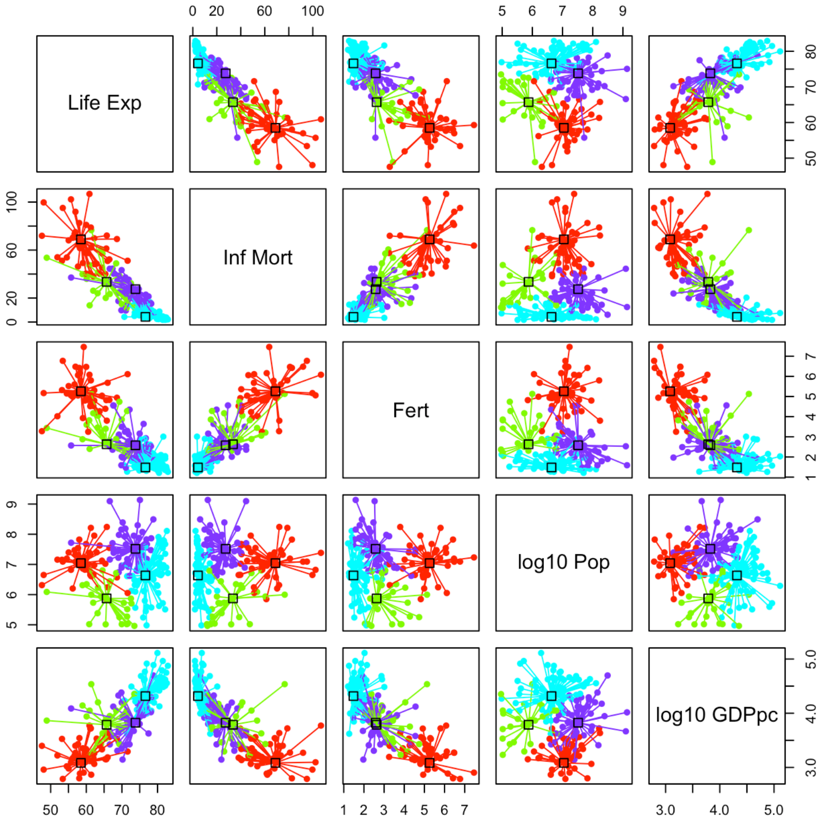
If instead we use the minimum similarity, we obtain 4 clusters (**exemplars:** Guinea, Guyana, Croatia, Morocco).

```
ap.gap.2 <- apcluster(negDistMat(r=2),
    scale(gapminder.SoCL.2011[,c(3:7)]), q=0)
ap.gap.2
```

```
Number of clusters    =  4

Exemplars:
   gin guy hrv mar
Clusters:
   Cluster 1, exemplar gin:
      afg ago bdi ben bfa caf civ cmr cog com eri eth gha gin gmb gnb hti ken
      lbr lso mdg mli moz mrt mwi ner nga pak png rwa sdn sen sle som ssd tcd
      tgo tza uga zmb zwe cod tls
```

```
Cluster 2, exemplar guy:
   blz btn bwa cpv dji fji gab gnq guy lao mng nam stp sur swz lby slb tkm
   ton vct vut wsm grd kir syc fsm
Cluster 3, exemplar hrv:
   arm aus aut bel bgr bhs blr brb can che chl cri cub cze deu dnk esp est
   fin fra gbr geo grc hrv hun irl isl isr ita jam jpn kor lbn ltu lux lva
   mda mdv mlt mus mys nld nor nzl omn pan pol prt qat rou sgp srb svk svn
   swe tto tun ury alb are bhr brn cyp kwt lca mkd mne sau bih atg
Cluster 4, exemplar mar:
   arg aze bgd bol bra chn col dom dza ecu egy gtm hnd idn ind irn kaz kgz
   khm lka mar mex mmr nic npl per phl pry rus slv tha tjk tur ukr usa uzb
   ven vnm yem zaf irq jor pse syr prk
```

```
plot(ap.gap.2, gapminder.SoCL.2011[,c(3:7)])
```



Which of these two schemes seems to provide a better segmentation?

