Chapter 6
Clustering

As we saw earlier in the visualizations provided by methods like PCA and SOM, it is often interesting to look for structure, or groupings, in the data. However, these methods do not explicitly define clusters; that is left to the pattern recognition capabilities of the scientist studying the plot. In many cases, however, it is useful to rely on somewhat more formal methods, and this is where clustering methods come in. They are usually based on object-wise similarities or distances, and since the late nineties have become hugely popular in the area of high-throughput measurement techniques in biology, such as DNA microarrays. There, the activities of tens of thousands of genes are measured, often as a function of a specific treatment, or as a time series. Of course, the question is which genes show the same activity pattern: if an unknown gene has much the same behavior as another gene of which it is known that it is involved in a process like cell differentiation, one can hypothesize that the unknown gene is somehow related to this process as well.

With only a slight exaggeration one could say that there are about as many clustering algorithms as there are scientists and by no means do these methods always give the same results. Modern software packages have made many of these clustering methods available to a wide audience; unfortunately, this provides the temptation to try all methods in order to get the result one is looking for, rather than the result that is suggested by the data. There are no formal rules to help you decide which clustering method to use.

One of the reasons for this is that most clustering methods are heuristic in nature, rather than that they stem from solid statistical foundations. Moreover, assessing the quality of the clustering, or *validation*, is a problem: since the "real" clustering is by definition unknown (otherwise it would be more appropriate to use a supervised approach such as the classification methods described in Chap. 7) we can not say that one clustering is better than the other. Also cluster characteristics (sphericity, density, ...) can not be used for this, since different clustering methods "optimize" different criteria. It is often difficult for users to get a good idea of the behavior of the separate methods, since our visualization abilities break down in more than three dimensions, and at the same time the assumptions behind the clustering methods are often unknown.

R. Wehrens, *Chemometrics with R*, Use R!,
https://doi.org/10.1007/978-3-662-62027-4_6

In this chapter, we concentrate on several popular classes of methods. Hierarchical methods are represented by *single*, *average* and *complete linkage*, respectively, while *k-means* is an example of partitional methods. Both yield "crisp" clusterings; objects belong to exactly one cluster. More sophisticated methods lead to a clustering where membership values are assigned to each object: the object can be assigned to the cluster with the highest membership value. An example is given by model-based clustering methods.

6.1 Hierarchical Clustering

Quite often, data have a hierarchical structure in the sense that groups consist of mutually exclusive sub-groups. This is often visualized in a tree-like structure, called a dendrogram. The dendrogram presents an intuitive and appealing way for visualizing the hierarchical structure: the y-axis indicates the "distance" between different groups, whereas the connections show where successive splits (or joins) take place.

Hierarchical clustering starts with a square matrix containing distances or (dis)similarities; in the following we will assume we have the data in the form of distances. It is almost always performed in a bottom-up fashion. Starting with all objects in separate clusters, one looks for the two most similar clusters and joins them. Then, the distance matrix is updated. There are several possibilities to determine the distance between clusters. One option is to take the shortest distance between clusters. In Fig. 6.1 this would correspond to the distance between objects `d` and `D`. This choice leads to the *single-linkage* algorithm. It joins two groups if any members of both groups are close together, a strategy that is sometimes also referred to as friends-of-friends: "any friend of yours is my friend, too!".

The opposite strategy is *complete linkage* clustering: there, the distance between clusters is determined by the objects in the respective clusters that are furthest apart—in Fig. 6.1 objects `a` and `A`. In other words: to belong to the same cluster, the distances to *all* cluster members must be small.[1] This strategy leads to much more compact and

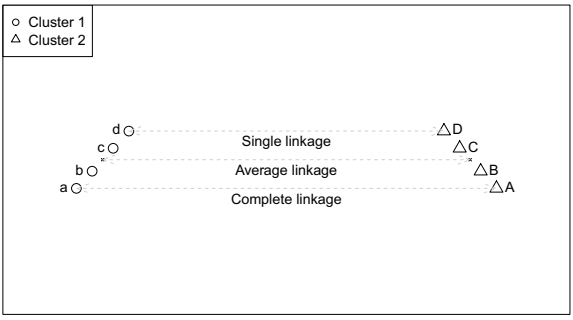

Fig. 6.1 Distances between clusters: single linkage, average linkage and complete linkage consider the closest points, the averages, and the farthest points, respectively

[1] We can only be friends if all our friends are friends of *both* of us.

