13

Cluster Analysis of Genomic Data

K. S. Pollard and M. J. van der Laan

Abstract

We provide an overview of existing partitioning and hierarchical clustering algorithms in R. We discuss statistical issues and methods in choosing the number of clusters, the choice of clustering algorithm, and the choice of dissimilarity matrix. We also show how to visualize a clustering result by plotting ordered dissimilarity matrices in R. A new R package hopach, which implements the Hierarchical Ordered Partitioning And Collapsing Hybrid (HOPACH) algorithm, is presented (van der Laan and Pollard, 2003). The methodology is applied to a renal cell cancer gene expression data set.

13.1 Introduction

As the means for collecting and storing ever larger amounts of data develop, it is essential to have good methods for identifying patterns. For example, an important goal with large-scale gene expression studies is to find biologically important subsets of genes or samples. Clustering algorithms have been widely applied to microarray microarray data analysis (Eisen et al., 1998).

Consider a study in which one collects on each of I randomly sampled subjects (or more generally, experimental units) a J-dimensional gene expression profile X_i, $i = 1, \ldots, I$: for example, X_i can denote the gene expression profile of cancer tissue relative to healthy tissue within a randomly sampled cancer patient. To view clustering as a statistical procedure it is important to consider X_i as an observation of a random vector with a population distribution we will denote with P. These I independent and identically distributed (*i.i.d.*) observations can be stored in an observed $J \times I$ data matrix X. Genes are represented by I-dimensional vectors

$[X_i(j) : i = 1, \ldots, I]$, while the samples are represented by J-dimensional vectors X_i. The goal could now be to cluster genes or samples. A cluster is a group of similar elements. Each cluster can be represented by a profile, either a summary measure such as a cluster mean or one of the elements itself, which is called a medoid or centroid.

13.2 Methods

13.2.1 Overview of clustering algorithms

For the sake of presenting a unified view of available clustering algorithms, we generalize the output of a clustering algorithm as a sequence of clustering results indexed by the number of clusters $k = 2, 3, \ldots$ and options such as the choice of dissimilarity metric. The algorithm is a mapping from the empirical distribution of $X_1, \ldots, X_I$ to this sequence of k-specific clustering results. For instance, this mapping could be the construction of an agglomerative hierarchical tree of gene clusters using 1 minus correlation as dissimilarity and single linkage as distance between clusters. Given a clustering algorithm, consider the output if the algorithm were applied to the data generating distribution P (i.e., infinite sample size). We call this output a *clustering parameter*, where we stress that any variation in the algorithm results in a different parameter. An example is the J-dimensional vector of gene cluster labels produced by applying a particular partitioning method (e.g., k-means using Euclidean distance) with a particular number of clusters (e.g., $k = 5$) to P. We might think of these as the true cluster labels, in contrast to the observed labels from a sample of size I. Another parameter is the k-dimensional vector of cluster sizes produced by the same algorithm.

We will focus on non-parametric clustering algorithms, in which one makes no assumptions about the data generating distribution P. Model based clustering algorithms are based on assuming that the vectors X_i are *i.i.d.* from a mixture of distributions (e.g., a multivariate normal mixture). The clustering result is typically a summary measure, such as the conditional probabilities of cluster membership (given the data), of the maximum likelihood estimator of the data generating distribution (Fraley and Raftery, 1998, 2000). Of course, if one only views this mixture model as a working model to define a clustering result, then these approaches fall in the category of non-parametric clustering algorithms. In this case, however, statistical inference cannot be based on the working model, and, contrary to the case in which one assumes this mixture model to contain the true data generating distribution, there does not exist a true number of clusters.

13.2.2 Ingredients of a clustering algorithm

We review here the choices one needs to consider before performing a cluster analysis.

Dissimilarity matrix: All clustering algorithms are (either implicitly or explicitly) indexed by a choice of dissimilarity measure, which quantifies the distinctness of each pair of elements (see Chapter 12). For clustering genes, this is a $J \times J$ symmetric matrix. Typical choices of dissimilarity include Euclidean distance, Manhattan distance, 1 minus correlation, 1 minus absolute correlation and 1 minus cosine-angle (i.e., : 1 minus uncentered correlation). The R function `dist` allows one to compute a variety of dissimilarities. Other distance functions are available in the function `daisy` from the cluster package or from the bioDist package. In hopach we have written `distancematrix` and implemented specialized versions of many of the standard distances. Data transformations, such as standardization of rows or columns, are some times performed before computing the dissimilarity matrix.

Number of clusters: One must specify the number of clusters or an algorithm for determining this number. In Section 13.2.7, we discuss and compare methods for selecting the number of clusters, including various data-adaptive approaches.
Criterion: Clustering algorithms are deterministic mappings that aim to optimize some criterion. This is often a real-valued function of the cluster labels that measures how similar elements are within clusters or how different elements are between clusters. The choice of criterion can have a dramatic effect on the clustering result. We recommend a careful study of a proposed criterion so that the user fully understands its strengths and weaknesses (i.e., its scoring strategy) in evaluating a clustering result. Simulations are a useful tool for comparing different criteria.

Searching strategy: One sensible goal is to find the clustering result that globally maximizes the selected criterion. Because of computational issues, heuristic search strategies (that guarantee convergence to a local maximum) are often needed. If the user prefers a tree structure linking all clusters, then forward or backward selection strategies are often used, and they do not correspond with local maxima of the criterion.

13.2.3 Building sequences of clustering results

We can classify clustering algorithms by their searching strategies. Figure 13.1 compares the clustering results from a partitioning (`pam`) and a hierarchical (`diana`) algorithm.

