9. Clustering

Clustering analysis is the grouping of individuals, elements or cases in a population or dataset to discover structure in the data. In some sense, we would like to have the cases within a group to be close or similar to one another, but dissimilar from cases in other groups.

Clustering is fundamentally a collection methods of data exploration. They are *unsupervised learning* algorithms, but can also be applied in the supervised way provided that there exists class label information. When they are used in the unsupervised way as usual, generated – if these emerge - groups called clusters are named and their ”meaning” defined by a user. Clusters can then be employed as classes for further supervised machine learning. Clustering produces a *partition* of the dataset – its division into mutually non-overlapping groups.

To compute clusters we apply distance or similarity measures in order to decide which cases are assigned in each cluster and how clusters are made.

Dividing *N* data cases into *K* clusters (groups) gives a huge number of possible partitions, which is expressed in the form of Stirling number:

$$\frac{1}{K!}\sum_{i=1}^{K}(-1)^{K-i}\binom{K}{i}i^{N}$$

(The sum is divided by *K*!, because the order does not matter to clusters, i.e., combinations are counted, not permutations.)

9.1 Partition-based or objective function-based clustering

Next the basic version of K-means algorithm is described, where the number K of clusters has to first be given. The algorithm begins by taking randomly K cases to be initial values of cluster centers. Every other case is included in a cluster the center of which is the closest to the case. Then new cluster centers (means) also called prototypes are computed, and the process is continued iteratively.

There are N cases in set $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$. The aim is to find K clusters $\{C_1, \ldots, C_K\}$. There are many different objective functions, but we only present the simplest, minimum variance criterion coming from a clear and intuitive motivation: the *centers* or *prototypes* should be such that they minimize a dispersion of data around them. Having N cases and aiming at K clusters, we compute the sum of dispersion between cases and a set of centers $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_K$ as follows.

$$Q = \sum_{i=1}^{K} \sum_{k=1}^{N} u_{ik} \left\| \mathbf{x}_k - \mathbf{v}_i \right\|^2$$

Here the norm is Euclidean distance between data case $\mathbf{x}_k$ and center $\mathbf{v}_i$. The important part is the partition matrix **U**=[u_{ik}], i=1,2,…,K, k=1,2,…,N the role of which is to allocate the cases to the clusters. The entries of **U** are binary. Case k belongs to cluster i when u_{ik}=1, and does not belong if u_{ik}=0.

Every cluster is nontrivial, i.e., it does not include all cases and is nonempty:

$$0 < \sum_{k=1}^{N} u_{ik} < N, i = 1,2,..., K$$

Every case belongs to a single cluster:

$$\sum_{i=1}^{K} u_{ik} = 1, k = 1,2,\ldots, N$$

The family of binary partition matrices {**U**} satisfies these two conditions.

Formally, the optimization problem due to be solved is expressed in the following way.

$$\min_{\mathbf{v}_i}(Q), \mathbf{U} \in \{\mathbf{U}\}$$

K-means clustering algorithm

There are several approaches for this optimization. The most common is *K*-means.

(1) Let $\mathbf{v}_i$ be $i = 1,\ldots,K$ randomly selected cases (vectors) from set D.

(2) Repeat

    (2.1) Construct a partition matrix by assigning numeric values to **U** according to the following rule:

$$u_{ik} = \begin{cases} 1, \text{ if } d(\mathbf{x}_k, \mathbf{v}_i) < d(\mathbf{x}_k, \mathbf{v}_j), \forall j \neq i \\ 0, \text{ otherwise} \end{cases}$$

(2.2) Update the centers by computing the weighted mean, which involves the entries of **U**:

$$\mathbf{v}_i = \frac{\sum_{k=1}^{N} u_{ik}\mathbf{x}_k}{\sum_{k=1}^{N} u_{ik}}$$

until index *Q* (p. 275) stabilizes and does not change, or until the changes are negligible.

(In Matlab *K*-means clustering is made with ’kmeans’.)