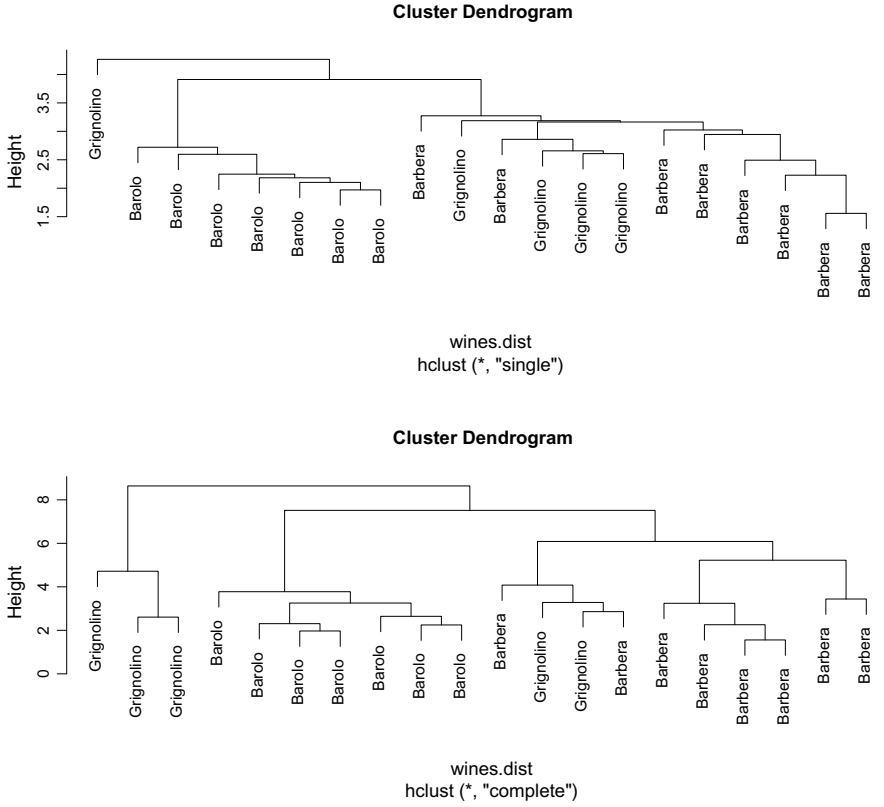

Fig. 6.2 Single linkage clustering (top) and complete linkage clustering (bottom) of 20 samples from the `wine` data

equal-sized clusters. Of course, intermediate strategies are possible, too. Taking the distance between cluster means leads to *average linkage*. *Ward's method* explicitly takes into account the cluster size in calculating a weighted average, and in many cases gives very similar result to average linkage.

Let us see how this works by clustering a random subset of the wine data. In R hierarchical clustering is available through function `hclust`, which takes an object of class `dist` as its first argument:

```
> subset <- sample(nrow(wines), 20)
> wines.dist <- dist(wines.sc[subset, ])
> wines.hcsingle <-  hclust(wines.dist, method = "single")
> plot(wines.hcsingle, labels = vintages[subset])
```

This leads to the dendrogram at the top in Fig. 6.2. When we go down in distance, starting from the top, one Grignolino sample is split off from the main branch as a singleton before the whole Barolo cluster is identified. Going down even further,

individual Grignolino and Barbera samples are split off before arriving at a cluster of Grignolino wines, and a cluster of Barberas.

Also the complete linkage dendrogram in the bottom panel of Fig. 6.2, suggesting a four- or five-cluster solution, shows the confusion between Barberas and Grignolinos, and separate, pure, Barolo and Grignolino clusters. This plot is obtained by:

```
> wines.hccomplete <- hclust(wines.dist, method = "complete")
> plot(wines.hccomplete, labels = vintages[subset])
```

The layout of the dendrogram is very different from the single-linkage one: there, the typical friends-of-friends behaviour is observed, where single objects are gradually added to one large group, in addition to a number of singletons. In complete linkage one often finds more clear distinctions between groups of samples, as is the case here.

In principle, a dendrogram from a hierarchical clustering method in itself is not yet a clustering, since it does not give a grouping as such. However, these can be obtained by "cutting" the diagram at a certain height: all objects that are connected are supposed to be in one and the same cluster. For this, function `cutree` is available, which either takes the height at which to cut, or the number of clusters to obtain as an argument. In this case, let's cut at a height of 3:

```
> wines.cl.single <- cutree(wines.hcsingle, h = 3)
> table(wines.cl.single, vintages[subset])

wines.cl.single Barbera Barolo Grignolino
              1       0      7          0
              2       1      0          0
              3       1      0          3
              4       0      0          1
              5       1      0          0
              6       5      0          0
              7       0      0          1
```

The clustering is very good in the sense that there are almost no mixed clusters containing samples from more than one type; on the other hand, the Barbera wines are split over four different clusters. Cutting the dendrogram at a height larger than three will lead to fewer clusters but inevitably also to more mixed clusters. Conversely, lowering the height at which one cuts leads to more, and more pure clusters. What is most useful needs to be determined on a case-to-case basis.

Now we turn to the complete data set, and recalculate the clusterings. Single linkage, cut at a height to obtain three clusters, does not show anything useful:

```
> wines.dist <- dist(wines.sc)
> wines.hcsingle <- hclust(wines.dist, method = "single")
> table(vintages, cutree(wines.hcsingle, k = 3))

vintages      1  2  3
  Barbera    48  0  0
  Barolo     58  0  0
  Grignolino 67  3  1
```

Almost all samples are in cluster 1, and small bits of the data set (all Grignolino samples) are chipped off leading to clusters 2 and 3, each with only very few elements. On the other hand, the three-cluster solution from complete linkage is already quite good:

```
> wines.hccomplete <- hclust(wines.dist, method = "complete")
> table(vintages, cutree(wines.hccomplete, k = 3))

vintages      1  2  3
  Barbera     3  0 45
  Barolo     50  8  0
  Grignolino 14 52  5
```

Cluster 1 corresponds to mainly Barolo wines, cluster two to Grignolinos, and cluster three to the Barberas. Of course, there still is significant overlap between the clusters.

Hierarchical clustering methods enjoy great popularity: the intuitive visualization through dendrograms is one of the main reasons. These also provide the opportunity to see the effects of increasing the number of clusters, without actually recalculating the cluster structure. Obviously, hierarchical clustering will work best when the data actually have a hierarchical structure: that is, when clusters contain subclusters, or when some clusters are more similar than others. In practice, this is quite often the case.

A further advantage is that the clustering is unique: no random element is involved in creating the cluster model. For many other clustering methods, this is not the case. Note that the uniqueness property is present only in the case that there are no ties in the distances. If there are, one may obtain several different dendrograms, depending on the order of the data and the actual implementation of the software. Usually, the first available merge with the minimal distance is picked. When equal distances are present, one or more equivalent merges are possible, which may lead to different dendrograms. An easy way to investigate this is to repeat the clustering many times on distance matrices from data where the rows have been shuffled.

There are a number of drawbacks to hierarchical clustering, too. For data sets with many samples (more than ten thousand, say) these methods are less suitable. To start with, calculating the distance matrix may be very expensive, or even impossible. More importantly, interpreting the dendrograms quickly becomes cumbersome, and there is a real danger of over-interpretation. Examples where hierarchical methods are used with large data sets can be found in the field of DNA microarrays, where the ground-breaking paper of Eisen et al. (1998) seems to have set a trend.