Partitioning: Partitioning methods, such as self-organizing maps (SOM) (Törönen et al., 1999), partitioning around medoids (PAM) (Kaufman and Rousseeuw, 1990), and k-means, map a collection of elements (e.g., genes) into $k \geq 2$ disjoint clusters by aiming to maximize a particular criterion. In this case, a clustering result for $k = 2$ is not used in computing the clustering result for $k = 3$.
Hierarchical: Hierarchical methods involve constructing a tree of clusters in which the root is a single cluster containing all the elements and the leaves each contain only one element. These trees are typically binary; that is, each node has exactly two children. The final level of the tree can be viewed as an ordered list of the elements, though most algorithms produce an ordering that is very dependent on the initial ordering of the data, and is thus not necessarily distance based.
A hierarchical tree can be divisive (i.e., built from the top down by recursively partitioning the elements) or agglomerative (i.e., built from the bottom up by recursively combining the elements) . The R function `diana` [R package cluster, Kaufman and Rousseeuw (1990)] is an example of a divisive hierarchical algorithm, while `agnes` (R package cluster,Kaufman and Rousseeuw (1990)) and Cluster (Eisen et al., 1998) are examples of agglomerative hierarchical algorithms. Agglomerative methods can be employed with different types of linkage, which refers to the distance between groups of elements and is typically a function of the dissimilarities between pairs of elements. In average linkage methods, the distance between two clusters is the average of the dissimilarities between the elements in one cluster and the elements in the other cluster. In single linkage methods (nearest neighbor methods), the dissimilarity between two clusters is the smallest dissimilarity between an element in the first cluster and an element in the second cluster.
Hybrid: The hierarchical ordered partitioning and collapsing hybrid (HOPACH) algorithm (van der Laan and Pollard, 2003) builds a tree of clusters, where the clusters in each level are ordered based on the pairwise dissimilarities between cluster medoids. This algorithm starts at the root node and aims to find the right number of children for each node by alternating partitioning (divisive) steps with collapsing (agglomerative) steps. The resulting tree is non-binary with a deterministically ordered final level.

Several R packages contain clustering algorithms. Table 13.2.3 provides a non-exhaustive list. We use the `agriculture` data set from the package cluster to demonstrate code and output of some standard clustering methods.

```
> library("cluster")
> data(agriculture)
```

```
> part <- pam(agriculture, k = 2)
> round(part$clusinfo, 2)
```

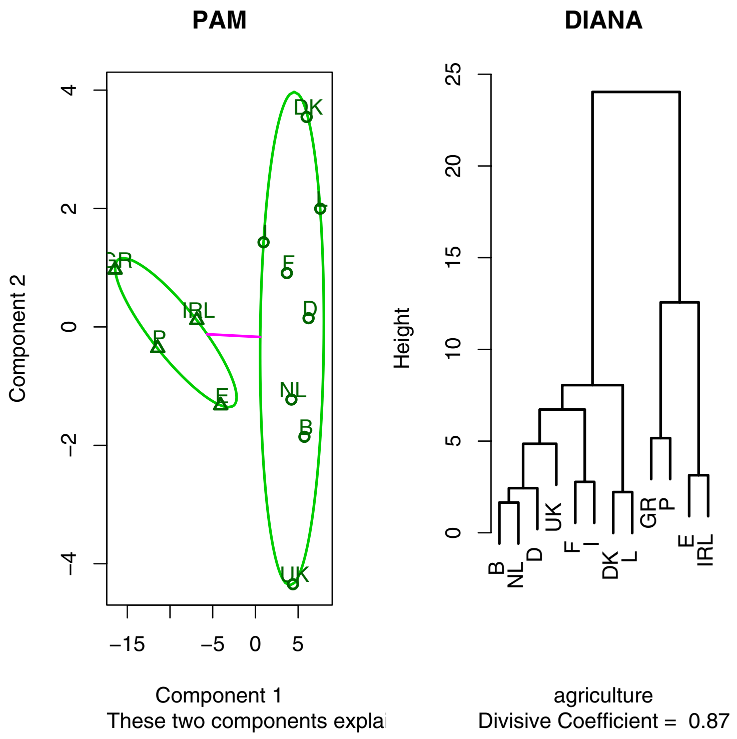


Figure 13.1. Partitioning versus hierarchical clustering. The `agriculture` data set from the package cluster contains two variables (Gross National Product per capita and percentage of the population working in agriculture) for each country belonging to the European Union in 1993. The countries were clustered by two algorithms from the package: (i) `pam` with two clusters and (ii) `diana`. The results are visualized as a `clusplot` for `pam` and a dendrogram for `diana`.

```
     size max_diss av_diss diameter separation
[1,]    8     5.42    2.89     8.05       5.73
[2,]    4     7.43    4.30    12.57       5.73
```

```
> hier <- diana(agriculture)
```

```
> par(mfrow = c(1, 2))
> plot(part, which.plots = 1, labels = 3, col.clus = 3,
+     lwd = 2, main = "PAM")
> plot(hier, which.plots = 2, lwd = 2, main = "DIANA")
```

Package	Functions	Description
cclust		Convex clustering methods
class	`SOM`	Self-organizing maps
cluster	`agnes`	AGglomerative NESting
	`clara`	Clustering LARge Applications
	`diana`	DIvisive ANAlysis
	`fanny`	Fuzzy Analysis
	`mona`	MONothetic Analysis
	`pam`	Partitioning Around Medoids
e1071	`bclust`	Bagged clustering
	`cmeans`	Fuzzy C-means clustering
flexmix		Flexible mixture modeling
fpc		Fixed point clusters, clusterwise regression and discriminant plots
hopach	`hopach`, `boothopach`	Hierarchical Ordered Partitioning and Collapsing Hybrid
mclust		Model-based cluster analysis
stats	`hclust`, `cophenetic`	Hierarchical clustering
	`heatmap`	Heatmaps with row and column dendrograms
	`kmeans`	k-means

Table 13.1. R functions and packages for cluster analysis (CRAN, Bioconductor).

