17 *Hierarchical clustering*

HIERARCHICAL CLUSTERING

Flat clustering is efficient and conceptually simple, but as we saw in Chapter 16 it has a number of drawbacks. The algorithms introduced in Chapter 16 return a flat unstructured set of clusters, require a prespecified number of clusters as input and are nondeterministic. *Hierarchical clustering* (or *hierarchic clustering*) outputs a hierarchy, a structure that is more informative than the unstructured set of clusters returned by flat clustering.[1] Hierarchical clustering does not require us to prespecify the number of clusters and most hierarchical algorithms that have been used in IR are deterministic. These advantages of hierarchical clustering come at the cost of lower efficiency. The most common hierarchical clustering algorithms have a complexity that is at least quadratic in the number of documents compared to the linear complexity of K-means and EM (cf. Section 16.4, page 364).

This chapter first introduces *agglomerative* hierarchical clustering (Section 17.1) and presents four different agglomerative algorithms, in Sections 17.2–17.4, which differ in the similarity measures they employ: single-link, complete-link, group-average, and centroid similarity. We then discuss the optimality conditions of hierarchical clustering in Section 17.5. Section 17.6 introduces top-down (or *divisive*) hierarchical clustering. Section 17.7 looks at labeling clusters automatically, a problem that must be solved whenever humans interact with the output of clustering. We discuss implementation issues in Section 17.8. Section 17.9 provides pointers to further reading, including references to soft hierarchical clustering, which we do not cover in this book.

There are few differences between the applications of flat and hierarchical clustering in information retrieval. In particular, hierarchical clustering is appropriate for any of the applications shown in Table 16.1 (page 351; see also Section 16.6, page 372). In fact, the example we gave for collection clustering is hierarchical. In general, we select flat clustering when efficiency is important and hierarchical clustering when one of the potential problems

1. In this chapter, we only consider hierarchies that are binary trees like the one shown in Figure 17.1 – but hierarchical clustering can be easily extended to other types of trees.

of flat clustering (not enough structure, predetermined number of clusters, non-determinism) is a concern. In addition, many researchers believe that hierarchical clustering produces better clusters than flat clustering. However, there is no consensus on this issue (see references in Section 17.9).

17.1 Hierarchical agglomerative clustering

Hierarchical clustering algorithms are either top-down or bottom-up. Bottom-up algorithms treat each document as a singleton cluster at the outset and then successively merge (or *agglomerate*) pairs of clusters until all clusters have been merged into a single cluster that contains all documents. Bottom-up hierarchical clustering is therefore called *hierarchical agglomerative clustering* or *HAC*. Top-down clustering requires a method for splitting a cluster. It proceeds by splitting clusters recursively until individual documents are reached. See Section 17.6. HAC is more frequently used in IR than top-down clustering and is the main subject of this chapter.

HIERARCHICAL AGGLOMERATIVE CLUSTERING
HAC

Before looking at specific similarity measures used in HAC in Sections 17.2–17.4, we first introduce a method for depicting hierarchical clusterings graphically, discuss a few key properties of HACs and present a simple algorithm for computing an HAC.

DENDROGRAM

An HAC clustering is typically visualized as a *dendrogram* as shown in Figure 17.1. Each merge is represented by a horizontal line. The y-coordinate of the horizontal line is the similarity of the two clusters that were merged, where documents are viewed as singleton clusters. We call this similarity the *combination similarity* of the merged cluster. For example, the combination similarity of the cluster consisting of *Lloyd's CEO questioned* and *Lloyd's chief / U.S. grilling* in Figure 17.1 is ≈ 0.56. We define the combination similarity of a singleton cluster as its document's self-similarity (which is 1.0 for cosine similarity).

COMBINATION SIMILARITY

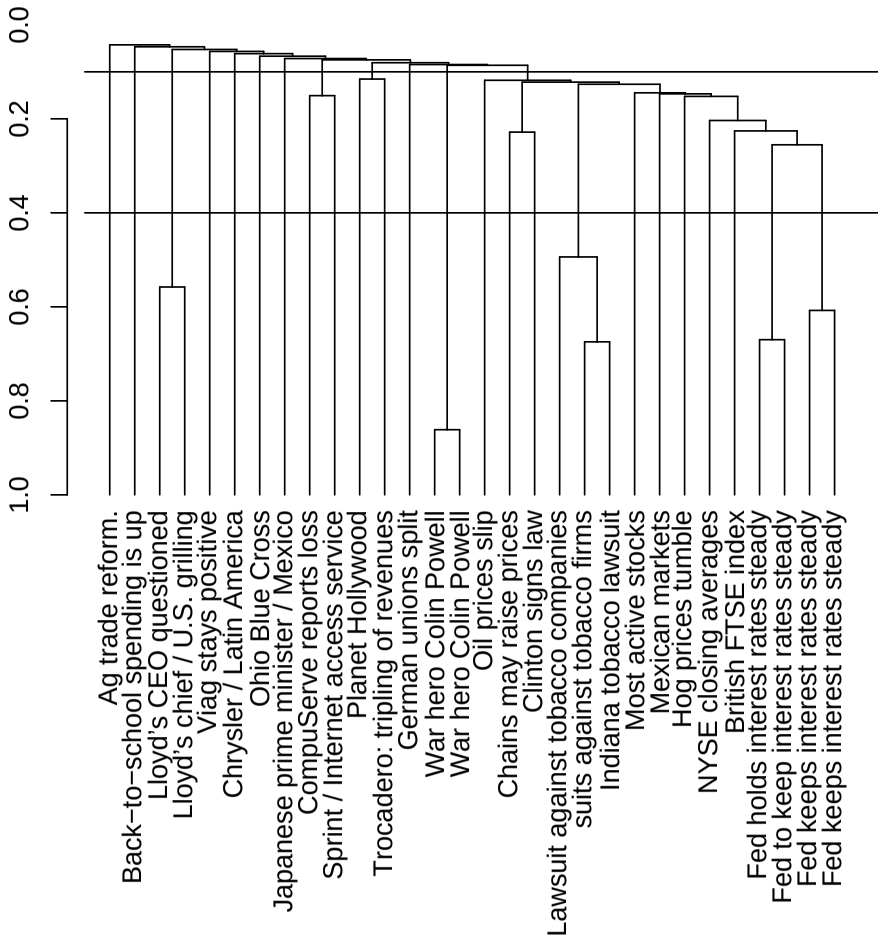
By moving up from the bottom layer to the top node, a dendrogram allows us to reconstruct the history of merges that resulted in the depicted clustering. For example, we see that the two documents entitled *War hero Colin Powell* were merged first in Figure 17.1 and that the last merge added *Ag trade reform* to a cluster consisting of the other 29 documents.

MONOTONICITY

A fundamental assumption in HAC is that the merge operation is *monotonic*. Monotonic means that if $s_1, s_2, \ldots, s_{K-1}$ are the combination similarities of the successive merges of an HAC, then $s_1 \geq s_2 \geq \ldots \geq s_{K-1}$ holds. A non-monotonic hierarchical clustering contains at least one *inversion* $s_i < s_{i+1}$ and contradicts the fundamental assumption that we chose the best merge available at each step. We will see an example of an inversion in Figure 17.12.