There are a number of cases where the results of hierarchical clustering can be misleading. The first is the case where in reality there is no class structure. Cutting a dendrogram will always give you clusters: unfortunately, there is no warning light flashing when you investigate a data set with no class structure. Furthermore, even when there are clusters, they may be too close to separate, or they may overlap. In these cases it is impossible to conclude anything about individual cases (although it can still be possible to infer characteristics of the clusters as a whole). The two keys to get out of this conundrum are formed by the use of prior information, and by visualization. If you know class structure is present, and you already have information about part of that structure, the clustering methods that fail to reproduce that knowledge obviously are not performing well, and you are more likely to trust the results of the methods that do find what you already know. Another idea is to visualize the (original) data, and give every cluster a different color and plotting symbol. One can easily see if clusters are overlapping or are nicely separated. Note that the dendrogram can be visualized in a number of equivalent ways: the ordering of the groupings from left to right is arbitrary to some extent and may depend on your software package.

The **cluster** package in R also provides functions for hierarchical clustering: `agnes` implements single, average and complete linkage methods but also allows more control over the distance calculations using the `method ="flexible"` argument. In addition, it provides a coefficient measuring the amount of cluster structure, the "agglomerative coefficient", ac:

$$ac = \frac{1}{n}\sum_i (1 - m_i)$$

where the summation is over all n objects, and m_i is the ratio of the dissimilarity of the first cluster an object is merged to and the dissimilarity level of the final merge (after which only one cluster remains). Compare these numbers for three hierarchical clusterings of the wine data:

```
> wines.agness <- agnes(wines.dist, method = "single")
> wines.agnesa <- agnes(wines.dist, method = "average")
> wines.agnesc <- agnes(wines.dist, method = "complete")


> cbind(wines.agness$ac, wines.agnesa$ac, wines.agnesc$ac)
        [,1]    [,2]    [,3]
[1,] 0.53802 0.69945 0.81625
```

Complete linkage is doing the best job for these data, according to this quality measure.

6.2 Partitional Clustering

A completely different approach is taken by partitional clustering methods. Instead of starting with individual objects as clusters and progressively merging similar clusters, partitional methods choose a set of cluster centers in such a way that the overall distance of all objects to the closest cluster centers is minimised. Algorithms are iterative and usually start with random cluster centers, ending when no more changes in the cluster assignments of individual objects are observed. Again, many different flavours exist, each with its own characteristics. In general, however, these algorithms are very fast and are suited for large numbers of objects. The calculation of the complete distance matrix is unnecessary—only the distances to the cluster centers need to be calculated, where the number of clusters is much smaller than the number of objects—and this saves resources. Two examples will be treated here: k-means and k-medoids. The latter is a more robust version, where outlying observations do not influence the clustering to a large extent.

6.2.1 K-Means

The k-means algorithm is very simple and basically consists of two steps. It is initialized by a random choice of cluster centers, e.g., a random selection of objects in the data set or random values within the range for each variable. Then the following two steps are iterated:

1. Calculate the distance of an object to all cluster centers and assign the object to the closest center; do this for all objects.
2. Replace the cluster centers by the means of all objects assigned to them.

The quality of the final model can then be assessed by summing the distances of all objects to the centers of the clusters to which they are assigned. Note the similarity to the training of SOMs in Chap. 5, in particular to the batch training algorithm. The goals of the two methods, however, are quite different: SOMs aim at providing a suitable mapping to two dimensions, and the units should not be seen as individual clusters, whereas k-means explicitly focusses on finding a specific number of groups.

The basic R function is for k-means clustering conveniently called `kmeans`. Application to the wine data leads to the following result:

```
> (wines.km <- kmeans(wines.sc, centers = 3))
K-means clustering with 3 clusters of sizes 65, 51, 61

Cluster means:
   alcohol malic acid      ash ash alkalinity magnesium
1 -0.91833   -0.39533 -0.49050        0.16370 -0.483216
2  0.17364    0.86425  0.18718        0.51684 -0.064971
3  0.83336   -0.30131  0.36617       -0.60655  0.569222
  tot. phenols flavonoids non-flav. phenols   proanth col. int.
1    -0.071141   0.026589         -0.037096  0.065095  -0.89558
2    -0.971065  -1.206242          0.719152 -0.771710   0.93782
3     0.887680   0.980165         -0.561730  0.575837   0.17023
  col. hue OD ratio  proline
1  0.46141  0.28236 -0.74607
2 -1.15662 -1.28723 -0.40027
3  0.47535  0.77533  1.12965

Clustering vector:
  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [31] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1
 [61] 2 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1
 [91] 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1
[121] 3 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[151] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

Within cluster sum of squares by cluster:
[1] 559.30 326.35 382.19
 (between_SS / total_SS =  44.6 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"
[5] "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
```

The algorithm not only returns the clustering of the individual objects, but also cluster-specific information such as the sum of squares, the cumulative distance of all cluster objects to the center of the cluster.

So the question now is: how good is the agreement with the vintages? Let's see:

```
> table(vintages, wines.km$cluster)

vintages      1  2  3
  Barbera     0 48  0
  Barolo      0  0 58
  Grignolino 65  3  3
```

Only six of the Grignolino samples are classified as Barbera and Barolo wines; a lot better than the complete-linkage solution.

The k-means algorithm enjoys great popularity through its simplicity, ease of interpretation, and speed. It does have a few drawbacks, however. We already mentioned the fact that one should pick the number of clusters in advance. In general, the

correct number (if such a thing exists at all) is never known, and one will probably try several different clusterings with different numbers of clusters. Whereas hierarchical clustering delivers this in one go—the dendrogram only has to be cut at different positions—for k-means clustering (and partitional methods in general) one should repeat the whole clustering procedure. As already said, the results with four or five clusters may differ dramatically.