13.2.4 Visualizing clustering results

Chapter 10 describes a variety of useful methods for visualizing gene expression data. The function `heatmap`, for example, implements the plot employed by Eisen et al. (1998) to visualize the $J \times I$ data matrix with rows and columns ordered by separate applications of their Cluster algorithm to both genes and arrays. Figure 13.2 shows an example of such a heat map. Heat maps can also be made of dissimilarity matrices (Figure 13.6 and Chapter 10), which are particularly useful when clustering patterns might not be easily visible in the data matrix, as with absolute correlation distance (van der Laan and Pollard, 2003).

As we see in Figures 13.2 and 13.1, there appears to be two clusters, one with four countries and another with eight. All visualizations make that reasonably obvious, although in different ways.

```
> heatmap(as.matrix(t(agriculture)), Rowv = NA,
+     labRow = c("GNP", "% in Agriculture"), cexRow = 1,
+     xlab = "Country")
```

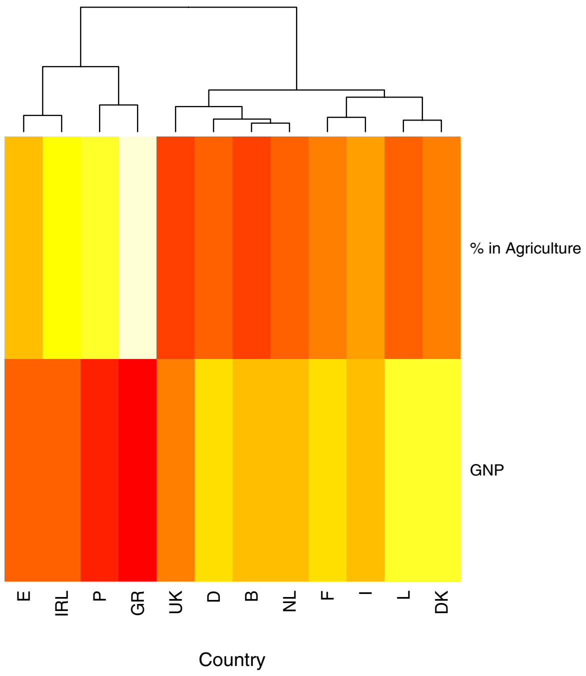


Figure 13.2. Heatmap for hierarchical clustering of the countries in the `agriculture` data set. The function `hclust` produces a dendrogram that is equivalent to that produced by `diana` with left and right children swapped at several nodes. Note that the ordering of countries in the `diana` tree depends a great deal on their order in the input data set, so that permuting the rows before running the algorithm will produce a different tree. The `hopach` (and to a lesser degree the `hclust` tree) is not sensitive to the initial order of the data.

13.2.5 Statistical issues in clustering

Exploratory techniques are capable of identifying interesting patterns in data, but they do not inherently lend themselves to statistical inference. The ability to assess reliability in an experiment is particularly crucial with the high dimensional data structures and relatively small samples presented by genomic experiments (Getz G., 2000; Hughes et al., 2000; Lockhart and Winzeler, 2000). Both jackknife (K.Y. et al., 2001) and bootstrap (Kerr and Churchill, 2001; van der Laan and Bryan, 2001) approaches have been used to perform statistical inference with gene expression data. van der Laan and Bryan (2001) present a statistical framework for clustering genes, where the clustering parameter θ is defined as a deterministic function $S(P)$ applied to the data generating distribution P. The parameter $\theta = S(P)$ is estimated by the observed sample subset $S(P_I)$, where the empirical distribution P_I is substituted for P. Most currently employed clustering methods fit into this framework, as they need only be deterministic functions of the empirical distribution. The authors also establish consistency of the clustering result under the assumption that $I/\log[J(I)] \to \infty$ (for a sample of I J-dimensional vectors), and asymptotic validity of the bootstrap in this context.

An interesting approach to clustering samples is to first cluster the genes and then cluster the samples using only the gene cluster profiles, such as medoids or means (Pollard and van der Laan, 2002b). In this way, the dimension of the data is reduced to the number of gene clusters so that the multiplicity problem for comparing subpopulations of samples is much less. Gene cluster profiles (particularly medoids) are very stable and hence the comparison of samples will not be affected by a few outlier genes [see also Nevins et al. (2003)]. Pollard and van der Laan (2002b) generalize the statistical framework proposed in van der Laan and Bryan (2001) to any clustering parameter $S(P)$, including algorithms that involve clustering both genes and samples.

13.2.6 Bootstrapping a cluster analysis

Though the clustering parameter $\theta = S(P)$ might represent an interesting clustering pattern in the true data generating distribution/population, when applied to empirical data P_I, it is likely to find patterns due to noise. To deal with this issue, one needs methods for assessing the variability of $\theta_I = S(P_I)$. One also needs to be able to test if certain components of θ_I are significantly different from the value of these components in a specified null experiment. Note that θ_I and P_I depend on the sample size I.