INVERSION

Hierarchical clustering does not require a prespecified number of clusters. However, in some applications we want a partition of disjoint clusters just as



► **Figure 17.1** A dendrogram of a single-link clustering of 30 documents from Reuters-RCV1. Two possible cuts of the dendrogram are shown: at 0.4 into 24 clusters and at 0.1 into 12 clusters.

in flat clustering. In those cases, the hierarchy needs to be cut at some point. A number of criteria can be used to determine the cutting point:

- Cut at a prespecified level of similarity. For example, we cut the dendrogram at 0.4 if we want clusters with a minimum combination similarity of 0.4. In Figure 17.1, cutting the diagram at $y = 0.4$ yields 24 clusters (grouping only documents with high similarity together) and cutting it at $y = 0.1$ yields 12 clusters (one large financial news cluster and 11 smaller clusters).
- Cut the dendrogram where the gap between two successive combination similarities is largest. Such large gaps arguably indicate "natural" clusterings. Adding one more cluster decreases the quality of the clustering significantly, so cutting before this steep decrease occurs is desirable. This strategy is analogous to looking for the knee in the K-means graph in Figure 16.8 (page 366).
- Apply Equation (16.11) (page 366):

$$K = \arg\min_{K'}[\text{RSS}(K') + \lambda K']$$

  where K' refers to the cut of the hierarchy that results in K' clusters, RSS is the residual sum of squares and λ is a penalty for each additional cluster. Instead of RSS, another measure of distortion can be used.
- As in flat clustering, we can also prespecify the number of clusters K and select the cutting point that produces K clusters.

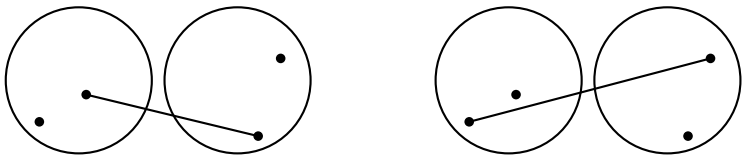
A simple, naive HAC algorithm is shown in Figure 17.2. We first compute the $N \times N$ similarity matrix C. The algorithm then executes $N - 1$ steps of merging the currently most similar clusters. In each iteration, the two most similar clusters are merged and the rows and columns of the merged cluster i in C are updated.[2] The clustering is stored as a list of merges in A. I indicates which clusters are still available to be merged. The function $\text{SIM}(i, m, j)$ computes the similarity of cluster j with the merge of clusters i and m. For some HAC algorithms, $\text{SIM}(i, m, j)$ is simply a function of $C[j][i]$ and $C[j][m]$, for example, the maximum of these two values for single-link.

We will now refine this algorithm for the different similarity measures of single-link and complete-link clustering (Section 17.2) and group-average and centroid clustering (Sections 17.3 and 17.4). The merge criteria of these four variants of HAC are shown in Figure 17.3.

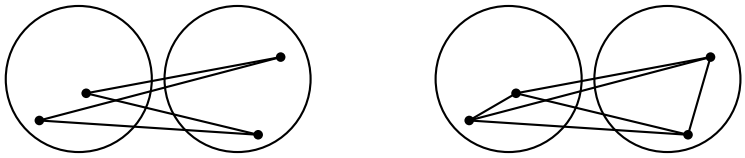
2. We assume that we use a deterministic method for breaking ties, such as always choose the merge that is the first cluster with respect to a total ordering of the subsets of the document set D.

SIMPLEHAC$(d_1, \ldots, d_N)$
1 **for** $n \leftarrow 1$ **to** N
2 **do for** $i \leftarrow 1$ **to** N
3 **do** $C[n][i] \leftarrow \text{SIM}(d_n, d_i)$
4 $I[n] \leftarrow 1$ *(keeps track of active clusters)*
5 $A \leftarrow []$ *(assembles clustering as a sequence of merges)*
6 **for** $k \leftarrow 1$ **to** $N - 1$
7 **do** $\langle i, m \rangle \leftarrow \arg\max_{\{\langle i,m \rangle : i \neq m \wedge I[i]=1 \wedge I[m]=1\}} C[i][m]$
8 A.APPEND$(\langle i, m \rangle)$ *(store merge)*
9 **for** $j \leftarrow 1$ **to** N
10 **do** $C[i][j] \leftarrow \text{SIM}(i, m, j)$
11 $C[j][i] \leftarrow \text{SIM}(i, m, j)$
12 $I[m] \leftarrow 0$ *(deactivate cluster)*
13 **return** A

► **Figure 17.2** A simple, but inefficient HAC algorithm.

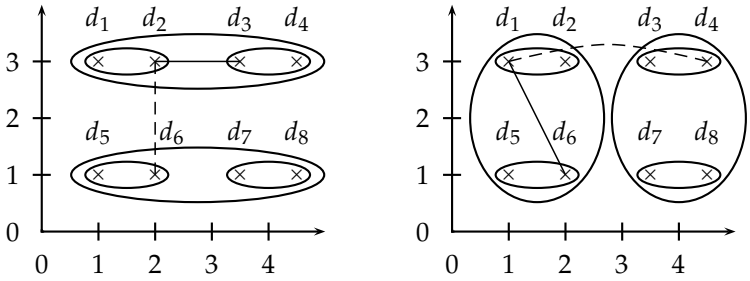


(a) single-link: maximum similarity (b) complete-link: minimum similarity



(c) centroid: average inter-similarity (d) group-average: average of all similarities

► **Figure 17.3** The different notions of cluster similarity used by the four HAC algorithms. An *inter-similarity* is a similarity between two documents from different clusters.



► **Figure 17.4** A single-link (left) and complete-link (right) clustering of eight documents. The ellipses correspond to successive clustering stages. Left: The single-link similarity of the two upper two-point clusters is the similarity of d_2 and d_3 (solid line), which is greater than the single-link similarity of the two left two-point clusters (dashed line). Right: The complete-link similarity of the two upper two-point clusters is the similarity of d_1 and d_4 (dashed line), which is smaller than the complete-link similarity of the two left two-point clusters (solid line).

17.2 Single-link and complete-link clustering

SINGLE-LINK CLUSTERING

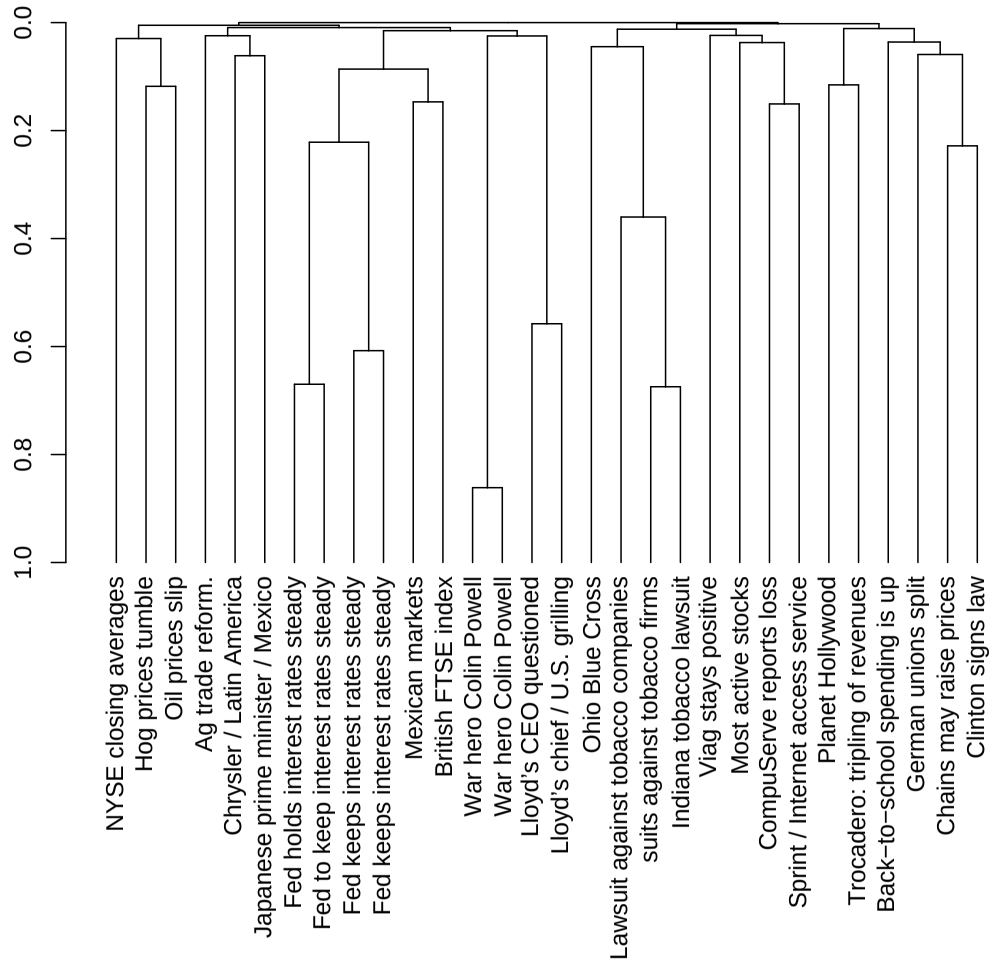
In *single-link clustering* or *single-linkage clustering*, the similarity of two clusters is the similarity of their *most similar* members (see Figure 17.3, (a))[3]. This single-link merge criterion is *local*. We pay attention solely to the area where the two clusters come closest to each other. Other, more distant parts of the cluster and the clusters' overall structure are not taken into account.