Worse, even a repeated clustering with the *same* number of clusters will give a different result, sometimes even a very different result. Remember that we start from a random initialization: an unlucky starting point may get the algorithm stuck in a local minimum. Repeated application, starting from different initial guesses, gives some idea of the variability. The `kmeans` function returns the within-cluster sums of squares for the separate clusters, which can be used as a quality criterion:

```
> wines.km <- kmeans(wines.sc, centers = 3)
> best <- wines.km
> for (i in 1:100) {
+   tmp <- kmeans(wines.sc, centers = 3)
+   if (sum(tmp$withinss) < sum(best$withinss))
+     best <- tmp
+ }
```

One can then pick the one that leads to the best description of the data or, equivalently, the smallest overall distance. In this particular case, the overall best solution is found every time—the wine data do not present that much of a problem. The `kmeans` function has a built-in argument for repeating the clustering and only returning the best solution. Thus, the loop in the previous example can be replaced by

```
> wines.km <- kmeans(wines.sc, centers = 3, nstart = 100)
```

Several minima with comparable overall distance measure may exist, so that different but equally good clustering solutions can be found by the algorithm.

6.2.2 K-Medoids

In k-means, cluster centers are given by the mean coordinates of the objects in that cluster. Since averages are very sensitive to outlying observations, the clustering may be dominated by a few objects, and the interpretation may be difficult. One way to resolve this is to assess clusterings with more groups than expected: the outliers may end up in a cluster of their own. A more practical alternative would be to use a more robust algorithm where the influence of outliers is diminished. One example is the k-medoids algorithm (Kaufman and Rousseeuw 1990), available in R through the function `pam`—Partitioning Around Medoids— in the **cluster** package. Rather than finding cluster centers at optimal positions, k-medoids aims at finding k representative objects within the data set. Typically, the sum of the distances is

minimized rather than the sum of the squared distances, decreasing the importance of large distances.

Applied to the wine data, k-medoids gives the following result:

```
> (wines.pam <- pam(wines.dist, k = 3))
Medoids:
      ID
[1,]  35  35
[2,] 106 106
[3,] 148 148
Clustering vector:
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [31] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2
 [61] 2 2 1 2 1 2 2 2 1 2 1 2 1 1 2 2 2 1 1 2 2 2 3 2 2 2 2 2 2
 [91] 2 2 2 2 1 1 2 1 2 2 2 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 1
[121] 1 2 2 1 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[151] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
Objective function:
 build   swap
2.9085 2.8086

Available components:
[1] "medoids"    "id.med"     "clustering" "objective"
[5] "isolation"  "clusinfo"   "silinfo"    "diss"
[9] "call"
```

The result presents the medoids with their row numbers. The objective function, the sum of the distances of objects to the medoids, is reported in two stages: the first stage serves to find a good initial set of medoids, whereas the second stage performs a local search, trying all possible medoid swaps until no more improvement can be found. In this rather simple example, the average distance after the second stage has decreased by 0.1, compared to the distances to the initial set of medoids—not a huge decrease.

The implementation of `pam` in the **cluster** package comes with additional visualization methods. The first is the "silhouette" plot (Kaufman and Rousseeuw 1990). It shows a quality measure for individual clusterings: object with a high silhouette width (close to 1) are very well clustered, while objects with low values lie in between two or more clusters. Objects with a negative value may be even in the wrong cluster. The silhouette width s_i of object i is given by:

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

where a_i is the average distance of object i to all other objects in the same cluster, and b_i is the smallest distance of object i to another cluster. Thus, the maximal value will be obtained in those cases where the intra-cluster distance a is much smaller than the inter-cluster distance b.

For the wine data clustering, the silhouette plot shown in Fig. 6.3 is obtained by:

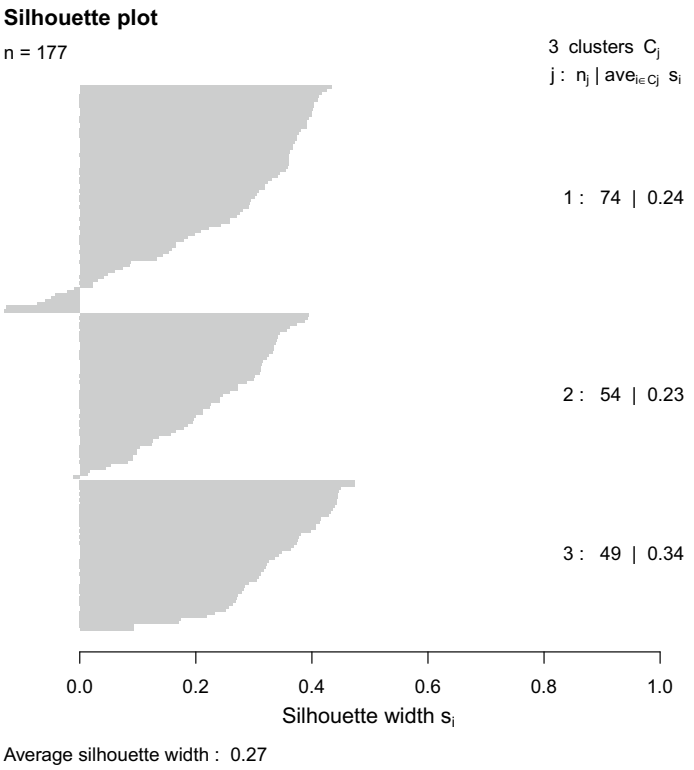

Fig. 6.3 Silhouette plot for the k-medoids clustering of the wine data. The three clusters contain 74, 54 and 49 objects, and have average silhouette widths of 0.24, 0.23 and 0.34, respectively

```
> plot(wines.pam, main = "Silhouette plot")
```