22.4.5 Fuzzy Clustering

Fuzzy clustering (FC) is also called “soft” clustering (in opposition to “hard” clustering). Rather than assigning each observation to a cluster, they are assigned a **cluster signature**, a set of values that indicate their relative membership in each of the clusters.

The signature vector is often interpreted as a **probability vector**: observation $\mathbf{x}_i$ belongs to cluster ℓ with probability $p_{i,\ell} \geq 0$, with

$$p_{i,1} + \cdots + p_{i,c} = 1, \quad \text{for all } 1 \leq i \leq n.$$

Fuzzy c−Means: the Typical Approach The most prevalent algorithm for carrying out FC is called fuzzy c−means (FCM). It is a variant of k−means with two modifications:

- the presence of a new parameter $m > 1$, called the **fuzzyfier**, which determines the degree of "fuzziness" of the clusters, and
- cluster membership is output as a **weight vector**, with weights in $[0, 1]$ adding to 1.

As in k−means, c observations are selected randomly as the initial cluster centroids, as are the membership weights of each observation. The membership weights of each observations, relative to the current centroid, are re-calculated based on how "close" the point is to the given centroid in comparison to the distance to all of the other centroids.[62]

Effectively, we look for clusters that minimize the objective function

$$\sum_{\ell=1}^{c} \sum_{\mathbf{x}_i \in C_\ell} u_{i,\ell}^{m} \text{variation}(\mathbf{x}_i, \boldsymbol{\mu}_\ell),$$

where the **degree** $u_{i,\ell}^{m}$ to which observation $\mathbf{x}_i$ belongs to cluster C_ℓ is

$$u_{i,\ell}^{m} = \frac{1}{\sum_{j=1}^{c} \left(\frac{\text{variation}(\mathbf{x}_i, \boldsymbol{\mu}_\ell)}{\text{variation}(\mathbf{x}_i, \boldsymbol{\mu}_j)} \right)^{2/(m-1)}}.$$

The value of m effectively determines the width of **fuzziness bands** around clusters, where clusters may overlap with other clusters. Within these bands, if there are overlaps, points will have weights between 0 and 1.

Outside of these bands, points will have a membership of 1 for a particular cluster (that it is close to) and a membership of 0 for other bands.

As with k−means, the algorithm is generally run until the change in membership values, or in this case the weights, falls below a particular threshold. In practice, we typically use $m = 2$ and

$$\text{variation}(\mathbf{x}_i, \boldsymbol{\mu}_\ell) = \|\mathbf{x}_i - \boldsymbol{\mu}_\ell\|^2.$$

As $m \longrightarrow 1$, FCM converges to k−means.

Comparison Between Fuzzy c−Means and k−Means To gain an appreciation for how FCM works, it can be useful to compare its results to those provided by k−means. Figure 22.25 shows the same dataset clustered by k−means (left) and fuzzy c−means (right) [280].

On the right, we can see observations that "belong" to the 2 clusters. FCM is useful in this context because it would seem almost arbitrary for some of the points to be assigned to one or the other cluster (which is what k−means does).

[62] The centroid of the ℓth cluster is the weighted average of ALL observations by the degree to which they belong to cluster ℓ.

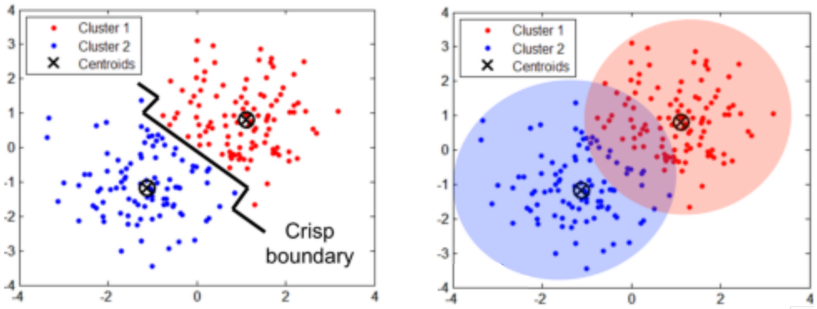


Figure 22.25: Fuzzy $c-$means vs. $k-$means clustering (author unknown).