COMPLETE-LINK CLUSTERING

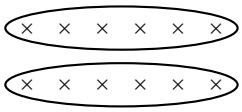
In *complete-link clustering* or *complete-linkage clustering*, the similarity of two clusters is the similarity of their *most dissimilar* members (see Figure 17.3, (b)). This is equivalent to choosing the cluster pair whose merge has the smallest diameter. This complete-link merge criterion is non-local; the entire structure of the clustering can influence merge decisions. This results in a preference for compact clusters with small diameters over long, straggly clusters, but also causes sensitivity to outliers. A single document far from the center can increase diameters of candidate merge clusters dramatically and completely change the final clustering.

Figure 17.4 depicts a single-link and a complete-link clustering of eight documents. The first four steps, each producing a cluster consisting of a pair of two documents, are identical. Then single-link clustering joins the upper two pairs (and after that the lower two pairs) because on the maximum-similarity definition of cluster similarity, those two clusters are closest. Complete-

3. Throughout this chapter, we equate similarity with proximity in 2D depictions of clustering.



► **Figure 17.5** A dendrogram of a complete-link clustering. The same 30 documents were clustered with single-link clustering in Figure 17.1.



► **Figure 17.6** Chaining in single-link clustering. The local criterion in single-link clustering can cause undesirable elongated clusters.

link clustering joins the left two pairs (and then the right two pairs) because those are the closest pairs according to the minimum-similarity definition of cluster similarity.[4]

Figure 17.1 is an example of a single-link clustering of a set of documents and Figure 17.5 is the complete-link clustering of the same set. When cutting the last merge in Figure 17.5, we obtain two clusters of similar size (documents 1–16, from *NYSE closing averages* to *Lloyd's chief / U.S. grilling*, and documents 17–30, from *Ohio Blue Cross* to *Clinton signs law*). There is no cut of the dendrogram in Figure 17.1 that would give us an equally balanced clustering.

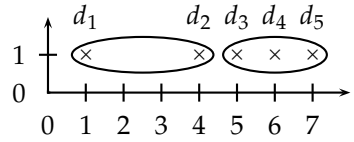
Both single-link and complete-link clustering have graph-theoretic interpretations. Define s_k to be the combination similarity of the two clusters merged in step k, and $G(s_k)$ the graph that links all data points with a similarity of at least s_k. Then the clusters after step k in single-link clustering are the connected components of $G(s_k)$ and the clusters after step k in complete-link clustering are maximal cliques of $G(s_k)$. A *connected component* is a maximal set of connected points such that there is a path connecting each pair. A *clique* is a set of points that are completely linked with each other.

CONNECTED COMPONENT
CLIQUE

These graph-theoretic interpretations motivate the terms single-link and complete-link clustering. Single-link clusters at step k are maximal sets of points that are linked via at least one link (a single link) of similarity $s \geq s_k$; complete-link clusters at step k are maximal sets of points that are completely linked with each other via links of similarity $s \geq s_k$.

Single-link and complete-link clustering reduce the assessment of cluster quality to a single similarity between a pair of documents: the two most similar documents in single-link clustering and the two most dissimilar documents in complete-link clustering. A measurement based on one pair cannot fully reflect the distribution of documents in a cluster. It is therefore not surprising that both algorithms often produce undesirable clusters. Single-link clustering can produce straggling clusters as shown in Figure 17.6. Since the merge criterion is strictly local, a chain of points can be extended for long

4. If you are bothered by the possibility of ties, assume that d_1 has coordinates $(1+\epsilon, 3-\epsilon)$ and that all other points have integer coordinates.



▶ **Figure 17.7** Outliers in complete-link clustering. The five documents have the x-coordinates $1+2\epsilon, 4, 5+2\epsilon, 6$ and $7-\epsilon$. Complete-link clustering creates the two clusters shown as ellipses. The most intuitive two-cluster clustering is $\{\{d_1\},\{d_2,d_3,d_4,d_5\}\}$, but in complete-link clustering, the outlier d_1 splits $\{d_2,d_3,d_4,d_5\}$ as shown.

CHAINING

distances without regard to the overall shape of the emerging cluster. This effect is called *chaining*.

The chaining effect is also apparent in Figure 17.1. The last eleven merges of the single-link clustering (those above the 0.1 line) add on single documents or pairs of documents, corresponding to a chain. The complete-link clustering in Figure 17.5 avoids this problem. Documents are split into two groups of roughly equal size when we cut the dendrogram at the last merge. In general, this is a more useful organization of the data than a clustering with chains.

However, complete-link clustering suffers from a different problem. It pays too much attention to outliers, points that do not fit well into the global structure of the cluster. In the example in Figure 17.7 the four documents d_2, d_3, d_4, d_5 are split because of the outlier d_1 at the left edge (Exercise 17.1). Complete-link clustering does not find the most intuitive cluster structure in this example.

17.2.1 Time complexity of HAC

The complexity of the naive HAC algorithm in Figure 17.2 is $\Theta(N^3)$ because we exhaustively scan the $N \times N$ matrix C for the largest similarity in each of $N-1$ iterations.

For the four HAC methods discussed in this chapter a more efficient algorithm is the priority-queue algorithm shown in Figure 17.8. Its time complexity is $\Theta(N^2 \log N)$. The rows $C[k]$ of the $N \times N$ similarity matrix C are sorted in decreasing order of similarity in the priority queues P. $P[k]$.MAX() then returns the cluster in $P[k]$ that currently has the highest similarity with ω_k, where we use ω_k to denote the k^{th} cluster as in Chapter 16. After creating the merged cluster of ω_{k_1} and ω_{k_2}, ω_{k_1} is used as its representative. The function SIM computes the similarity function for potential merge pairs: largest similarity for single-link, smallest similarity for complete-link, average similarity for GAAC (Section 17.3), and centroid similarity for centroid clustering (Sec-