An overall measure of clustering quality can be obtained by averaging all silhouette widths. This is an easy way to decide on the most appropriate number of clusters:

```
> best.pam <- pam(wines.dist, k = 2)
> for (i in 3:10) {
+   tmp.pam <- pam(wines.dist, k = i)
+   if (tmp.pam$silinfo$avg.width < best.pam$silinfo$avg.width)
+     best.pam <- tmp.pam
+ }
> best.pam$medoids
[1]  12  56  34  97  91 163 125 148
```

In this case, eight clusters seem to give the clustering with the least ambiguity. The agreement with the true class labels is quite good:

```
> table(vintages, best.pam$clustering)

vintages      1  2  3  4  5  6  7  8
  Barbera     0  0  0  0  0 18  0 30
  Barolo     21 17 20  0  0  0  0  0
  Grignolino  0  1  5 20 15  5 24  1
```

Clusters 1, 2 and 3 correspond to the Barolo wines, and clusters 6 and 8 to the Barbera. Again, the Grignolino wines are the most difficult to cluster, and 12 Grignolino samples end up in clusters dominated by other wines.

For large data sets, `pam` is too slow; in the **cluster** package, an alternative is provided in the function `clara` (Kaufman and Rousseeuw 1990) which considers subsets of size `sampsize`. Each subset is partitioned using the same algorithm as in `pam`. The sets of medoids that result are used to cluster the complete data set, and the best set of medoids, i.e., the one for which the sum of the distances is minimal, is retained.

6.3 Probabilistic Clustering

In probabilistic clustering, sometimes also called fuzzy clustering, objects are not allocated to one cluster only. Rather, cluster memberships are used to indicate which of the clusters is more likely. If a "crisp" clustering result is needed, an object is assigned to the cluster with the highest membership value.

The most well-established methods are found in the area of mixture modelling, where individual clusters are represented by mixtures of parametric distributions, and the overall clustering is a weighted sum of the individual components (McLachlan and Peel 2000; Fraley and Raftery 2002). Usually, multivariate normal distributions are applied. In that case, assuming G clusters, the likelihood is given by

$$L(\boldsymbol{\tau}, \boldsymbol{\mu}, \boldsymbol{\Sigma}|\mathbf{x}) = \prod_{i=1}^{n} \sum_{k=1}^{G} \tau_k \phi_k(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \ ,$$

where τ_k is the fraction of objects in cluster k, $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ correspond to the cluster means and covariance matrices of cluster k, respectively, and ϕ_k is the density of cluster k. If the cluster labels would be known, one could estimate the unknown parameters τ_k, $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ by maximizing the likelihood (for example). Vice versa, when these parameters are known, it is easy to calculate the conditional probabilities of belonging to class k:

$$z_{ik} = \phi_k(\mathbf{x}_i|\theta_k) / \sum_{j=1}^{K} \phi_j(\mathbf{x}_i|\theta_j)$$

These two steps are the components in the Expectation-Maximization algorithm (EM) (Dempster et al. 1977; McLachlan and Krishnan 1997): estimating the conditional probabilities is indicated with the E-step, whereas estimating the parameters (class means and variances, and mixing proportions) is the M-step. The conditional probabilities z_{ik} can also be seen as indicators of uncertainty: the larger $z_{i,\max}$, the maximal value of all z_{ik} values for object i, the more certain the classification.

One can use the likelihood to determine what number of clusters is optimal. Of course, the likelihood will increase with the number of clusters, so one usually defines a penalty depending on the number of parameters that are estimated. Two popular measures are Akaike's Information Criterion (AIC) and the Bayesian Information Criterion (BIC). The AIC criterion (Akaike 1974) is defined by

$$AIC = -2 \log L + 2p \tag{6.1}$$

where L is the likelihood and p the number of parameters in the model (here $\boldsymbol{\tau}$, $\boldsymbol{\mu}$, and $\boldsymbol{\Sigma}$). The closely related BIC criterion (Schwarz 1978) uses a penalty that is usually stronger than the AIC penalty:

$$BIC = -2 \log L + p \log n \tag{6.2}$$

The optimal model has a minimal value for AIC and/or BIC[2]—because of the more heavy penalty, BIC is likely to select slightly more parsimonious models than AIC. Several other criteria exist (McLachlan and Peel 2000). None of these is able to correctly identify the number of clusters in all cases, but in practice, differences are not very big and both AIC and BIC criteria are often used.

Several packages in R implement this form of clustering. In the **mclust** package (Fraley and Raftery 2003), for example, one can calculate BIC values for different numbers of clusters easily:

```
> wines.BIC <- mclustBIC(wines.sc, modelNames = "VVV")
> plot(wines.BIC)
```

This produces the plot in Fig. 6.4. The BIC value, here given in its negative form, has a maximum at two clusters, which will be the model picked by the function `mclustModel` if no specific number of clusters is given. Alternatively, one can specify a specific number of clusters by providing a value for the `G` argument:

```
> wines.mclust2 <- mclustModel(wines.sc, wines.BIC)
> wines.mclust3 <- mclustModel(wines.sc, wines.BIC, G = 3)
```

One can make scatter plots at specific combinations of variables with the `coordProj` function, visualizing the clustering in low-dimensional subspaces. Also the uncertainties, given by $1 - z_{i,\max}$, can be visualized. This provides an easy way to compare the two- and three-cluster solutions graphically:

[2]Especially for the BIC value, one often sees the negative form so that maximization will lead to an optimal model. This is also the definition by (Schwarz 1978).