Other Fuzzy Clustering Options Although FCM is the most popular fuzzy clustering algorithm, it is not a particularly **nuanced** algorithm. Like $k-$means, the resulting clusters are essentially blob-shaped. Sophisticated results can be gained by using more complex algorithms.

The **Gustafson–Kessel** (GK) clustering algorithm [281] is an early extension of FCM which replaces the simple distance measure used in FCM with a (covariance) matrix. This brings FCM more in-line with EM clustering, which also provides fuzzy results, and can be carried out with a variety of statistical models, resulting in a more mature clustering results, albeit at the cost of heavier processing. **FANNY** [282] is another fuzzy approach; it is less sensitive to outliers than FCM is.

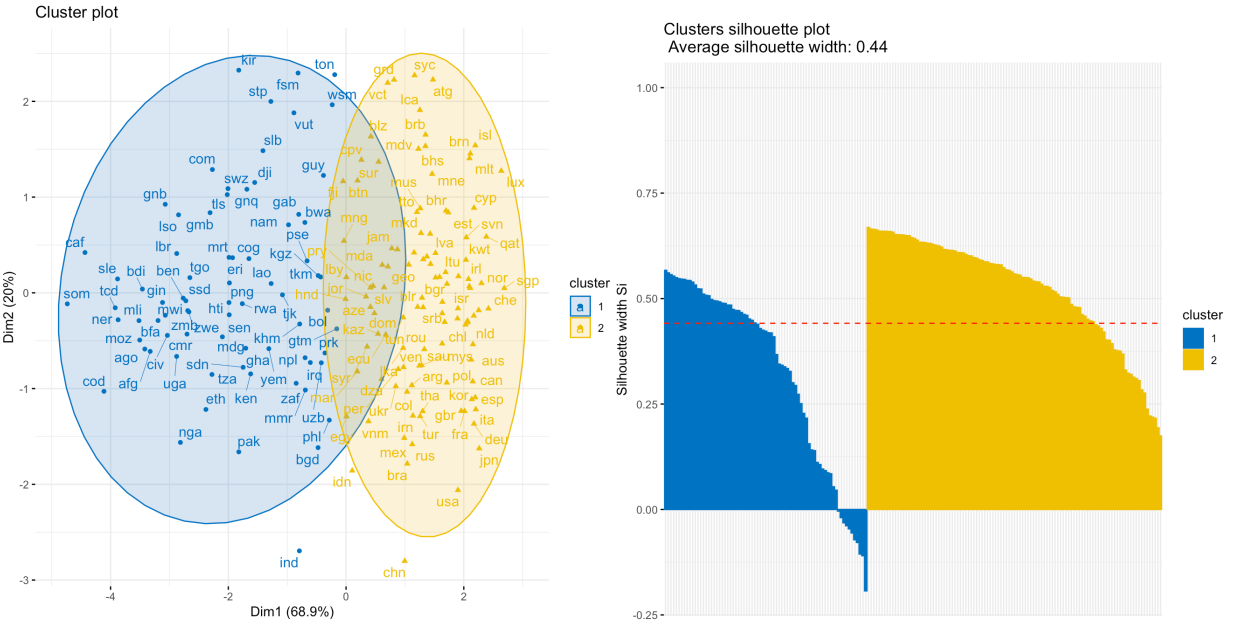
Fuzzy Clustering Validation As with hard clustering, it is important to **validate** fuzzy clusters. A number of validation strategies have been developed; the **Xie-Beni index** is a popular choice. It can be calculated for non-fuzzy clusters as well as for fuzzy clusters. However, it takes into accounts the weights of the points for each clustering by weighting the clustering separation and compactness measures using the membership matrix (i.e., the matrix that contains the weights for each observation with respect to each cluster). Other metrics include the **Tang index** and the **Kwon index** [283–285].