EFFICIENTHAC($\vec{d}_1, \ldots, \vec{d}_N$)
1 **for** $n \leftarrow 1$ **to** N
2 **do for** $i \leftarrow 1$ **to** N
3 **do** $C[n][i].\text{sim} \leftarrow \vec{d}_n \cdot \vec{d}_i$
4 $C[n][i].\text{index} \leftarrow i$
5 $I[n] \leftarrow 1$
6 $P[n] \leftarrow$ priority queue for $C[n]$ sorted on sim
7 $P[n].\text{DELETE}(C[n][n])$ *(don't want self-similarities)*
8 $A \leftarrow []$
9 **for** $k \leftarrow 1$ **to** $N - 1$
10 **do** $k_1 \leftarrow \arg\max_{\{k:I[k]=1\}} P[k].\text{MAX}().\text{sim}$
11 $k_2 \leftarrow P[k_1].\text{MAX}().\text{index}$
12 $A.\text{APPEND}(\langle k_1, k_2 \rangle)$
13 $I[k_2] \leftarrow 0$
14 $P[k_1] \leftarrow []$
15 **for each** i **with** $I[i] = 1 \wedge i \neq k_1$
16 **do** $P[i].\text{DELETE}(C[i][k_1])$
17 $P[i].\text{DELETE}(C[i][k_2])$
18 $C[i][k_1].\text{sim} \leftarrow \text{SIM}(i, k_1, k_2)$
19 $P[i].\text{INSERT}(C[i][k_1])$
20 $C[k_1][i].\text{sim} \leftarrow \text{SIM}(i, k_1, k_2)$
21 $P[k_1].\text{INSERT}(C[k_1][i])$
22 **return** A

clustering algorithm	$\text{SIM}(i, k_1, k_2)$
single-link	$\max(\text{SIM}(i, k_1), \text{SIM}(i, k_2))$
complete-link	$\min(\text{SIM}(i, k_1), \text{SIM}(i, k_2))$
centroid	$(\frac{1}{N_m}\vec{v}_m) \cdot (\frac{1}{N_i}\vec{v}_i)$
group-average	$\frac{1}{(N_m+N_i)(N_m+N_i-1)}[(\vec{v}_m + \vec{v}_i)^2 - (N_m + N_i)]$

compute $C[5]$	1	2	3	4	5
	0.2	0.8	0.6	0.4	1.0
create $P[5]$ (by sorting)	2	3	4	1	
	0.8	0.6	0.4	0.2	
merge 2 and 3, update similarity of 2, delete 3	2	4	1		
	0.3	0.4	0.2		
delete and reinsert 2	4	2	1		
	0.4	0.3	0.2		

► **Figure 17.8** The priority-queue algorithm for HAC. Top: The algorithm. Center: Four different similarity measures. Bottom: An example for processing steps 6 and 16–19. This is a made up example showing $P[5]$ for a 5×5 matrix C.

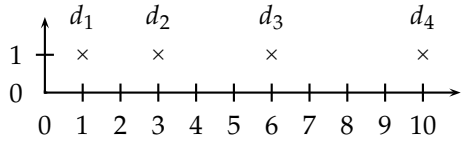
```
SINGLELINKCLUSTERING(d_1, ..., d_N)
 1  for n ← 1 to N
 2  do for i ← 1 to N
 3     do C[n][i].sim ← SIM(d_n, d_i)
 4        C[n][i].index ← i
 5     I[n] ← n
 6     NBM[n] ← arg max_{X∈{C[n][i]:n≠i}} X.sim
 7  A ← []
 8  for n ← 1 to N − 1
 9  do i_1 ← arg max_{i:I[i]=i} NBM[i].sim
10     i_2 ← I[NBM[i_1].index]
11     A.APPEND(⟨i_1, i_2⟩)
12     for i ← 1 to N
13     do if I[i] = i ∧ i ≠ i_1 ∧ i ≠ i_2
14           then C[i_1][i].sim ← C[i][i_1].sim ← max(C[i_1][i].sim, C[i_2][i].sim)
15        if I[i] = i_2
16           then I[i] ← i_1
17     NBM[i_1] ← arg max_{X∈{C[i_1][i]:I[i]=i∧i≠i_1}} X.sim
18  return A
```

▶ **Figure 17.9** Single-link clustering algorithm using an NBM array. After merging two clusters i_1 and i_2, the first one (i_1) represents the merged cluster. If $I[i] = i$, then i is the representative of its current cluster. If $I[i] \neq i$, then i has been merged into the cluster represented by $I[i]$ and will therefore be ignored when updating $NBM[i_1]$.

tion 17.4). We give an example of how a row of C is processed (Figure 17.8, bottom panel). The loop in lines 1–7 is $\Theta(N^2)$ and the loop in lines 9–21 is $\Theta(N^2 \log N)$ for an implementation of priority queues that supports deletion and insertion in $\Theta(\log N)$. The overall complexity of the algorithm is therefore $\Theta(N^2 \log N)$. In the definition of the function SIM, $\vec{v}_m$ and $\vec{v}_i$ are the vector sums of $\omega_{k_1} \cup \omega_{k_2}$ and ω_i, respectively, and N_m and N_i are the number of documents in $\omega_{k_1} \cup \omega_{k_2}$ and ω_i, respectively.

The argument of EFFICIENTHAC in Figure 17.8 is a set of vectors (as opposed to a set of generic documents) because GAAC and centroid clustering (Sections 17.3 and 17.4) require vectors as input. The complete-link version of EFFICIENTHAC can also be applied to documents that are not represented as vectors.

For single-link, we can introduce a next-best-merge array (NBM) as a further optimization as shown in Figure 17.9. NBM keeps track of what the best merge is for each cluster. Each of the two top level for-loops in Figure 17.9 are $\Theta(N^2)$, thus the overall complexity of single-link clustering is $\Theta(N^2)$.



► **Figure 17.10** Complete-link clustering is not best-merge persistent. At first, d_2 is the best-merge cluster for d_3. But after merging d_1 and d_2, d_4 becomes d_3's best-merge candidate. In a best-merge persistent algorithm like single-link, d_3's best-merge cluster would be $\{d_1, d_2\}$.

BEST-MERGE PERSISTENCE

Can we also speed up the other three HAC algorithms with an NBM array? We cannot because only single-link clustering is *best-merge persistent*. Suppose that the best merge cluster for ω_k is ω_j in single-link clustering. Then after merging ω_j with a third cluster $\omega_i \neq \omega_k$, the merge of ω_i and ω_j will be ω_k's best merge cluster (Exercise 17.6). In other words, the best-merge candidate for the merged cluster is one of the two best-merge candidates of its components in single-link clustering. This means that C can be updated in $\Theta(N)$ in each iteration – by taking a simple max of two values on line 14 in Figure 17.9 for each of the remaining $\leq N$ clusters.

Figure 17.10 demonstrates that best-merge persistence does not hold for complete-link clustering, which means that we cannot use an NBM array to speed up clustering. After merging d_3's best merge candidate d_2 with cluster d_1, an unrelated cluster d_4 becomes the best merge candidate for d_3. This is because the complete-link merge criterion is non-local and can be affected by points at a great distance from the area where two merge candidates meet.

In practice, the efficiency penalty of the $\Theta(N^2 \log N)$ algorithm is small compared with the $\Theta(N^2)$ single-link algorithm since computing the similarity between two documents (e.g., as a dot product) is an order of magnitude slower than comparing two scalars in sorting. All four HAC algorithms in this chapter are $\Theta(N^2)$ with respect to similarity computations. So the difference in complexity is rarely a concern in practice when choosing one of the algorithms.

?

Exercise 17.1

Show that complete-link clustering creates the two-cluster clustering depicted in Figure 17.7.

17.3 Group-average agglomerative clustering

GROUP-AVERAGE AGGLOMERATIVE CLUSTERING

Group-average agglomerative clustering or *GAAC* (see Figure 17.3, (d)) evaluates cluster quality based on *all* similarities between documents, thus avoiding the pitfalls of the single-link and complete-link criteria, which equate cluster

similarity with the similarity of a single pair of documents. GAAC is also called *group-average clustering* and *average-link clustering*. GAAC computes the average similarity SIM-GA of all pairs of documents, including pairs from the same cluster. But self-similarities are not included in the average:

$$\text{SIM-GA}(\omega_i, \omega_j) = \frac{1}{(N_i + N_j)(N_i + N_j - 1)} \sum_{d_m \in \omega_i \cup \omega_j} \sum_{d_n \in \omega_i \cup \omega_j, d_n \neq d_m} \vec{d}_m \cdot \vec{d}_n \tag{17.1}$$

where $\vec{d}$ is the length-normalized vector of document d, $\cdot$ denotes the dot product, and N_i and N_j are the number of documents in ω_i and ω_j, respectively.