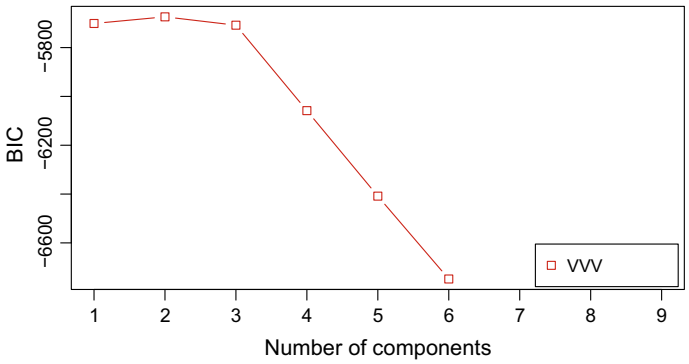

Fig. 6.4 BIC values for clustering the autoscaled wine data with **mclust**. The label "VVV" indicates a completely unconstrained model. The optimal model has two clusters

```
> par(mfrow = c(2, 2))
> coordProj(wines.sc, dimens = c(7, 13),
+           parameters = wines.mclust2$parameters,
+           z = wines.mclust2$z, what = "classification")
> title("2 clusters: classification")
> coordProj(wines.sc, dimens = c(7, 13),
+           parameters = wines.mclust3$parameters,
+           z = wines.mclust3$z, what = "classification")
> title("3 clusters: classification")
> coordProj(wines.sc, dimens = c(7, 13),
+           parameters = wines.mclust2$parameters,
+           z = wines.mclust2$z, what = "uncertainty")
> title("2 clusters: uncertainty")
> coordProj(wines.sc, dimens = c(7, 13),
+           parameters = wines.mclust3$parameters,
+           z = wines.mclust3$z, what = "uncertainty")
> title("3 clusters: uncertainty")
```

The result, here for the variables `flavonoids` and `proline`, is shown in Fig. 6.5. The top row shows the classifications of the two- and three-cluster models, respectively. The bottom row shows the corresponding uncertainties.

Just like with k-means and k-medoids, the clustering using the EM algorithm needs to be kick-started with an initial guess. This may be a random initialization, but the EM algorithm has a reputation for being slow to converge, and an unlucky guess may lead into a local optimum. In `mclust`, the initialization is done by hierarchical clustering.[3] This has the advantage that initial models for many different numbers of clusters can be generated quickly. Moreover, this initialization algorithm is stable in the sense that the same clustering is obtained upon repetition. A BIC table, such as the one depicted in Fig. 6.4 is therefore easily obtained.

[3] To be more precise, model-based hierarchical clustering (Fraley 1998).

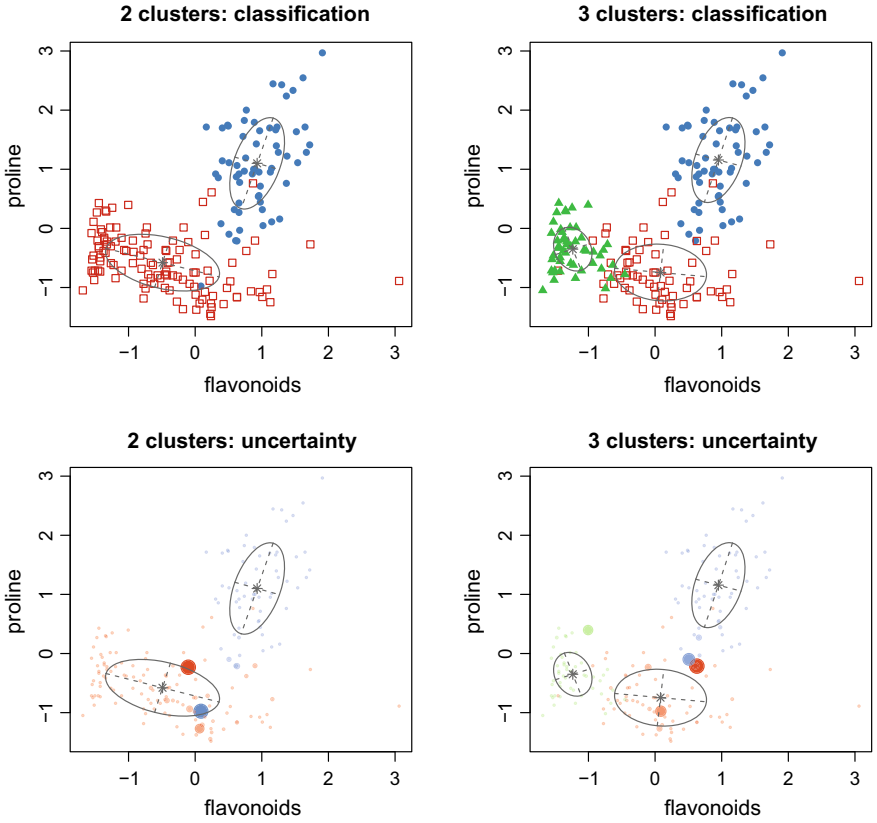

Fig. 6.5 Two- and three-cluster models for the `wines` data, obtained by **mclust**. The top row shows the classifications; the bottom row shows uncertainties at three levels, where the smallest dots have z-values over 0.95 and the largest, black, dots have z-values below 0.75. The others are in between

While mixtures of gaussians (or other distributions) have many attractive properties, they suffer from one big disadvantage: the number of parameters to estimate quickly becomes large. This is the reason why the BIC curve in Fig. 6.4 does not run all the way to nine clusters, although that is the default in `mclust`: in high dimensions, clusters with only few members quickly lead to singular covariance matrices. In such cases, no BIC value is returned. Banfield and Raftery (Banfield and Raftery 1993) suggested to impose restrictions on the covariance matrices of the clusters: one can, e.g., use spherical and equal-sized covariance matrices for all clusters. In this case, which is also the most restricted, the criterion that is optimized corresponds to the criterion used in k-means and in Ward's hierarchical clustering. For each cluster, Gp parameters need to be estimated for the cluster centers, one parameter for the covariance matrices, and p mixing proportions, a total of $(G+1)p+1$. In contrast, for the completely free model such as the ones in Figs. 6.4 and 6.5, indicated with