K-means++ algorithm

Occasionally, randomly chosen initial cluster centers may represent poorly data cases in clustering, which may also achieve poor clustering results. A more effective intialization is in *K*-means++ algorithm, where initial centers are selected so that they are as far as possible from each other. Because of the more advanced intialization, *K*-means++ is also typically faster to run compared with *K*-means.

Let $d(\mathbf{x})$ denote the shortest distance from a data case $\mathbf{x}$ to the closest center that is already chosen.

(1) Choose an initial center $\mathbf{v}_1$ uniformly at random from all data cases.

(2) From the rest data, choose the next center $\mathbf{v}_i$ according to probability

$$\frac{(d(\mathbf{x}))^2}{\sum_{\mathbf{x}\in X}(d(\mathbf{x}))^2}$$

where *X* is the data set.

(3) Repeat step (2) until *K* centers are chosen.

(4) Then *K*-means is used to make the actual clustering.

(Nowadays, in Matlab *K*-means++ clustering is made for ’kmeans’ by default.)

There is another variant of *K*-means clustering called *K*-medoids which uses medians instead of means for cluster centers. This is reasonable if data may contain noise. In addition, medians represent actual data cases, while means are slightly ”artificial” which can be treated as an undesired phenomenon. (To be exact, median represents the centermost data case in increasing order only if there are an odd number of cases. Otherwise, it is equal to the average of two centermost data cases.) Its basic algorithm (not given here) may, however, be rather time consuming, since finding the minimum of its object function can be quite tedious.

9.2 Hierarchical clustering

A hierarchical tree is a nested set of partitions represented by a tree structure called *dendrogram* (Fig. 9.1). Sectioning a tree at a particular level produces a partition into *g* disjoint groups. If two groups are chosen from different partitions (the results of partitioning at different levels), then either the groups are disjoint or one group wholly contains the other. An example of a hierarchical classification is that of the animal kingdom. There are different measures, some to be presented later. On the basis of dendrograms, the number of clusters can be defined by a section at a selected level; those disjoint subsets including cases below the section are interpreted as clusters.