The motivation for GAAC is that our goal in selecting two clusters ω_i and ω_j as the next merge in HAC is that the resulting merge cluster $\omega_k = \omega_i \cup \omega_j$ should be coherent. To judge the coherence of ω_k, we need to look at all document-document similarities within ω_k, including those that occur within ω_i and those that occur within ω_j.

We can compute the measure SIM-GA efficiently because the sum of individual vector similarities is equal to the similarities of their sums:

$$\sum_{d_m \in \omega_i} \sum_{d_n \in \omega_j} (\vec{d}_m \cdot \vec{d}_n) = (\sum_{d_m \in \omega_i} \vec{d}_m) \cdot (\sum_{d_n \in \omega_j} \vec{d}_n) \tag{17.2}$$

With (17.2), we have:

$$\text{SIM-GA}(\omega_i, \omega_j) = \frac{1}{(N_i + N_j)(N_i + N_j - 1)} [(\sum_{d_m \in \omega_i \cup \omega_j} \vec{d}_m)^2 - (N_i + N_j)] \tag{17.3}$$

The term $(N_i + N_j)$ on the right is the sum of $N_i + N_j$ self-similarities of value 1.0. With this trick we can compute cluster similarity in constant time (assuming we have available the two vector sums $\sum_{d_m \in \omega_i} \vec{d}_m$ and $\sum_{d_m \in \omega_j} \vec{d}_m$) instead of in $\Theta(N_i N_j)$. This is important because we need to be able to compute the function SIM on lines 18 and 20 in EFFICIENTHAC (Figure 17.8) in constant time for efficient implementations of GAAC. Note that for two singleton clusters, Equation (17.3) is equivalent to the dot product.

Equation (17.2) relies on the distributivity of the dot product with respect to vector addition. Since this is crucial for the efficient computation of a GAAC clustering, the method cannot be easily applied to representations of documents that are not real-valued vectors. Also, Equation (17.2) only holds for the dot product. While many algorithms introduced in this book have near-equivalent descriptions in terms of dot product, cosine similarity and Euclidean distance (cf. Section 14.1, page 291), Equation (17.2) can only be expressed using the dot product. This is a fundamental difference between single-link/complete-link clustering and GAAC. The first two only require a

square matrix of similarities as input and do not care how these similarities were computed.

To summarize, GAAC requires (i) documents represented as vectors, (ii) length normalization of vectors, so that self-similarities are 1.0, and (iii) the dot product as the measure of similarity between vectors and sums of vectors.

The merge algorithms for GAAC and complete-link clustering are the same except that we use Equation (17.3) as similarity function in Figure 17.8. Therefore, the overall time complexity of GAAC is the same as for complete-link clustering: $\Theta(N^2 \log N)$. Like complete-link clustering, GAAC is not best-merge persistent (Exercise 17.6). This means that there is no $\Theta(N^2)$ algorithm for GAAC that would be analogous to the $\Theta(N^2)$ algorithm for single-link in Figure 17.9.

We can also define group-average similarity as including self-similarities:

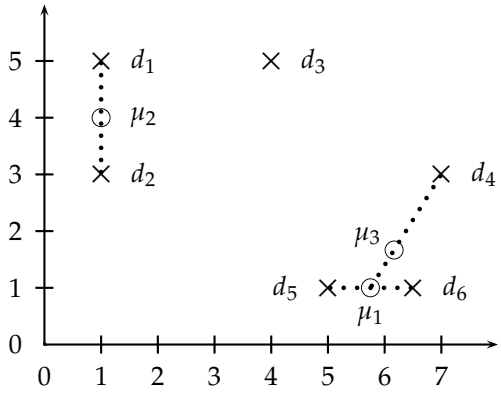
$$\text{SIM-GA}'(\omega_i, \omega_j) = \frac{1}{(N_i+N_j)^2}\Big(\sum_{d_m \in \omega_i \cup \omega_j} \vec{d}_m\Big)^2 = \frac{1}{N_i+N_j} \sum_{d_m \in \omega_i \cup \omega_j} [\vec{d}_m \cdot \vec{\mu}(\omega_i \cup \omega_j)] \tag{17.4}$$

where the centroid $\vec{\mu}(\omega)$ is defined as in Equation (14.1) (page 292). This definition is equivalent to the intuitive definition of cluster quality as average similarity of documents $\vec{d}_m$ to the cluster's centroid $\vec{\mu}$.

Self-similarities are always equal to 1.0, the maximum possible value for length-normalized vectors. The proportion of self-similarities in Equation (17.4) is $i/i^2 = 1/i$ for a cluster of size i. This gives an unfair advantage to small clusters since they will have proportionally more self-similarities. For two documents d_1, d_2 with a similarity s, we have $\text{SIM-GA}'(d_1, d_2) = (1+s)/2$. In contrast, $\text{SIM-GA}(d_1, d_2) = s \leq (1+s)/2$. This similarity $\text{SIM-GA}(d_1, d_2)$ of two documents is the same as in single-link, complete-link and centroid clustering. We prefer the definition in Equation (17.3), which excludes self-similarities from the average, because we do not want to penalize large clusters for their smaller proportion of self-similarities and because we want a consistent similarity value s for document pairs in all four HAC algorithms.

? **Exercise 17.2**

Apply group-average clustering to the points in Figures 17.6 and 17.7. Map them onto the surface of the unit sphere in a three-dimensional space to get length-normalized vectors. Is the group-average clustering different from the single-link and complete-link clusterings?



► **Figure 17.11** Three iterations of centroid clustering. Each iteration merges the two clusters whose centroids are closest.

17.4 Centroid clustering

In centroid clustering, the similarity of two clusters is defined as the similarity of their centroids:

$$\text{SIM-CENT}(\omega_i, \omega_j) = \vec{\mu}(\omega_i) \cdot \vec{\mu}(\omega_j) \tag{17.5}$$

$$= \left(\frac{1}{N_i} \sum_{d_m \in \omega_i} \vec{d}_m\right) \cdot \left(\frac{1}{N_j} \sum_{d_n \in \omega_j} \vec{d}_n\right)$$

$$= \frac{1}{N_i N_j} \sum_{d_m \in \omega_i} \sum_{d_n \in \omega_j} \vec{d}_m \cdot \vec{d}_n \tag{17.6}$$

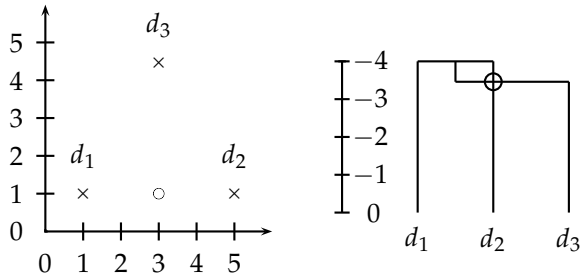
Equation (17.5) is centroid similarity. Equation (17.6) shows that centroid similarity is equivalent to average similarity of all pairs of documents from *different* clusters. Thus, the difference between GAAC and centroid clustering is that GAAC considers all pairs of documents in computing average pairwise similarity (Figure 17.3, (d)) whereas centroid clustering excludes pairs from the same cluster (Figure 17.3, (c)).

Figure 17.11 shows the first three steps of a centroid clustering. The first two iterations form the clusters $\{d_5, d_6\}$ with centroid μ_1 and $\{d_1, d_2\}$ with centroid μ_2 because the pairs $\langle d_5, d_6\rangle$ and $\langle d_1, d_2\rangle$ have the highest centroid similarities. In the third iteration, the highest centroid similarity is between μ_1 and d_4 producing the cluster $\{d_4, d_5, d_6\}$ with centroid μ_3.

Like GAAC, centroid clustering is not best-merge persistent and therefore $\Theta(N^2 \log N)$ (Exercise 17.6).