To assess the variability of the estimator θ_I, we propose to use the bootstrap. The idea of the bootstrap method is to estimate the distribution of θ_I with the distribution of $\theta_I^* = S(P_I^*)$, where P_I^* is the empirical distribution based on an *i.i.d.* bootstrap sample [i.e., a sample of I *i.i.d.* observations X_i^* $(i = 1, \ldots, I)$ from P_I]. The distribution of θ_I^* is obtained by applying the rule S to P_I^*, from each of B bootstrap samples, keeping track of parameters of interest. The distribution of a parameter is approximated by its empirical distribution over the B samples. There are several methods for generating bootstrap samples.

- **Non-parametric:** Resample I arrays with replacement.
- **Smoothed non-parametric:** Modify non-parametric bootstrap sampling with one of a variety of methods (e.g., Bayesian bootstrap or convex pseudo-data) for producing a smoother distribution.
- **Parametric:** Fit a model (e.g., multivariate normal, mixture of multivariate normals) and generate observations from the fitted distribution.

The non-parametric bootstrap avoids distributional assumptions about the parameter of interest. However, if the model assumptions are appropriate, or have little effect on the estimated distribution of θ_I, the parametric bootstrap might perform better.

13.2.7 Number of clusters

Consider a series of proposed clustering results. With a partitioning algorithm, these may consist of applying the clustering routine with $k = 2, 3, \ldots, K$ clusters, where K is a user-specified upper bound on the number of clusters. With a hierarchical algorithm the series may correspond to levels of the tree. With both types of methods, identifying cluster labels requires choosing the number of clusters. From a formal point of view, the question "How many clusters are there?" is essentially equivalent to asking "Which parameter is correct?" as each k defines a new parameter of the data generating distribution in the non-parametric model for P. Thus, selecting the correct number of clusters requires user input and typically there is no single right answer. Having said this, one is free to come up with a criterion for selecting the number of clusters, just as one might have an argument to prefer a mean above a median as location parameter. This criterion need not be the same as the criterion used to identify the clusters in the algorithm.

Overview of methods for selecting the number of clusters. Currently available methods for selecting the number of significant clusters include direct methods and testing methods. Direct methods consist of optimizing a criterion, such as functions of the within and between cluster sums of squares (Milligan and Cooper, 1985), occurrences of phase transitions in simulated annealing (Rose et al., 1990), likelihood ratios (Scott and Simmons, 1971), or average silhouette (Kaufman and Rousseeuw, 1990). The method of maximizing average silhouette is advantageous because it can be used with any clustering routine and any dissimilarity metric. A disadvantage of average silhouette is that, like many criteria for selecting the number of clusters, it measures the global clustering structure only. Testing methods take a different approach, assessing evidence against a specific null hypothesis. Examples of testing methods that have been used with gene expression data are the gap statistic (Tibshirani et al., 2000), the weighted average discrepant pairs (WADP) method (Bittner et al., 2000), a variety of permutation methods (Bittner et al., 2000; Hughes et al., 2000), and Clest (Fridlyand and Dudoit, 2001). Because they typically involve resampling, testing methods are computationally much more expensive than direct methods.

Median Split Silhouette. Median split silhouette (MSS) is a new direct method for selecting the number of clusters with either partitioning or hierarchical clustering algorithms (Pollard and van der Laan, 2002a). This method was motivated by the problem of finding relatively small, possibly nested clusters in the presence of larger clusters (Figure 13.3). It is frequently this finer structure that is of interest biologically, but most methods find only the global structure. The key idea is to evaluate how well the elements in a cluster belong together by applying a chosen clustering algorithm to the elements in that cluster alone (ignoring the other clusters) and then evaluating average silhouette after the split to determine the homogeneity of the parent cluster. We first define silhouettes and then describe how to use them in the MSS criterion.

Suppose we are clustering genes. The silhouette for a given gene is calculated as follows. For each gene j, calculate the average dissimilarity a_j of gene j with other genes in its cluster. For each gene j and each cluster l to which it does not belong, calculate the average dissimilarity b_{jl} of gene j with the members of cluster l. Let $b_j = \min_l b_{jl}$. The silhouette of gene j is defined by the formula: $S_j = (b_j - a_j)/\max(a_j, b_j)$. Heuristically, the silhouette measures how well matched an object is to the other objects in its own cluster versus how well matched it would be if it were moved to the next closest cluster. Note that the largest possible silhouette is 1, which occurs only if there is no dissimilarity within gene j's cluster (i.e., $a_j = 0$). A silhouette near 0 indicates that a gene lies between two clusters, and a silhouette near -1 means that the gene is very similar to elements in the neighboring cluster and hence is probably in the wrong cluster.

For a clustering result with k clusters, split each cluster into two or more clusters (the number of which can be determined, for example, by maximizing average silhouette). Each gene has a new silhouette after the split, which is computed relative to only those genes with which it shares a parent. We call the median of these for each parent cluster the split silhouette, SS_i, for $i = 1, 2, \ldots, k$, which is low if the cluster was homogeneous and should not have been split. $MSS(k) = median(SS_1, \ldots, SS_k)$ is a measure of the overall homogeneity of the clusters in the clustering result with k clusters. We advocate choosing the number of clusters which minimizes MSS. Note that all uses of median can be replaced with mean for a more sensitive, less robust criterion.