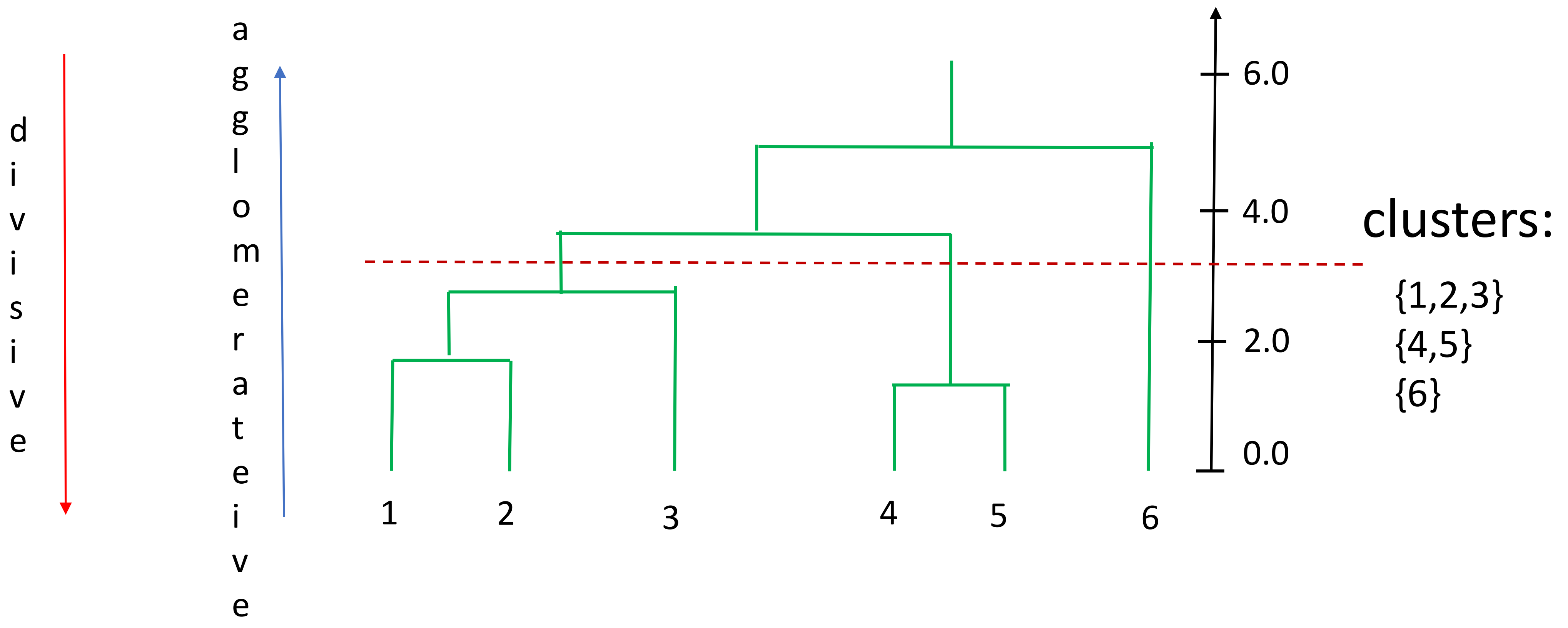


Fig. 9.1 Dendrogram: the vertical axis shows the distance or dissimilarity between two merged clusters, and the horizontal axis no measure, but just to show relations between clusters.

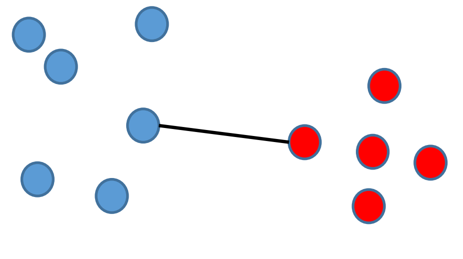
There are different algorithms for finding a hierarchical tree. An *agglomerative* algorithm (Fig. 9.1) begins with subclusters, each containing a single case, and at each stage merges the two most similar groups to form a new cluster, thus reducing the number of clusters by one. The algorithm proceeds until all the data fall within a single cluster. A *divisive* algorithm operates by successively splitting clusters, beginning with a single cluster, and continuing until there are *N clusters*, each of a single case. Generally, the divisive algorithms are computationally inefficient, except where most variables are binary. We do only present the former that are more frequently used and are easier to apply.

Single-link method

Single-link or nearest neighbor method (Fig. 9.2.(a)) is one of the oldest in cluster analysis. The distance or dissimilarity *d*(*A*,*B*) is based on the minimal distance between the cases or patterns belonging to *A* and *B*. It is computed as follows.

$$d(A,B) = \min_{\mathbf{x}\in A,\mathbf{y}\in B} d(\mathbf{x},\mathbf{y})$$

In essence, the distance is a very ”optimistic” mode, where we involve the nearest cases located in different clusters.



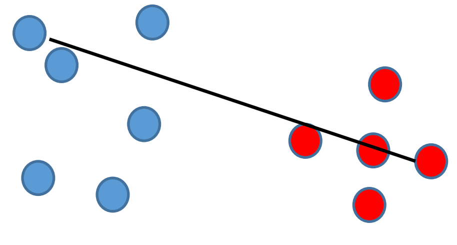


Fig. 9.2 Two clusters *A* and *B* linked with (a) the single-link and (b) complete-link method.

Complete-link or furthest neighbor method (Fig. 9.2.(b)) is the opposite to the former, since this is based on the furthest distance between two cases in two clusters.

$$d(A, B) = \max_{\mathbf{x} \in A, \mathbf{y} \in B} d(\mathbf{x}, \mathbf{y})$$

Group average link is contrast to the two previous approaches, in which the distance is determined on the basis of extreme values of the distance function, in this method we compute the average between the distances as computed between every pair of cases, with one case from each cluster.

$$d(A, B) = \frac{1}{|A||B|} \sum_{\mathbf{x} \in A, \mathbf{y} \in B} d(\mathbf{x}, \mathbf{y})$$

Obviously, the computations are more intensive, but they reflect a general tendency between the distances computed for single pairs of cases.