INVERSION

In contrast to the other three HAC algorithms, centroid clustering is not monotonic. So-called *inversions* can occur: Similarity can increase during



► **Figure 17.12** Centroid clustering is not monotonic. The documents d_1 at $(1+\epsilon, 1)$, d_2 at $(5, 1)$, and d_3 at $(3, 1+2\sqrt{3})$ are almost equidistant, with d_1 and d_2 closer to each other than to d_3. The non-monotonic inversion in the hierarchical clustering of the three points appears as an intersecting merge line in the dendrogram. The intersection is circled.

clustering as in the example in Figure 17.12, where we define similarity as negative distance. In the first merge, the similarity of d_1 and d_2 is $-(4-\epsilon)$. In the second merge, the similarity of the centroid of d_1 and d_2 (the circle) and d_3 is $\approx -\cos(\pi/6) \times 4 = -\sqrt{3}/2 \times 4 \approx -3.46 > -(4-\epsilon)$. This is an example of an inversion: similarity *increases* in this sequence of two clustering steps. In a monotonic HAC algorithm, similarity is monotonically *decreasing* from iteration to iteration.

Increasing similarity in a series of HAC clustering steps contradicts the fundamental assumption that small clusters are more coherent than large clusters. An inversion in a dendrogram shows up as a horizontal merge line that is *lower* than the previous merge line. All merge lines in Figures 17.1 and 17.5 are higher than their predecessors because single-link and complete-link clustering are monotonic clustering algorithms.

Despite its non-monotonicity, centroid clustering is often used because its similarity measure – the similarity of two centroids – is conceptually simpler than the average of all pairwise similarities in GAAC. Figure 17.11 is all one needs to understand centroid clustering. There is no equally simple graph that would explain how GAAC works.

?

Exercise 17.3

For a fixed set of N documents there are up to N^2 distinct similarities between clusters in single-link and complete-link clustering. How many distinct cluster similarities are there in GAAC and centroid clustering?

17.5 Optimality of HAC

To state the optimality conditions of hierarchical clustering precisely, we first define the combination similarity COMB-SIM of a clustering $\Omega = \{\omega_1, \ldots, \omega_K\}$ as the smallest combination similarity of any of its K clusters:

$$\text{COMB-SIM}(\{\omega_1, \ldots, \omega_K\}) = \min_k \text{COMB-SIM}(\omega_k)$$

Recall that the combination similarity of a cluster ω that was created as the merge of ω_1 and ω_2 is the similarity of ω_1 and ω_2 (page 378).

OPTIMAL CLUSTERING

We then define $\Omega = \{\omega_1, \ldots, \omega_K\}$ to be *optimal* if all clusterings Ω' with k clusters, $k \leq K$, have lower combination similarities:

$$|\Omega'| \leq |\Omega| \Rightarrow \text{COMB-SIM}(\Omega') \leq \text{COMB-SIM}(\Omega)$$

Figure 17.12 shows that centroid clustering is not optimal. The clustering $\{\{d_1, d_2\}, \{d_3\}\}$ (for $K = 2$) has combination similarity $-(4 - \epsilon)$ and $\{\{d_1, d_2, d_3\}\}$ (for $K = 1$) has combination similarity -3.46. So the clustering $\{\{d_1, d_2\}, \{d_3\}\}$ produced in the first merge is not optimal since there is a clustering with fewer clusters ($\{\{d_1, d_2, d_3\}\}$) that has higher combination similarity. Centroid clustering is not optimal because inversions can occur.

The above definition of optimality would be of limited use if it was only applicable to a clustering together with its merge history. However, we can show (Exercise 17.4) that combination similarity for the three non-inversion algorithms can be read off from the cluster without knowing its history. These direct definitions of combination similarity are as follows.

COMBINATION SIMILARITY

single-link The combination similarity of a cluster ω is the smallest similarity of any bipartition of the cluster, where the similarity of a bipartition is the largest similarity between any two documents from the two parts:

$$\text{COMB-SIM}(\omega) = \min_{\{\omega' : \omega' \subset \omega\}} \max_{d_i \in \omega'} \max_{d_j \in \omega - \omega'} \text{SIM}(d_i, d_j)$$

where each $\langle \omega', \omega - \omega' \rangle$ is a bipartition of ω.

complete-link The combination similarity of a cluster ω is the smallest similarity of any two points in ω: $\min_{d_i \in \omega} \min_{d_j \in \omega} \text{SIM}(d_i, d_j)$.

GAAC The combination similarity of a cluster ω is the average of all pairwise similarities in ω (where self-similarities are not included in the average): Equation (17.3).

If we use these definitions of combination similarity, then optimality is a property of a set of clusters and not of a process that produces a set of clusters.

We can now prove the optimality of single-link clustering by induction over the number of clusters K. We will give a proof for the case where no two pairs of documents have the same similarity, but it can easily be extended to the case with ties.

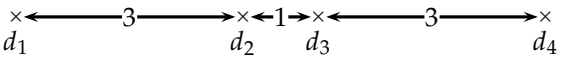
The inductive basis of the proof is that a clustering with $K = N$ clusters has combination similarity 1.0, which is the largest value possible. The induction hypothesis is that a single-link clustering Ω_K with K clusters is optimal: $\text{COMB-SIM}(\Omega_K) \geq \text{COMB-SIM}(\Omega'_K)$ for all Ω'_K. Assume for contradiction that the clustering Ω_{K-1} we obtain by merging the two most similar clusters in Ω_K is not optimal and that instead a different sequence of merges Ω'_K, Ω'_{K-1} leads to the optimal clustering with $K-1$ clusters. We can write the assumption that Ω'_{K-1} is optimal and that Ω_{K-1} is not as $\text{COMB-SIM}(\Omega'_{K-1}) > \text{COMB-SIM}(\Omega_{K-1})$.

Case 1: The two documents linked by $s = \text{COMB-SIM}(\Omega'_{K-1})$ are in the same cluster in Ω_K. They can only be in the same cluster if a merge with similarity smaller than s has occurred in the merge sequence producing Ω_K. This implies $s > \text{COMB-SIM}(\Omega_K)$. Thus, $\text{COMB-SIM}(\Omega'_{K-1}) = s > \text{COMB-SIM}(\Omega_K) > \text{COMB-SIM}(\Omega'_K) > \text{COMB-SIM}(\Omega'_{K-1})$. Contradiction.

Case 2: The two documents linked by $s = \text{COMB-SIM}(\Omega'_{K-1})$ are not in the same cluster in Ω_K. But $s = \text{COMB-SIM}(\Omega'_{K-1}) > \text{COMB-SIM}(\Omega_{K-1})$, so the single-link merging rule should have merged these two clusters when processing Ω_K. Contradiction.

Thus, Ω_{K-1} is optimal.

In contrast to single-link clustering, complete-link clustering and GAAC are not optimal as this example shows:



Both algorithms merge the two points with distance 1 (d_2 and d_3) first and thus cannot find the two-cluster clustering $\{\{d_1, d_2\}, \{d_3, d_4\}\}$. But $\{\{d_1, d_2\}, \{d_3, d_4\}\}$ is optimal on the optimality criteria of complete-link clustering and GAAC.

However, the merge criteria of complete-link clustering and GAAC approximate the desideratum of approximate sphericity better than the merge criterion of single-link clustering. In many applications, we want spherical clusters. Thus, even though single-link clustering may seem preferable at first because of its optimality, it is optimal with respect to the wrong criterion in many document clustering applications.

Table 17.1 summarizes the properties of the four HAC algorithms introduced in this chapter. We recommend GAAC for document clustering because it is generally the method that produces the clustering with the best

method	combination similarity	time compl.	optimal?	comment
single-link	max inter-similarity of any 2 docs	$\Theta(N^2)$	yes	chaining effect
complete-link	min inter-similarity of any 2 docs	$\Theta(N^2 \log N)$	no	sensitive to outliers
group-average	average of all sims	$\Theta(N^2 \log N)$	no	best choice for most applications
centroid	average inter-similarity	$\Theta(N^2 \log N)$	no	inversions can occur

► **Table 17.1** Comparison of HAC algorithms.

properties for applications. It does not suffer from chaining, from sensitivity to outliers and from inversions.