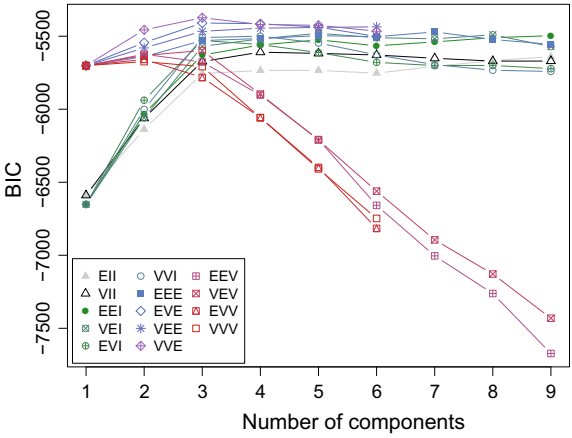

Fig. 6.6 BIC plots for all covariance models implemented in `mclust`: although the constrained models do not fit as well for the same numbers of clusters, they are penalized less and achieve higher BIC values for larger numbers of clusters

"VVV" in `mclust`, every single covariance matrix requires $p(p+1)/2$ parameters. This leads to a grand total of $p(Gp+G+4)/2$ estimates. For low-dimensional data, this is still doable, but for higher-dimensional data the unrestricted models are no longer workable.

Consider the wine data again, but now consider all ten models implemented in `mclust`:

```
> wines.BIC <- mclustBIC(wines.sc)
> plot(wines.BIC, legendArgs = list(x = "bottom", ncol = 2))
```

This leads to the output in Fig. 6.6. The three-letter codes in the legend stand for volume, shape and orientation, respectively. The "E" indicates equality for all clusters, the "V" indicates variability, and the "I" indicates identity. Thus, the "EEI" model stands for diagonal covariance matrices (the "I") with equal volumes and shapes, and the "VEV" model indicates an ellipsoidal model with equal shapes for all clusters, but complete freedom in size and orientation. It is clear that the more constrained models achieve much higher BIC values for higher numbers of clusters: the unconstrained models are penalized more heavily for estimating so many parameters.

6.4 Comparing Clusterings

In many cases, one is interested in comparing the results of different clusterings. This may be to assess the behavior of different methods on the same data set, but also to find out how variable the clusterings are that are obtained by randomly initialized methods like k-means. The difficulty here, of course, is that there is no golden standard; one cannot simply count the number of incorrect assignments and use that as a quality criterion. Moreover, the number of clusters may differ—still we may be interested in assessing the agreement between the partitions.

Several measures have been proposed in literature. Hubert (1985) compares several of these, and proposes the adjusted Rand index, inspired by earlier work by Rand (1971). The original Rand index is based on the number of times two objects are classified in the same cluster, n. In the formulas below, $n_{i\cdot}$ indicates the number of object pairs classified in the same cluster in partition one, but not in partition two, $n_{\cdot j}$ the reverse, and n_{ij} the number of pairs classified in different clusters in both partitions. The index, comparing two partitions with I and J objects, respectively, is given by

$$R = \binom{n}{2} + 2\sum_{i=1}^{I}\sum_{j=1}^{J}\binom{n_{ij}}{2} - \left\{\sum_{i=1}^{I}\binom{n_{i\cdot}}{2} + \sum_{j=1}^{J}\binom{n_{\cdot j}}{2}\right\} \tag{6.3}$$

The adjusted Rand index "corrects for chance" by taking into account the expected value of the index under the null hypothesis of random partitions:

$$R_{\text{adj}} = \frac{R - E(R)}{\max(R) - E(R)} = \frac{a\binom{n}{2} - bc}{\frac{1}{2}\binom{n}{2}(b+c) - bc} \tag{6.4}$$

with

$$a = \sum_{i,j}\binom{n_{ij}}{2} \tag{6.5}$$

$$b = \sum_{i}\binom{n_{i\cdot}}{2} \tag{6.6}$$

$$c = \sum_{j}\binom{n_{\cdot j}}{2} \tag{6.7}$$

This measure is zero when the Rand index takes its expected value, and has a maximum of one.

The implementation in R takes only a few lines:

```
> AdjRkl <- function(part1, part2) {
+   confusion <- table(part1, part2)
+
+   n <- sum(confusion)
+   a <- sum(choose(confusion[confusion>1], 2))
+   b <- apply(confusion, 1, sum)
+   b <- sum(choose(b[b>1], 2))
+   c <- apply(confusion, 2, sum)
+   c <- sum(choose(c[c>1], 2))
+
+   Rexp <- b*c/choose(n, 2)
+   (a - Rexp) / (.5*(b+c) - Rexp )
+ }
```

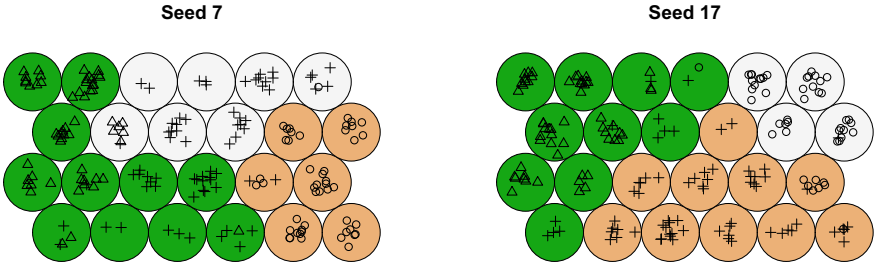

Fig. 6.7 Clustering of the codebook vectors of two mappings of the wine data, indicated by background colors. Symbols indicate vintages

The function takes two partitionings, i.e., class vectors, and returns the value of the adjusted Rand index. Note that the number of classes in both partitionings need not be the same. An alternative is function `adjustedRandIndex` in package **mclust**.