One can develop other ways of expressing the distance between clusters *A* and *B*. For example, one can compute the Hausdorff distance between two sets of cases in the following way.

$$d(A,B) = \max\{\max_{\mathbf{x}\in A}\min_{\mathbf{y}\in B} d(\mathbf{x},\mathbf{y}), \max_{\mathbf{y}\in B}\min_{\mathbf{x}\in A} d(\mathbf{x},\mathbf{y})\}$$

Sum-of-squares method

The *sum-of-squares method* is appropriate for the clustering of cases in Euclidean space. The aim is to minimize the total *within-group sum of squares*. *Ward's hierarchical clustering method* uses an agglomerative algorithm to compute a set of hierarchically nested partitions that can be represented by a dendrogram. However, the optimal sum-of-squares partitions for different numbers of groups are not necessarily hierarchically nested. Thus the algorithm is *suboptimal*.

At each stage of the algorithm, two groups producing the smallest increase in the total within-group sum of squares are amalgamated.

The updating formula for the dissimilarity or distance matrix is

$$d_{i+j,k} = \frac{n_k + n_i}{n_k + n_i + n_j} d_{ik} + \frac{n_k + n_j}{n_k + n_i + n_j} d_{jk} - \frac{n_k}{n_k + n_i + n_j} d_{ij}$$

where $d_{i+j,k}$ is the distance between the amalgamated groups *i*+*j* and group *k*, and n_i is the number of cases in group *i*. Initially, every group contains a single case and the element of the dissimilarity matrix, d_{ij}, is the squared Euclidean distance between the *i*th and *j*th case.

General agglomerative algorithm

Several agglomerative algorithms for producing hierarchical trees can be expressed as a special case of a single algorithm. The algorithms differ in the way that the dissimilarity between a cluster *k* and the cluster formed by joining *i* and *j* as

$$d_{i+j,k} = a_i d_{ik} + a_j d_{jk} + b d_{ij} + c \left| d_{ik} - d_{jk} \right|$$

where a_i, a_j, *b* and *c* are parameters that, if chosen appropriately, will give an agglomerative algorithm for implementing some of the more commonly applied techniques (Table 9.1).

Table 9.1 Special cases of the general agglomerative algorithm.

Method	a_i (a_j)	**b**	**c**
Single link	1/2	0	-1/2
Complete link	1/2	0	1/2
Centroid	$n_i/(n_i + n_j)$	$-n_i n_j/(n_i + n_j)^2$	0
Median	1/2	-1/4	0
Group average link	$n_i/(n_i + n_j)$	0	0
Ward's method	$(n_i + n_k)/(n_i + n_j + n_k)$	$- n_k/(n_i + n_j + n_k)$	0

Centroid distance defines the distance between two clusters to be the distance between the cluster means or centroids.

When a small cluster is joined to a larger one, the centroid of the result will be close to the centroid of the larger cluster. Sometimes this is disadvantageous. Median distance attempts to overcome this by defining the distance between two clusters to be the distance between the medians of the clusters.

Clustering algorithm

Let C_i and C_j be two clusters and function d measures the distance between them. The agglomerative algorithm is as follows.

(1) Let $C_i = \{\mathbf{x}_i\}$, when i=1,2,…,N.

(2) While there are more than 1 cluster left do

  (2.1) Let C_i and C_j be clusters that minimize distance $d(C_i,C_j)$ for all pairs of i and j.

  (2.2) $C_i = C_i \cup C_j$

  (2.3) Remove cluster C_j.

Time complexity of agglomerative clustering

The algorithm consists of N iterations, when there are N clusters at the beginning and 1 at the end. For iteration i it has to find pair from $N\text{-}i\text{+}1$ clusters. Their distance $d(C_i,C_j)$ can be computed in different ways, but for the first iteration all ways have to identify the closest pair. This takes $O(N^2)$ time. Thus the algorithm takes at least $O(N^2)$, but often more. This is the same for space complexity. The whole execution takes $O(N^3)$, but if the distances between clusters are stored in the heap data structure, the total time complexity is only $O(N^2 \log N)$. Thus, the algorithm is not suitable for large datasets.