Example We show some results of FANNY (with $c = 2, 3, 4,$ and 6 clusters, implemented in `cluster`'s `fanny()`) and FCM (with $c = 4$ clusters, implemented in `e1071`'s `cmeans()`) on the (scaled) 2011 Gapminder dataset (again, using Euclidean dissimilarity).

We start with FANNY(2).

```
set.seed(987) # for replicability
fuzzy.gap <- cluster::fanny(scale(gapminder.SoCL.2011[,c(3:7)]),
                            k=2, metric="euclidean", maxit=20000)
attributes(fuzzy.gap)

factoextra::fviz_cluster(fuzzy.gap, ellipse.type = "norm", repel = TRUE,
             palette = "jco", ggtheme = theme_minimal(),
             legend = "right")

factoextra::fviz_silhouette(fuzzy.gap, palette = "jco",
                ggtheme = theme_minimal())
```

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

$class
[1] "fanny"     "partition"

  cluster size ave.sil.width
1       1   75          0.32
2       2  109          0.52
```



The plots for FANNY(3), FANNY(4), and FANNY(5), are displayed in Figure 22.26.[63]

```
FANNY(3)
  cluster size ave.sil.width
1       1   64          0.26
2       2   74          0.34
3       3   46          0.15

FANNY(4)
  cluster size ave.sil.width
1       1   53          0.33
2       2   80         -0.01
3       3   50         -0.17
4       4    1          0.00