The following example of a data set with nested clusters demonstrates that MSS and average silhouette can identify different numbers of clusters. The data are generated by simulating a $J = 240$ dimensional vector consisting of eight groups of thirty normally distributed variables with the following means: $\mu \in (1, 2, 5, 6, 14, 15, 18, 19)$. The variables are uncorrelated with common standard deviation $\sigma = 0.5$. A sample of $I = 25$ is generated and the Euclidean distance computed.

```
> mu <- c(1, 2, 5, 6, 14, 15, 18, 19)
> X <- matrix(rnorm(240 * 25, 0, 0.5), nrow = 240,
+     ncol = 25)
> step <- 240/length(mu)
> for (m in 1:length(mu)) X[((m - 1) * step + 1):(m *
+     step), ] <- X[((m - 1) * step + 1):(m * step),
+      ] + mu[m]
> D <- dist(X, method = "euclidean")
```

Next, we check the number of clusters k identified by average silhouette with the function `silcheck` and by MSS with the function `msscheck`, both provided in the package hopach. These return a vector with the number of clusters optimizing the corresponding criterion in the first entry and the value of the criterion in the second.

```
> library("hopach")
> k.sil <- silcheck(X)[1]
> k.mss <- msscheck(as.matrix(D))[1]
> pam.sil <- pam(X, k.sil)
> pam.mss <- pam(X, k.mss)
```

We plot the distance matrix with the $J = 240$ variables ordered according to their `pam` cluster labels with each choice of k. We mark the two sets of cluster boundaries on each axis.

```
> image(1:240, 1:240, as.matrix(D)[order(pam.sil$clust),
+     order(pam.mss$clust)], col = topo.colors(80),
+     xlab = paste("Silhouette (k=", k.sil, ")", sep = ""),
+     ylab = paste("MSS (k=", k.mss, ")", sep = ""),
+     main = "PAM Clusters: Comparison of Two Criteria",
+     sub = "Ordered Euclidean Distance Matrix")
> abline(v = cumsum(pam.sil$clusinfo[, 1]), lty = 2, lwd = 2)
> abline(h = cumsum(pam.mss$clusinfo[, 1]), lty = 3, lwd = 2)
```

We have previously reported simulation results for MSS on a variety of data sets and relative to other direct methods (Pollard and van der Laan, 2002a). We refer the reader to the figures in that manuscript for further illustration of the MSS methodology.

HOPACH algorithm. The R package hopach implements the Hierarchical Ordered Partitioning and Collapsing Hybrid (HOPACH) algorithm for building a hierarchical tree of clusters (Figure 13.4). At each node, a cluster is split into two or more smaller clusters with an enforced ordering of the clusters. Collapsing steps uniting the two closest clusters into one cluster are used to correct for errors made in the partitioning steps. The `hopach` function uses the median split silhouette criterion to automatically choose (i) the number of children at each node, (ii) which clusters to collapse, and (iii) the main clusters (pruning the tree to produce a partition of homogeneous clusters). We describe the method as applied to clustering genes in an expression data set X, but the algorithm can be used much more generally. We will use the notation $PAM(X, k, d)$ for the PAM algorithm applied to the data X with k clusters ($k < 10$ for computational convenience) and dissimilarity d.

Initial level: Begin with all elements at the root node.

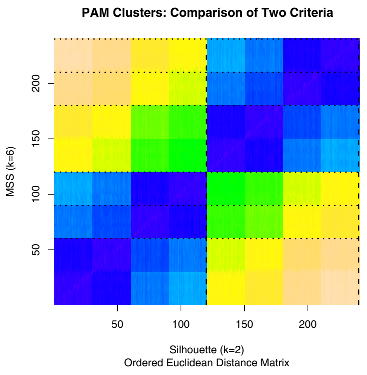


Figure 13.3. Median split silhouette (MSS) versus average silhouette. The Euclidean distance matrix from a data set with nested clusters is plotted here with the variables ordered according to their cluster labels. Blue corresponds to small and peach to large dissimilarity. The nested structure of the data is visible. Lines mark the boundaries of the PAM clusters, with the number of clusters k determined either by minimizing MSS or maximizing average silhouette. Average silhouette is more robust and therefore typically identifies fewer clusters.

1. *Partition*: Compute $PAM(X, k, d)$ and $MSS(k)$ for $k = 2, \ldots, 9$. Accept the minimizer $k1$ of $MSS(k)$ and corresponding partition $PAM(x, k1, d)$ as the first level of the tree. Also compute $MSS(1)$. If $MSS(1) < MSS(k1)$, print a warning message about the homogeneity of the data.
2. *Order*: Define the distance between a pair of clusters (i.e., linkage) as the dissimilarity between the corresponding medoids. If $k1 = 2$, then the ordering does not matter. If $k1 > 2$, then order the clusters by (a) building a hierarchical tree from the $k1$ medoids or (b) maximizing the empirical correlation between distance $j - i$ in the list and the corresponding dissimilarity $d(i, j)$ across all pairs (i, j) with $i < j$ with the function `correlationordering`.
3. *Collapse*: There is no collapsing at the first level of the tree.

Next level: For each cluster in the previous level of the tree, carry out the following procedure.
1. *Partition*: Apply PAM with $k = 1, \ldots, 9$ as in level 1, and select the minimizer of $MSS(k)$ and corresponding PAM partitioning.
2. *Order*: Order the child clusters by their dissimilarity with the medoid of the cluster next to the parent cluster in the previous level.
3. *Collapse*: Beginning with the closest pair of medoids (which may be on different branches of the tree), collapse the two clusters if doing so improves MSS. Continue collapsing until a collapse is rejected (or until all pairs of medoids are considered).The medoid of the new cluster can be chosen in a