There are two exceptions to this recommendation. First, for non-vector representations, GAAC is not applicable and clustering should typically be performed with the complete-link method.

FIRST STORY DETECTION

Second, in some applications the purpose of clustering is not to create a complete hierarchy or exhaustive partition of the entire document set. For instance, *first story detection* or *novelty detection* is the task of detecting the first occurrence of an event in a stream of news stories. One approach to this task is to find a tight cluster within the documents that were sent across the wire in a short period of time and are dissimilar from all previous documents. For example, the documents sent over the wire in the minutes after the World Trade Center attack on September 11, 2001 form such a cluster. Variations of single-link clustering can do well on this task since it is the structure of small parts of the vector space – and not global structure – that is important in this case.

Similarly, we will describe an approach to duplicate detection on the web in Section 19.6 (page 440) where single-link clustering is used in the guise of the union-find algorithm. Again, the decision whether a group of documents are duplicates of each other is not influenced by documents that are located far away and single-link clustering is a good choice for duplicate detection.

?

Exercise 17.4

Show the equivalence of the two definitions of combination similarity: the process definition on page 378 and the static definition on page 393.

17.6 Divisive clustering

TOP-DOWN CLUSTERING

So far we have only looked at agglomerative clustering, but a cluster hierarchy can also be generated top-down. This variant of hierarchical clustering is called *top-down clustering* or *divisive clustering*. We start at the top with all documents in one cluster. The cluster is split using a flat clustering algo-

rithm. This procedure is applied recursively until each document is in its own singleton cluster.

Top-down clustering is conceptually more complex than bottom-up clustering since we need a second, flat clustering algorithm as a "subroutine". It has the advantage of being more efficient if we do not generate a complete hierarchy all the way down to individual document leaves. For a fixed number of top levels, using an efficient flat algorithm like K-means, top-down algorithms are linear in the number of documents and clusters. So they run much faster than HAC algorithms, which are at least quadratic.

There is evidence that divisive algorithms produce more accurate hierarchies than bottom-up algorithms in some circumstances. See the references on bisecting K-means in Section 17.9. Bottom-up methods make clustering decisions based on local patterns without initially taking into account the global distribution. These early decisions cannot be undone. Top-down clustering benefits from complete information about the global distribution when making top-level partitioning decisions.

17.7 Cluster labeling

In many applications of flat clustering and hierarchical clustering, particularly in analysis tasks and in user interfaces (see applications in Table 16.1, page 351), human users interact with clusters. In such settings, we must label clusters, so that users can see what a cluster is about.

DIFFERENTIAL CLUSTER LABELING

Differential cluster labeling selects cluster labels by comparing the distribution of terms in one cluster with that of other clusters. The feature selection methods we introduced in Section 13.5 (page 271) can all be used for differential cluster labeling.[5] In particular, mutual information (MI) (Section 13.5.1, page 272) or, equivalently, information gain and the χ^2-test (Section 13.5.2, page 275) will identify cluster labels that characterize one cluster in contrast to other clusters. A combination of a differential test with a penalty for rare terms often gives the best labeling results because rare terms are not necessarily representative of the cluster as a whole.

We apply three labeling methods to a K-means clustering in Table 17.2. In this example, there is almost no difference between MI and χ^2. We therefore omit the latter.

CLUSTER-INTERNAL LABELING

Cluster-internal labeling computes a label that solely depends on the cluster itself, not on other clusters. Labeling a cluster with the title of the document closest to the centroid is one cluster-internal method. Titles are easier to read than a list of terms. A full title can also contain important context that didn't make it into the top 10 terms selected by MI. On the web, anchor text can

5. Selecting the most frequent terms is a non-differential feature selection technique we discussed in Section 13.5. It can also be used for labeling clusters.

	# docs	labeling method		
		centroid	mutual information	title
4	622	oil plant mexico production crude **power 000 refinery gas** bpd	plant oil production **barrels** crude bpd mexico **dolly capacity petroleum**	MEXICO: Hurricane Dolly heads for Mexico coast
9	1017	police security **russian** people military peace killed told **grozny court**	police killed military security peace told **troops forces rebels** people	RUSSIA: Russia's Lebed meets rebel chief in Chechnya
10	1259	00 000 tonnes traders futures wheat prices **cents september** tonne	**delivery** traders futures tonne tonnes **desk** wheat prices 000 00	USA: Export Business - Grain/oilseeds complex

► **Table 17.2** Automatically computed cluster labels. This is for three of ten clusters (4, 9, and 10) in a K-means clustering of the first 10,000 documents in Reuters-RCV1. The last three columns show cluster summaries computed by three labeling methods: most highly weighted terms in centroid (centroid), mutual information, and the title of the document closest to the centroid of the cluster (title). Terms selected by only one of the first two methods are in bold.

play a role similar to a title since the anchor text pointing to a page can serve as a concise summary of its contents.

In Table 17.2, the title for cluster 9 suggests that many of its documents are about the Chechnya conflict, a fact the MI terms do not reveal. However, a single document is unlikely to be representative of all documents in a cluster. An example is cluster 4, whose selected title is misleading. The main topic of the cluster is oil. Articles about hurricane Dolly only ended up in this cluster because of its effect on oil prices.

We can also use a list of terms with high weights in the centroid of the cluster as a label. Such highly weighted terms (or, even better, phrases, especially noun phrases) are often more representative of the cluster than a few titles can be, even if they are not filtered for distinctiveness as in the differential methods. However, a list of phrases takes more time to digest for users than a well crafted title.

Cluster-internal methods are efficient, but they fail to distinguish terms that are frequent in the collection as a whole from those that are frequent only in the cluster. Terms like year or Tuesday may be among the most frequent in a cluster, but they are not helpful in understanding the contents of a cluster with a specific topic like oil.

In Table 17.2, the centroid method selects a few more uninformative terms (000, court, cents, september) than MI (forces, desk), but most of the terms se-

lected by either method are good descriptors. We get a good sense of the documents in a cluster from scanning the selected terms.

For hierarchical clustering, additional complications arise in cluster labeling. Not only do we need to distinguish an internal node in the tree from its siblings, but also from its parent and its children. Documents in child nodes are by definition also members of their parent node, so we cannot use a naive differential method to find labels that distinguish the parent from its children. However, more complex criteria, based on a combination of overall collection frequency and prevalence in a given cluster, can determine whether a term is a more informative label for a child node or a parent node (see Section 17.9).

17.8 Implementation notes

Most problems that require the computation of a large number of dot products benefit from an inverted index. This is also the case for HAC clustering. Computational savings due to the inverted index are large if there are many zero similarities – either because many documents do not share any terms or because an aggressive stop list is used.

In low dimensions, more aggressive optimizations are possible that make the computation of most pairwise similarities unnecessary (Exercise 17.10). However, no such algorithms are known in higher dimensions. We encountered the same problem in kNN classification (see Section 14.7, page 314).

When using GAAC on a large document set in high dimensions, we have to take care to avoid dense centroids. For dense centroids, clustering can take time $\Theta(MN^2 \log N)$ where M is the size of the vocabulary, whereas complete-link clustering is $\Theta(M_{\text{ave}} N^2 \log N)$ where M_{ave} is the average size of the vocabulary of a document. So for large vocabularies complete-link clustering can be more efficient than an unoptimized implementation of GAAC. We discussed this problem in the context of K-means clustering in Chapter 16 (page 365) and suggested two solutions: truncating centroids (keeping only highly weighted terms) and representing clusters by means of sparse medoids instead of dense centroids. These optimizations can also be applied to GAAC and centroid clustering.