Example

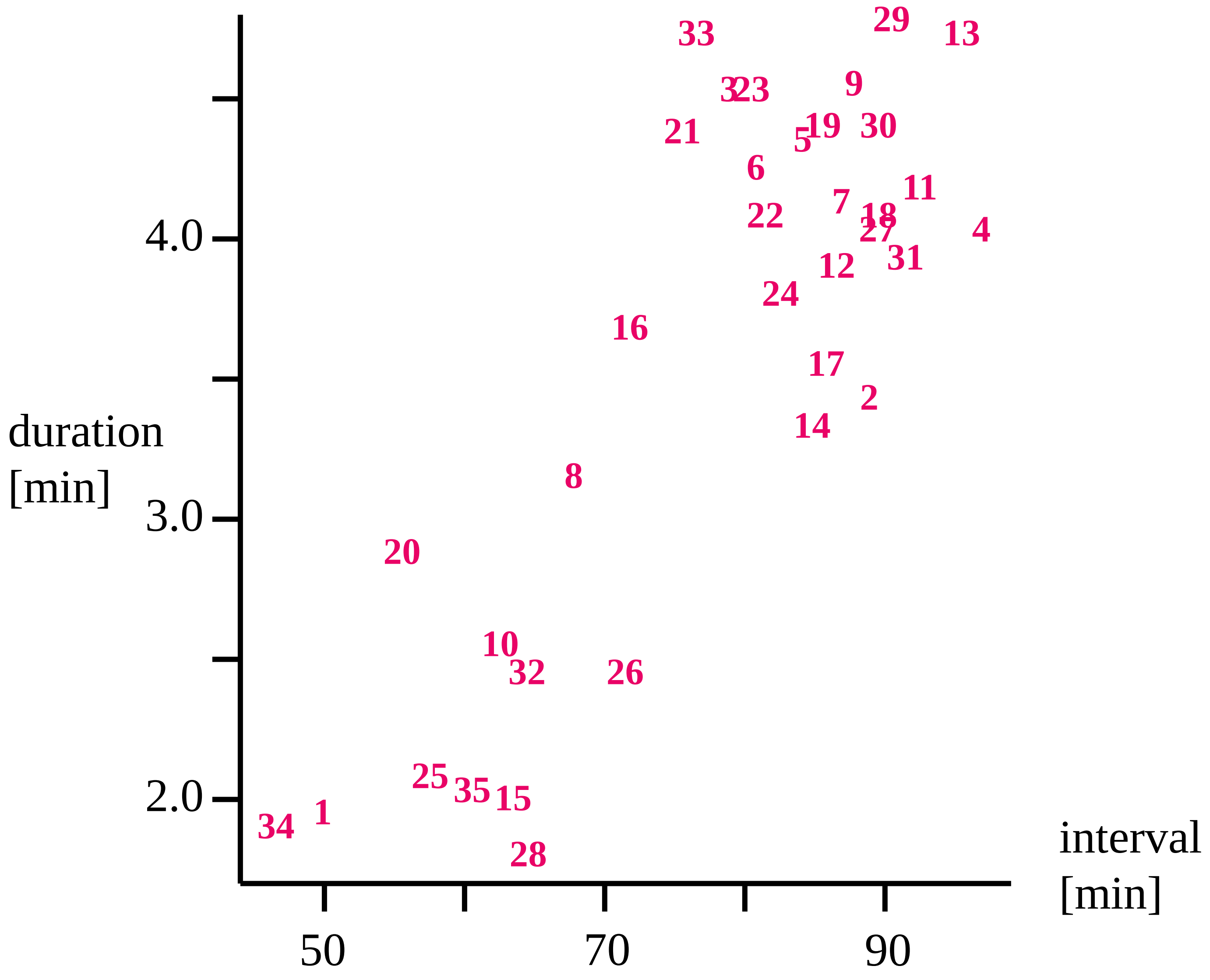


Fig. 9.3 Durations for eruptions of Geyser called Old Faifthfull in Yellowstone related to intervals of the eruptions.