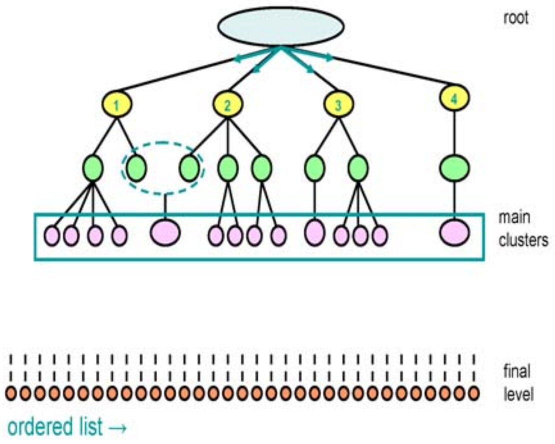


Figure 13.4. The HOPACH hierarchical tree unfolding through the steps of the clustering algorithm. First, the root node is partitioned and the children in the next level are ordered deterministically using the same dissimilarity matrix that is used for clustering. Next, each of these nodes is partitioned and its children are ordered. Before the next partitioning step, collapsing steps merge any similar clusters. The process is iterated until the main clusters are identified. Below the main clusters, the algorithm is run down without collapsing to produce a final ordered list.

variety of ways, including the nearest neighbor of the average of the two corresponding medoids.
Iterate: Repeat until each node contains no more than 2 genes or a maximum number of levels is reached (for computational reasons the limit is 16 levels in the current implementation).

Main clusters: The value of MSS at each level of the tree can be used to identify the level below which cluster homogeneity improves no further. The partition defined by the pruned tree at the selected level is identified as the main clusters.

The path that each gene follows through the HOPACH tree is encoded in a label with one digit for each level in the tree. Because we restrict the number of child clusters at each node to be less than ten, only a single digit is needed for each level. Zero denotes a cluster that is not split. A typical label of a gene at level 3 in the tree looks like 152, meaning that the gene is in the second child cluster of the fifth child cluster of the first cluster from level 1. In order to look at the cluster structure for level l of the tree, simply truncate the final cluster labels to l digits. Chapter 20 provides some relevant concepts and notation regarding paths and path labelling in graphs.

We refer the reader to van der Laan and Pollard (2003) for a comparison of HOPACH with other clustering algorithms. In simulations and real data analyses, we show that `hopach` is better able to identify small clusters and to produce a sensible final ordering of the elements than other algorithms discussed here.

13.3 Application: renal cell cancer

The renal cell cancer data package kidpack contains expression measures for 4224 genes and 74 patients. The tumor samples (labeled green) are compared to a common reference sample (labeled red). Log ratios measure expression in the control relative to each tumor.

13.3.1 Gene selection

To load the kidpack data set:

```
> library("kidpack")
> data(eset, package = "kidpack")
> data(cloneanno, package = "kidpack")
```

First, select a subset of interesting genes. Such a subset can be chosen in many ways, for example with the functions in the genefilter and multtest packages. For this analysis, we will simply take all genes (416 total) with log ratios greater than 3-fold in at least half of the arrays. This means that we are focusing on genes that are *suppressed* in the kidney tumor samples relative to the control sample. One would typically use a less arbitrary subset rule. We use the IMAGE ID (Lennon et al., 1996) as the gene name, adding the character "B" to the name of the second copy of any IMAGE ID.

```
> library("genefilter")
> ff <- pOverA(0.5, log10(3))
> subset <- genefilter(abs(exprs(eset)), filterfun(ff))
> kidney <- exprs(eset)[subset, ]
> dim(kidney)
> gene.names <- cloneanno[subset, "imageid"]
> is.dup <- duplicated(gene.names)
> gene.names[is.dup] <- paste(gene.names[is.dup],
+     "B", sep = "")
> rownames(kidney) <- gene.names
> colnames(kidney) <- paste("Sample", 1:ncol(kidney),
+     sep = "")
```

13.3.2 HOPACH clustering of genes

It is useful to compute the dissimilarity matrix before running `hopach`, because the dissimilarity matrix may be needed later in the analysis. The cosine-angle dissimilarity defined in Chapter 12 is often a good choice for clustering genes.

```
> gene.dist <- distancematrix(kidney, d = "cosangle")
> dim(gene.dist)

[1] 416 416
```

Now, run `hopach` to cluster the genes. The algorithm will take some time to run.

```
> gene.hobj <- hopach(kidney, dmat = gene.dist)

> gene.hobj$clust$k

[1] 84

> table(gene.hobj$clust$sizes)

  1   2   3   4   5   7   9  18  24  42  80 112
 52   8  13   3   1   1   1   1   1   1   1   1

> gene.hobj$clust$labels[1:5]

[1] 22200 22200 21300 23200 43000
```

The `hopach` algorithm identifies 84 gene clusters. Many of the clusters are 1 to 4 genes, though some are much larger. The cluster labels show the relationships between the clusters and how they evolved in the first few levels of the tree. Next, we examine how close clones that represent the same gene (i.e., genes with a ”B” in their name) are to one another in the HOPACH final ordering.

```
> gn.ord <- gene.names[gene.hobj$fin$ord]
> Bs <- grep("B", gn.ord)
> spaces <- NULL
> for (b in Bs) {
+     name <- unlist(strsplit(gene.names[gene.hobj$fin$ord][b],
+         "B"))
+     spaces <- c(spaces, diff(grep(name, gn.ord)))
+ }
> table(spaces)

spaces
  1   4   6  14  17  35  53  54  72  90 129
  5   1   1   1   1   1   1   1   1   1   1
```

Five of the fifteen pairs of replicate clones appear next to each other, and all of them appear closer to one another than expected for a random pair of clones.