How this can be useful is easily illustrated. As already stated, repeated application of SOM mapping will, in general, lead to mappings that visually can appear very different. However, objects may find themselves very close to the same neighbors in repeated training runs, so that conclusions from the two maps will be very much the same. One way to investigate that is to quantify the similarities. Consider the SOM mapping of the wine data for two initializations:

```
> set.seed(7)
> som.wines <- som(wines.sc, grid = somgrid(6, 4, "hexagonal"))
> set.seed(17)
> som.wines2 <- som(wines.sc, grid = somgrid(6, 4, "hexagonal"))
```

Assessing the similarities of the maps should not be done on the level of the individual units, since these are not relevant entities in themselves. Rather, the units should be aggregated into larger clusters. This can be achieved by looking at plots like Fig. 5.5; an alternative is to explicitly cluster the codebook vectors (see, e.g., Vesanto and Alhoniemi 2000). If hierarchical clustering is used, the dendrograms can be cut at the desired level, immediately providing cluster memberships for the individual samples.

```
> som.hc <- cutree(hclust(dist(getCodes(som.wines, 1))), k = 3)
> som.hc2 <- cutree(hclust(dist(getCodes(som.wines2, 1))), k = 3)
> plot(som.wines, "mapping", bgcol = terrain.colors(3)[som.hc],
+      pch = as.integer(vintages), main = "Seed 7")
> plot(som.wines2, "mapping", bgcol = terrain.colors(3)[som.hc2],
+      pch = as.integer(vintages), main = "Seed 17")
```

This leads to the plots in Fig. 6.7.

The mappings seem very different. Is this really the case, or is it just a visual artifact? Let's find out:

```
> som.clust <- som.hc[som.wines$unit.classif]
> som.clust2 <- som.hc2[som.wines2$unit.classif]
> AdjRkl(som.clust, som.clust2)
[1] 0.4501
```

This rather low value suggests that both mappings are quite different. Note that this analysis does not take into account the vintages and is applicable also in cases where "true" class labels are unknown. Of course, one can also use the adjusted Rand index to compare clusterings with a set of "true" labels:

```
> AdjRkl(vintages, som.clust)
[1] 0.47699
> AdjRkl(vintages, som.clust2)
[1] 0.67278
```

Clearly, the second random seed gives a mapping that is more in agreement with the class labels, something that is also clear when looking at the agreement between plotting symbols and background color in Fig. 6.7.

Other indices to measure correspondence between two partitionings include Fowlkes' and Mallows's B_k (Fowlkes and Mallows 1983), Goodmans and Kruskals γ (Goodman and Kruskal 1954), and Meila's Variation of Information criterion (Meila 2007), also available in **mclust**. The latter is a difference measure, rather than a similarity measure.

6.5 Discussion

Hierarchical clustering methods have many attractive features. They are suitable in cases where there is a hierarchical structure, i.e., subclusters, which very often is the case. A large number of variables does not pose a problem: the rate-limiting step is the calculation of the distance matrix, the size of which does not depend on the dimensionality of the data, but only on the number of samples. And last but not least, the dendrogram provides an appealing presentation of the cluster structure, which can be used to assess clusterings with different numbers of clusters very quickly. Partitional methods, on the other hand, are more general. In hierarchical clustering a split cannot be undone – once a sample is in one branch of the tree, there is no way it can move to the other branch. This can lead, in some cases, to suboptimal clusterings. Partitional methods do not know such restrictions: a sample can always be classified into a different class in the next iteration. Some of the less complicated partitional methods, such as k-means clustering, can also be applied with huge data sets, containing tens of thousands of samples, that cannot be tackled with hierarchical clustering.

Both types of clustering have their share of difficulties, too. In cases relying on distance calculations (all hierarchical methods, and some of the partitional methods, too), the choice of a distance function can dramatically influence the result. The

importance of this cannot be overstated. On the one hand, this is good, since it allows one to choose the most relevant distance function available—it even allows one to tackle data that do not consist of real numbers but are binary or have a more complex nature. As long as there is a distance function that adequately represents dissimilarities between objects, the regular clustering methods can be applied. On the other hand, it is bad: it opens up the possibility of a wrong choice. Furthermore, one should realize that when correlated groups of variables are present, as often is the case in life science data, these variables may receive a disproportionally large weight in a regular distance measure such as Euclidean distance, and smaller groups of variables, or uncorrelated variables, may fail to be recognized as important.

Partitional methods force one to decide on the number of clusters beforehand, or perform multiple clusterings with different numbers of clusters. Moreover, there can be considerable differences upon repeated clustering, something that is less prominent in hierarchical clustering (only with ties in the distance data). The main problem with hierarchical clustering is that the bottom-up joining procedure may be too strict: once an object is placed in a certain category, it will stay there, whatever happens further on in the algorithm. Of course, there are many examples where this leads to a sub-optimal clustering. More generally, there may not be a hierarchical structure to begin with.

Both partitional and hierarchical clustering yield "crisp" clusters, that is, objects are assigned to exactly one cluster, without any doubt. For partitional methods, there are alternatives where each object gets a membership value for each of the clusters. If a crisp clustering is required, at the end of the algorithm the object is assigned to the cluster for which it has the highest membership. We have seen one example in the model-based clustering methods.

Finally, one should take care not to over-interpret the results. If you ask for five clusters, that is exactly what you get. Suppose one has a banana-shaped cluster. Methods like k-means, but also complete linkage, will typically describe such a banana with three or four spherical clusters. The question is: are you interested in the peas or the pod[4]? It may very well be that several clusters in fact describe one and the same group, and that to find the other clusters one should actually instruct the clustering to look for more than five clusters.

Clustering is, because of the lack of "hard" criteria, more of an art than a science. Without additional knowledge about the data or the problem, it is hard to decide which one of several different clusterings is best. This, unfortunately, in some areas has led to a practice in which all available clustering routines are applied, and the one that seems most "logical" is selected and considered to describe "reality". One should always keep in mind that this may be a gross overestimation of the powers of clustering.

[4]Metaphor from Adrian Raftery.