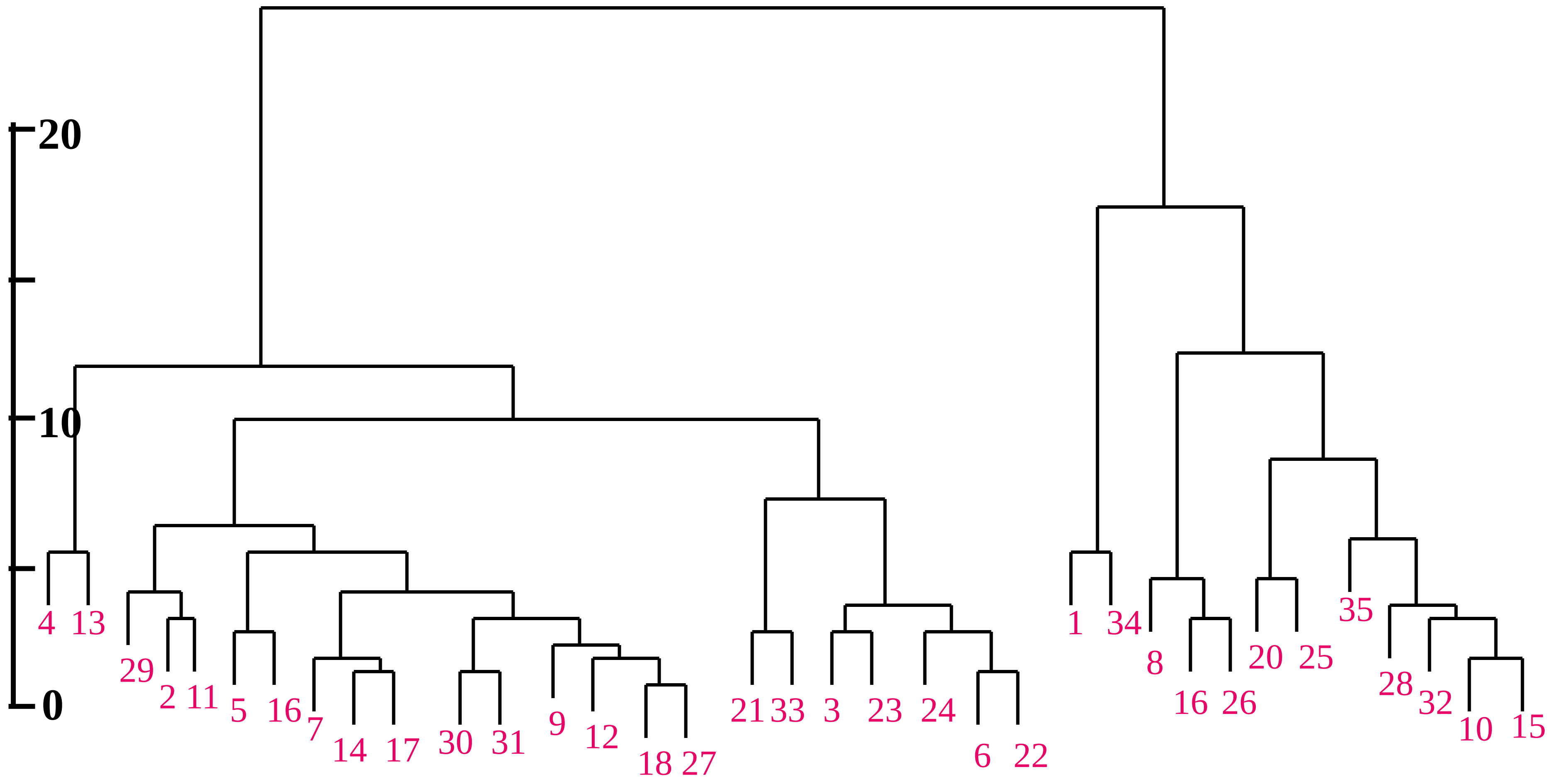


Fig. 9.4 Clustering from the data in Fig. 9.3 when clusters with the least increase of squared sums are joined (corresponding to *K*-means).