FANNY(5)
  cluster size ave.sil.width
1       1   30          0.35
2       2   83          0.06
3       3   34         -0.21
4       4   34          0.03
5       5    3          0.60
```

[63] We simply replace k=2 by k=3, k=4, and k=6 in the call to fanny().

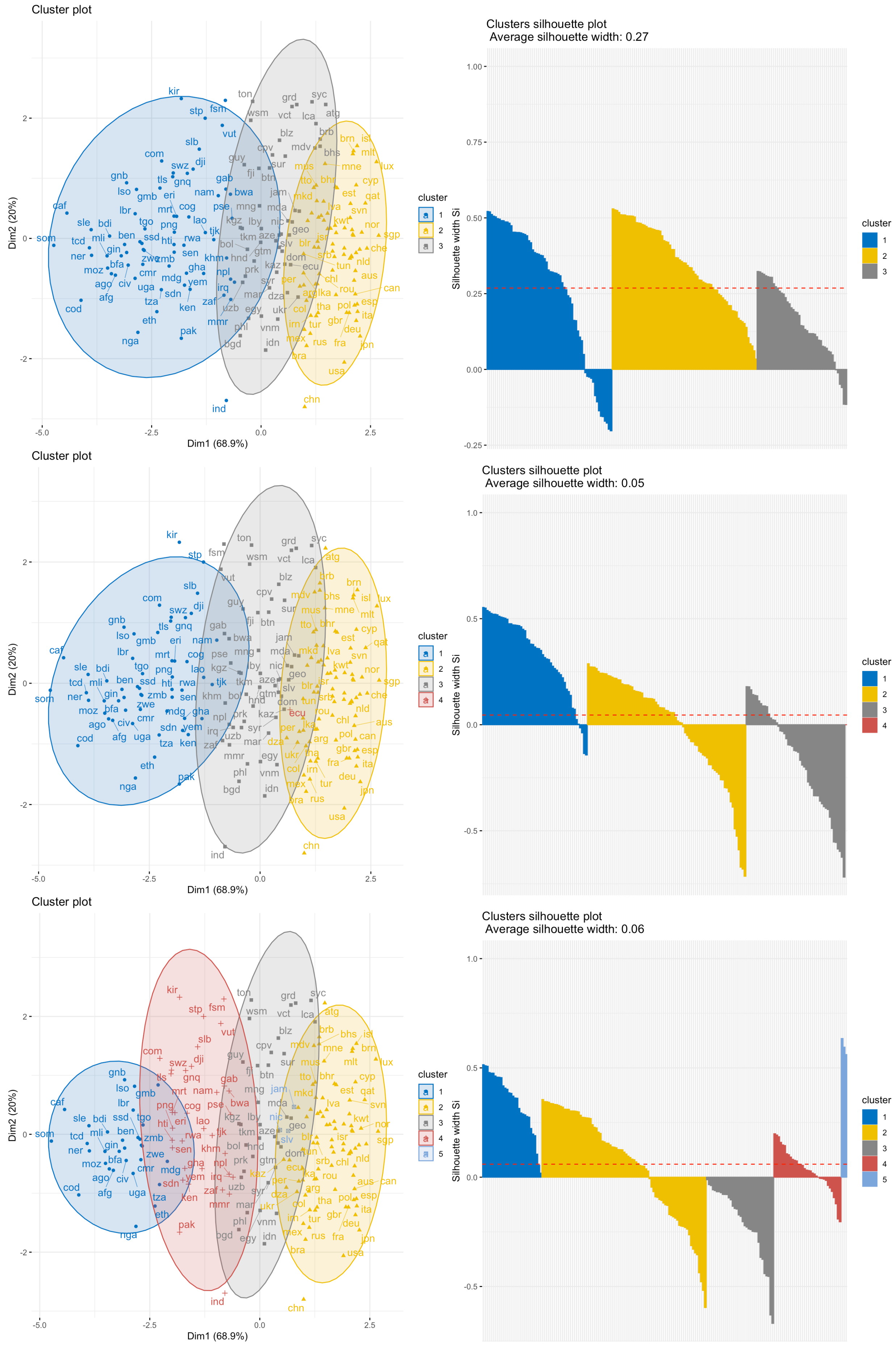


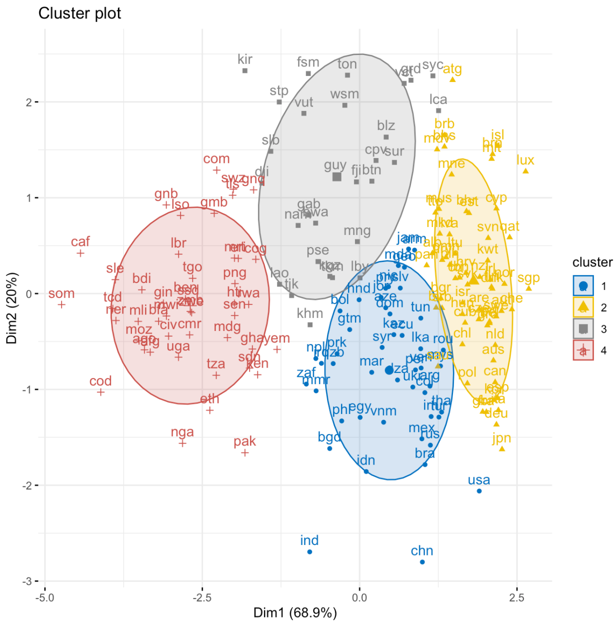