Even with these optimizations, HAC algorithms are all $\Theta(N^2)$ or $\Theta(N^2 \log N)$ and therefore infeasible for large sets of 1,000,000 or more documents. For such large sets, HAC can only be used in combination with a flat clustering algorithm like K-means. Recall that K-means requires a set of seeds as initialization (Figure 16.5, page 361). If these seeds are badly chosen, then the resulting clustering will be of poor quality. We can employ an HAC algorithm to compute seeds of high quality. If the HAC algorithm is applied to a document subset of size $\sqrt{N}$, then the overall runtime of K-means cum HAC seed

generation is $\Theta(N)$. This is because the application of a quadratic algorithm to a sample of size $\sqrt{N}$ has an overall complexity of $\Theta(N)$. An appropriate adjustment can be made for an $\Theta(N^2 \log N)$ algorithm to guarantee linearity. This algorithm is referred to as the *Buckshot algorithm*. It combines the determinism and higher reliability of HAC with the efficiency of *K*-means.

BUCKSHOT ALGORITHM

17.9 References and further reading

An excellent general review of clustering is (Jain et al. 1999). Early references for specific HAC algorithms are (King 1967) (single-link), (Sneath and Sokal 1973) (complete-link, GAAC) and (Lance and Williams 1967) (discussing a large variety of hierarchical clustering algorithms). The single-link algorithm in Figure 17.9 is similar to *Kruskal's algorithm* for constructing a minimum spanning tree. A graph-theoretical proof of the correctness of Kruskal's algorithm (which is analogous to the proof in Section 17.5) is provided by Cormen et al. (1990, Theorem 23.1). See Exercise 17.5 for the connection between minimum spanning trees and single-link clusterings.

KRUSKAL'S ALGORITHM

It is often claimed that hierarchical clustering algorithms produce better clusterings than flat algorithms (Jain and Dubes (1988, p. 140), Cutting et al. (1992), Larsen and Aone (1999)) although more recently there have been experimental results suggesting the opposite (Zhao and Karypis 2002). Even without a consensus on average behavior, there is no doubt that results of EM and *K*-means are highly variable since they will often converge to a local optimum of poor quality. The HAC algorithms we have presented here are deterministic and thus more predictable.

The complexity of complete-link, group-average and centroid clustering is sometimes given as $\Theta(N^2)$ (Day and Edelsbrunner 1984, Voorhees 1985b, Murtagh 1983) because a document similarity computation is an order of magnitude more expensive than a simple comparison, the main operation executed in the merging steps after the $N \times N$ similarity matrix has been computed.

The centroid algorithm described here is due to Voorhees (1985b). Voorhees recommends complete-link and centroid clustering over single-link for a retrieval application. The Buckshot algorithm was originally published by Cutting et al. (1993). Allan et al. (1998) apply single-link clustering to first story detection.

WARD'S METHOD

An important HAC technique not discussed here is *Ward's method* (Ward Jr. 1963, El-Hamdouchi and Willett 1986), also called *minimum variance clustering*. In each step, it selects the merge with the smallest RSS (Chapter 16, page 360). The merge criterion in Ward's method (a function of all individual distances from the centroid) is closely related to the merge criterion in GAAC (a function of all individual similarities to the centroid).

Despite its importance for making the results of clustering useful, comparatively little work has been done on labeling clusters. Popescul and Ungar (2000) obtain good results with a combination of χ^2 and collection frequency of a term. Glover et al. (2002b) use information gain for labeling clusters of web pages. Stein and zu Eissen's approach is ontology-based (2004). The more complex problem of labeling nodes in a hierarchy (which requires distinguishing more general labels for parents from more specific labels for children) is tackled by Glover et al. (2002a) and Treeratpituk and Callan (2006). Some clustering algorithms attempt to find a set of labels first and then build (often overlapping) clusters around the labels, thereby avoiding the problem of labeling altogether (Zamir and Etzioni 1999, Käki 2005, Osiński and Weiss 2005). We know of no comprehensive study that compares the quality of such "label-based" clustering to the clustering algorithms discussed in this chapter and in Chapter 16. In principle, work on multi-document summarization (McKeown and Radev 1995) is also applicable to cluster labeling, but multi-document summaries are usually longer than the short text fragments needed when labeling clusters (cf. Section 8.7, page 170). Presenting clusters in a way that users can understand is a UI problem. We recommend reading (Baeza-Yates and Ribeiro-Neto 1999, ch. 10) for an introduction to user interfaces in IR.

SPECTRAL CLUSTERING

An example of an efficient divisive algorithm is bisecting K-means (Steinbach et al. 2000). *Spectral clustering* algorithms (Kannan et al. 2000, Dhillon 2001, Zha et al. 2001, Ng et al. 2001a), including *principal direction divisive partitioning* (PDDP) (whose bisecting decisions are based on SVD, see Chapter 18) (Boley 1998, Savaresi and Boley 2004), are computationally more expensive than bisecting K-means, but have the advantage of being deterministic.

Unlike K-means and EM, most hierarchical clustering algorithms do not have a probabilistic interpretation. Model-based hierarchical clustering (Vaithyanathan and Dom 2000, Kamvar et al. 2002, Castro et al. 2004) is an exception.

The evaluation methodology described in Section 16.3 (page 356) is also applicable to hierarchical clustering. Specialized evaluation measures for hierarchies are discussed by Fowlkes and Mallows (1983), Larsen and Aone (1999) and Sahoo et al. (2006).

The R environment (R Development Core Team 2005) offers good support for hierarchical clustering. The R function `hclust` implements single-link, complete-link, group-average, and centroid clustering; and Ward's method. Another option provided is `median` clustering which represents each cluster by its medoid (cf. k-medoids in Chapter 16, page 365). Support for clustering vectors in high-dimensional spaces is provided by the software package CLUTO (http://glaros.dtc.umn.edu/gkhome/views/cluto).

17.10 Exercises

?

MINIMUM SPANNING TREE

Exercise 17.5

A single-link clustering can also be computed from the *minimum spanning tree* of a graph. The minimum spanning tree connects the vertices of a graph at the smallest possible cost, where cost is defined as the sum over all edges of the graph. In our case the cost of an edge is the distance between two documents. Show that if $\Delta_{k-1} > \Delta_k > \ldots > \Delta_1$ are the costs of the edges of a minimum spanning tree, then these edges correspond to the $k-1$ merges in constructing a single-link clustering.

Exercise 17.6

Show that single-link clustering is best-merge persistent and that GAAC and centroid clustering are not best-merge persistent.

Exercise 17.7

a. Consider running 2-means clustering on a collection with documents from two different languages. What result would you expect?

b. Would you expect the same result when running an HAC algorithm?

Exercise 17.8

Download Reuters-21578. Keep only documents that are in the classes *crude*, *interest*, and *grain*. Discard documents that are members of more than one of these three classes. Compute a (i) single-link, (ii) complete-link, (iii) GAAC, (iv) centroid clustering of the documents. (v) Cut each dendrogram at the second branch from the top to obtain $K = 3$ clusters. Compute the Rand index for each of the 4 clusterings. Which clustering method performs best?

Exercise 17.9

Suppose a run of HAC finds the clustering with $K = 7$ to have the highest value on some prechosen goodness measure of clustering. Have we found the highest-value clustering among all clusterings with $K = 7$?

Exercise 17.10

Consider the task of producing a single-link clustering of N points on a line:

Show that we only need to compute a total of about N similarities. What is the overall complexity of single-link clustering for a set of points on a line?

Exercise 17.11

Prove that single-link, complete-link, and group-average clustering are monotonic in the sense defined on page 378.

Exercise 17.12

For N points, there are $\leq N^K$ different flat clusterings into K clusters (Section 16.2, page 356). What is the number of different hierarchical clusterings (or dendrograms) of N documents? Are there more flat clusterings or more hierarchical clusterings for given K and N?