13.3.3 Comparison with PAM

The `hopach` clustering results can be compared to simply applying PAM with the choice of k that maximizes average silhouette (using the function `silcheck`).

```
> bestk <- silcheck(dissvector(gene.dist), diss = TRUE)[1]
> pamobj <- pam(dissvector(gene.dist), k = bestk,
+     diss = TRUE)
> round(pamobj$clusinfo, 2)
```

```
     size max_diss av_diss diameter separation
[1,]   68     0.96    0.64     1.10       0.39
[2,]  348     0.94    0.45     1.21       0.39
```

While `hopach` identifies 84 clusters of median size 1, `pam` identifies 2 larger clusters. This result is typical in the sense that `hopach` tends to be more aggressive at finding small clusters, whereas `pam` is more robust and therefore only identifies the global patterns (i.e., fewer, larger clusters).

13.3.4 Bootstrap resampling

For each gene and each `hopach` cluster we can compute the proportion of bootstrap data sets where the gene is in the cluster. These are estimates of the membership of the gene in each cluster and can be considered as a form of fuzzy clustering.

```
> bobj <- boothopach(kidney, gene.hobj, B = 100)
```

The argument `B` controls the number of bootstrap resampled data sets used. The default value is `B`= 1000, which represents a balance between precision and speed. For this example, we use `B`= 100 since larger values have much longer run times. The `bootplot` function makes a barplot of the bootstrap reappearance proportions (see Figure 13.5).

```
> bootplot(bobj, gene.hobj, ord = "bootp", main = "Renal Cell Cancer",
+     showclusters = FALSE)
```

13.3.5 HOPACH clustering of arrays

The HOPACH algorithm can also be applied to cluster samples (i.e., arrays), based on their expression profiles across genes. This analysis method differs from classification, which uses knowledge of class labels associated with each sample (i.e., array). Euclidean distance may be a good choice for

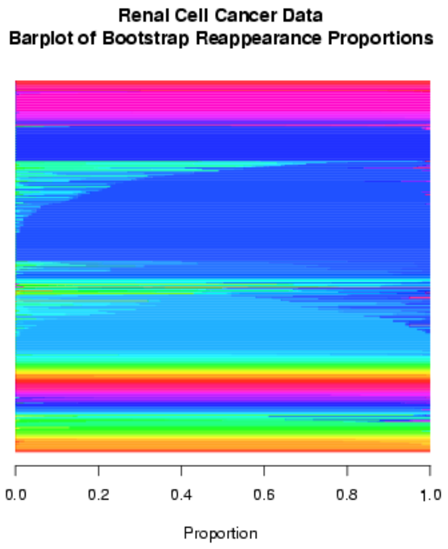


Figure 13.5. The `bootplot` function makes a barplot of the bootstrap reappearance proportions for each gene and each cluster. These proportions can be viewed as fuzzy cluster memberships. Every cluster is represented by a different color. The genes are ordered by `hopach` cluster, and then by bootstrap estimated membership within cluster and plotted on the vertical axis. Each gene is represented by a very narrow horizontal bar. The length of this bar that is each color is proportional to the percentage of bootstrap samples in which that gene appeared in the cluster represented by that color. If the bar is all or mostly one color, then the gene is estimated to belong strongly to that cluster. If the bar is many colors, the gene has fuzzy membership in all these clusters. The continuity of colors across the genes indicates that nearby clusters are more likely to "swap" genes than more distant clusters.

clustering arrays, because it measures differences in magnitude, which is often what we are interested in detecting when comparing the expression profiles for different samples. A comparison of magnitude is valid, because we expect the data from different arrays to be on the same scale after normalization has been performed.

```
> array.hobj <- hopach(t(kidney), d = "euclid")

> array.hobj$clust$k

[1] 51
```

51 array clusters are identified. The function `dplot` can be used to visualize the ordered dissimilarity matrix corresponding with the HOPACH tree's final level. Clusters of similar arrays will appear as blocks on the diagonal of the matrix (Figure 13.6). We can label the arrays from patients with

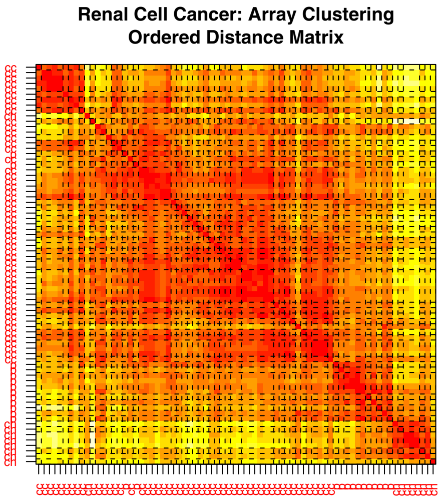


Figure 13.6. HOPACH clustering of patients with Euclidean distance. Patients are ordered according to the final level of the tree. Red corresponds to small distance and white to large distance. Dotted lines indicate the clusters boundaries in the level of the tree with minimum MSS. Many patients cluster alone, but there are several small groups of very similar patients. The ordering of patients by `hopach` coincides well with tumor type. cc: clear cell, p: papillary, ch: chromophobe.