Figure 22.26: Clusters and silhouette profiles for the 2011 Gampinder dataset; FANNY(3) (top row), FANNY(4) (middle row), FANNY(5) (bottom row). The scatterplots are displayed on the data's first two principal components.

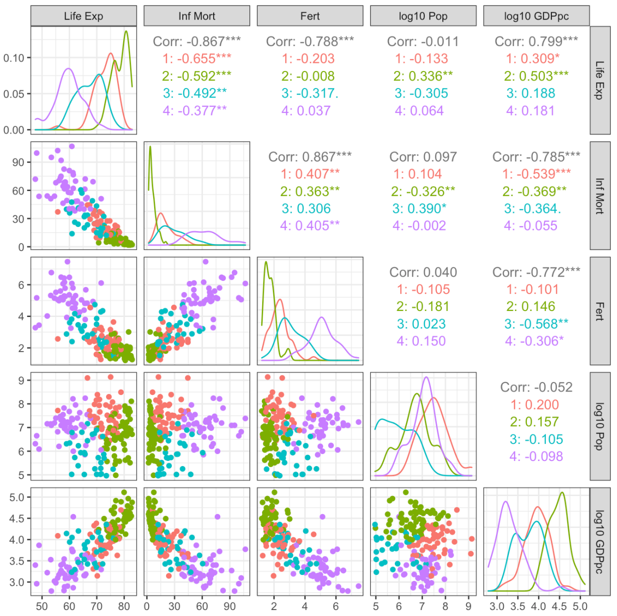
The corresponding results for F4M are computed below.