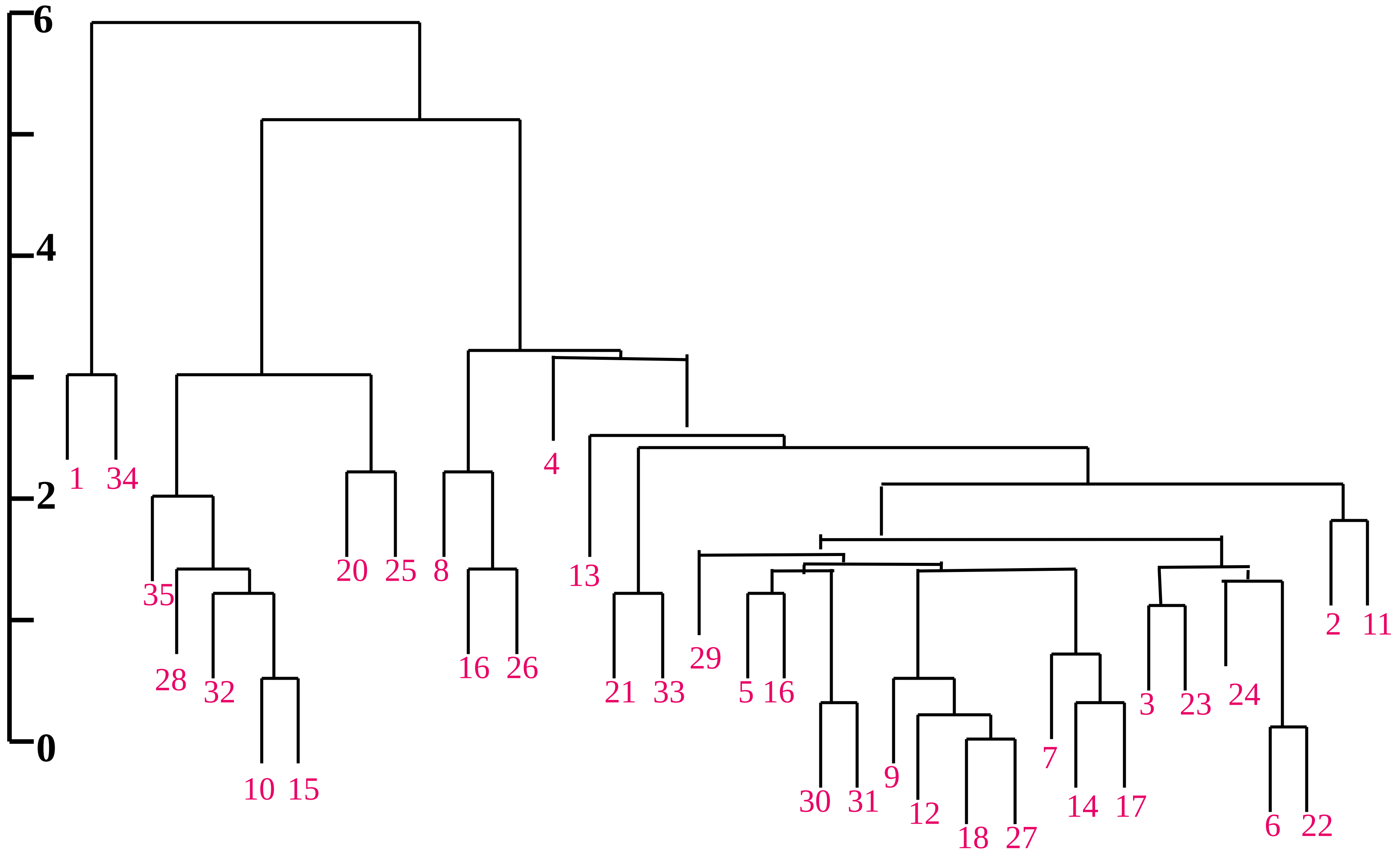


Fig. 9.5 The dendrogram given by the single link method.