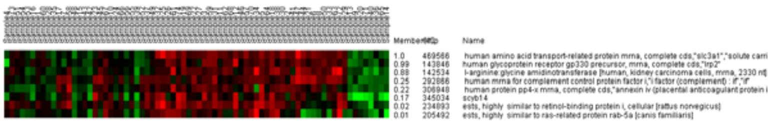
different tumor types (clear cell, papillary, and chromophobe) and examine how these labels correspond with the clusters.

```
> tumortype <- unlist(strsplit(phenoData(eset)$type, "RCC"))
> dplot(distancematrix(t(kidney), d = "euclid"), array.hobj,
+    labels = tumortype, main = "Renal Cell Cancer: Array Clustering")
```

13.3.6 Output files

Gene clustering and bootstrap results table. The `makeoutput` function is used to write a tab delimited text file that can be opened in a spreadsheet application or text editor. The file will contain the `hopach` clustering results, plus possibly the corresponding bootstrap results, if these are provided. The argument `gene.names` can be used to insert additional gene annotation, in this case accession numbers.

```
> gene.acc <- cloneanno[subset, "AccNumber"]
> makeoutput(kidney, gene.hobj, bobj, file = "kidney.out",
+     gene.names = gene.acc)
```



Member[illegible]		Name
1.0	469566	human amino acid transport-related protein mrna, complete cds,"slc3a1","solute carri
0.99	143846	human glycoprotein receptor gp330 precursor, mrna, complete cds,"lrp2"
0.88	142534	l-arginine:glycine amidinotransferase [human, kidney carcinoma cells, mrna, 2330 nt]
0.25	292866	human mrna for complement control protein factor i,"i factor (complement) : if","if"
0.22	306948	human protein pp4-x mrna, complete cds,"annexin iv (placental anticoagulant protein i
0.17	345034	scyb14
0.02	234893	ests, highly similar to retinol-binding protein i, cellular [rattus norvegicus]
0.01	205492	ests, highly similar to ras-related protein rab-5a [canis familiaris]

Figure 13.7. MapleTree zoom view of a single cluster in the kidney data. Genes are ordered according to their bootstrap membership. Red represents overexpression in control relative to tumor samples, and green is the opposite.

Bootstrap fuzzy clustering in MapleTree. MapleTree (Lisa Simirenko) is an open source, cross-platform, visualization tool for graphical browsing of results of cluster analyses. The software can be found at SourceForge. The `boot2fuzzy` function takes the gene expression data, plus corresponding `hopach` clustering output and bootstrap resampling output, and writes the (`.cdt, .fct, and .mb`) files needed to view these fuzzy clustering results in MapleTree.

```
> gene.desc <- cloneanno[subset, "description"]
> boot2fuzzy(kidney, bobj, gene.hobj, array.hobj,
+     file = "kidneyFzy", gene.names = gene.desc)
```

The three generated files can be opened in MapleTree by going to the `Load` menu and then `Fuzzy Clustering Data`. The heat map contains only the medoid genes (cluster profiles). Double clicking on a medoid opens a zoom window for that cluster, with a heat map of all genes ordered by their bootstrap estimated memberships in that cluster, with the highest membership first. Figure 13.7 contains the zoom window for gene cluster 15. The medoid and two other genes have high bootstrap reappearance probabilities.

HOPACH hierarchical clustering in MapleTree. The MapleTree software can also be used to view HOPACH hierarchical clustering results. The `hopach2tree` function takes the gene expression data, plus corresponding `hopach` clustering output for genes or arrays, and writes the (`.cdt`, `.gtr`, and optionally `.atr`) files needed to view these hierarchical clustering results in MapleTree. These files can also be opened in other viewers such as TreeView (Michael Eisen), jtreeview (Alok Saldanha), and GeneXPress (Eran Segal).

```
> hopach2tree(kidney, file = "kidneyTree", hopach.genes = gene.hobj,
+     hopach.arrays = array.hobj, dist.genes = gene.dist,
+     gene.names = gene.desc)
```

The `hopach2tree` function writes up to three text files. A `.cdt` file is always produced. When `hopach.genes`is not `NULL`, a `.gtr` is produced, and gene clustering results can be viewed, including ordering the genes in the heat map according to the final level of the `hopach` tree and drawing the dendrogram for hierarchical gene clustering. Similarly, when `hopach.arrays` is not `NULL`, an `.atr` file is produced, and array clustering results can

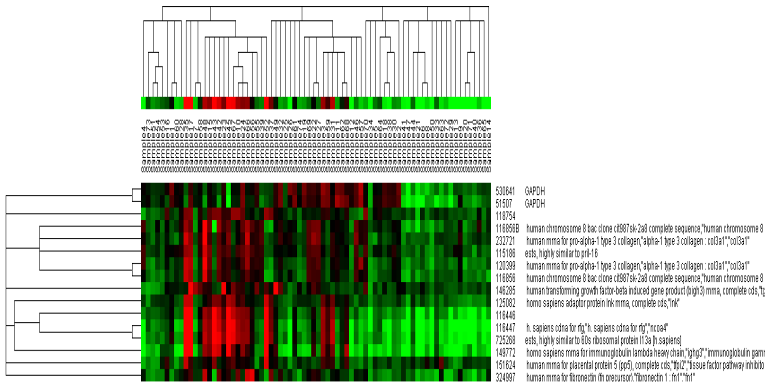


Figure 13.8. MapleTree HOPACH hierarchical view of a section of the gene tree and all of the array tree. Red represents overexpression in control relative to tumor samples, and green is the opposite. Two copies of the clone with I.M.A.G.E. ID 469566 appear near each other in the tree.

be viewed. These files can be opened in MapleTree by going to the `Load` menu and then `HOPACH Clustering Data`. By clicking on branches of the tree, a zoom window with gene names for that part of the tree is opened. Figure 13.8 illustrates this view for a section of the the kidney data in MapleTree.

13.4 Conclusion

This chapter has provided an overview of clustering methods in R, including the new hopach package. The variety of available dissimilarity measures, algorithms and criteria for choosing the number of clusters give the data analyst the ability to choose a clustering parameter that meets particular scientific goals. Viewing the output of a clustering algorithm as an estimate of this clustering parameter allows one to assess the reliability and repeatability of the observed clustering results. This kind of statistical inference is particularly important in the context of analyzing high-dimensional genomic data sets. Visualization tools, including data and distance matrix heatmaps, help summarize clustering results.