```
cm.gap <- e1071::cmeans(scale(gapminder.SoCL.2011[,c(3:7)]), 4)
attributes(cm.gap)
```

```
$names
[1] "centers"     "size"        "cluster"     "membership"  "iter"
[6] "withinerror" "call"

$class
[1] "fclust"
```

```
factoextra::fviz_cluster(list(data = scale(gapminder.SoCL.2011[,c(3:7)]),
                                      cluster=cm.gap$cluster),
              ellipse.type = "norm",
              ellipse.level = 0.68,
              palette = "jco",
              ggtheme = theme_minimal())

GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(cm.gap$cluster)),
                diag=list(continuous=my_dens))
```





How does that compare to all the other approaches we have used so far? Based on those, how many clusters do you think there are in the 2011 Gapminder dataset?

22.4.6 Cluster Ensembles

We have seen that the choice of clustering method and algorithm parameters may have an impact on the nature and number of clusters in the data; quite often, the resulting clusters are **volatile**. This is aligned with the idea that the ability to accurately assess the quality of a clustering outcome remains **elusive**, for the most part.

The goal of **ensemble clustering** is to combine the results of multiple clustering runs to create a more **robust** outcome.

Most ensemble models use the following two steps to generate an outcome:

1. **generate** different clustering schemes, using different models, parameters, or data selection mechanisms (the **ensemble components**), and
2. **combine** the different results into a single outcome.

Selecting Different Ensemble Components The ensemble components are either **model-based** or **data selection-based**.

In **model-based ensembles**, the different components of the ensemble reflect different models, such as the use of

- different clustering **approaches**;
- different **parameter settings** for a given approach;
- different **randomizations** (for stochastic algorithms),
- or some **combination** of these.

For instance, an ensemble's components could be built from:

1. 5 runs of $k-$means for each of $k = 2, \dots, 10$, for each of the Euclidean and Manhattan similarities (90 components);
2. the hierarchical clustering outcome for each of the complete, single, average, centroid, and Ward linkage, for each of the Euclidean and Manhattan distances, for each of $k = 2, \dots, 10$ clusters (90 components);
3. the DBSCAN outcome for each of 5 values of ε^*, for each of $minPts = 2, \dots, 10$, for each of the Euclidean and Manhattan distances (90 components), and
4. the spectral clustering outcome for each of 3 threshold values τ, for each of the 3 types of Laplacians, for $k = 2, 4, 6, 8, 10$, for each of the Euclidean and Manhattan distances (90 components),

This provides a total of $4 \times 90 = 360$ components. Note that we could also pick algorithms, settings, and similarity measures randomly, from a list of reasonable options.

In **data selection-based ensembles**, we might select a specific clustering approach, combined with a set of parameters, and a given randomization (if the approach is stochastic) and instead build the different components of the model by running the algorithm on different subsets of the data, either *via*:

- selecting **subsets of observations** using random or other probabilistic sampling scheme;
- selecting **subsets of variables**, again using probabilistic sampling, or
- some **combination** of both.

For instance, an ensemble's components could be built using affinity propagation with Euclidean distance and a specific combination of input preference and dampening parameter, and 360 subsets of the data, obtained as follows:

1. for each component, draw a % of observations to sample and a # of variables to select from the data;
2. randomly select a subset with these properties;
3. run affinity propagation on the subset to obtain a clustering outcome.

We could also combine model-based and data selection-based approaches to create the components.

Combining Different Ensemble Components However the components are obtained, we need to find a way to combine them to obtain a **robust clustering consensus**. There are three basic methods to do this:

- **general affiliation**;
- **hypergraph partitioning**, and
- **meta-clustering**.

In the **general affiliation** approach, we consider each pair of observations and determine how frequently they are found in the same clusters in each of the ensemble components. The corresponding proportions create a **similarity matrix**, which can then be used to cluster the data using some graph-based method, such as DBSCAN.

In the **hypergraph partitioning** approach, each observation in the data is represented by a **hypergraph vertex**. A cluster in any of the ensemble components is represented as a **hypergraph hyperedge**, a generalization of the notion of edge which connects (potentially) more than two vertices in the form of a **complete clique**. This hypergraph is then partitioned using graph clustering methods.[64]

The **meta-clustering** approach is also a graph-based approach, except that vertices are associated with each cluster in the ensemble components; each vertex therefore represents a set of **data objects**. A graph partitioning algorithm is then applied to this graph.[65] **Balancing constraints** may be added to the meta-clustering phase to ensure that the resulting clusters are balanced.

Cluster ensembles are implemented in `R` *via* the packages `diceR` and `clue`. More information is available in [4, 5, 269].

22.5 Exercises

1. Complete the Ward D linkage, maximum dissimilarity hierarchical clustering results for the 2011 data from `gapminder_all.csv` ↗.
2. Cluster the Iowa Housing, Vowel, Wisconsin Breast Cancer, and Wine datasets using $k-$means, for various distance metrics and algorithm parameters. What is your best estimation for the number of clusters in each case? Validate your results.
3. Cluster the Iowa Housing, Vowel, Wisconsin Breast Cancer, and Wine datasets using hierarchical clustering, for various algorithm parameters. Validate your results.
4. Cluster the Iowa Housing, Vowel, Wisconsin Breast Cancer, and Wine datasets using DBSCAN, for various algorithm parameters. Validate your results.

[64] 64: One major challenge with hypergraph partitioning is that a hyperedge can be "broken" by a partitioning in many different ways, not all of which are qualitatively equivalent. Most hypergraph partitioning algorithms use a constant penalty for breaking a hyperedge.

[65] 65: The distribution of the membership of different instances to the meta-partitions can be used to determine its meta-cluster membership, or soft assignment probability.

5. Cluster the Iowa Housing, Vowel, Wisconsin Breast Cancer, and Wine datasets using spectral clustering, for various algorithm parameters. Validate your results.
6. Cluster the Iowa Housing, Vowel, Wisconsin Breast Cancer, and Wine datasets using expectation-maximization clustering, for various algorithm parameters. Validate your results.
7. Cluster the Iowa Housing, Vowel, Wisconsin Breast Cancer, and Wine datasets using affinity propagation clustering, for various algorithm parameters. Validate your results.
8. Cluster the Iowa Housing, Vowel, Wisconsin Breast Cancer, and Wine datasets using fuzzy clustering, for various algorithm parameters. Validate your results.
9. Cluster the Iowa Housing, Vowel, Wisconsin Breast Cancer, and Wine datasets using the combined results of problems 2 to 8. Validate your results.
10. Cluster the datasets

    - `GlobalCitiesPBI.csv`
    - `2016collisionsfinal.csv`
    - `polls_us_election_2016.csv`
    - `HR_2016_Census_simple.xlsx`
    - `UniversalBank.csv`.

    and/or any other datasets of interest, using the approaches discussed in this module (or other other appropriate approaches). Validate your results. Where are there difficulties? What decisions must you make along the way? How could you use the results?