Hierarchical clustering

Flat clustering is efficient and conceptually simple, but as we saw in Chapter 16 it has a number of drawbacks. It returns a flat unstructured set of clusters; it requires a prespecified number of clusters as input; and the flat clustering algorithms k-means and EM are not guaranteed to find the optimal set of clusters. Hierarchical clustering (or hierarchic clustering) outputs a hierarchy, a structure that is more informative than the unstructured set of clusters in flat clustering. It does not require us to prespecify the number of clusters. And most hierarchical algorithms find the optimal solution. These advantages of hierarchical clustering come at the cost of lower efficiency. 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.

This chapter first introduces agglomerative hierarchical clustering (Section 17.1). We then present four different agglomerative algorithms, in Sections 17.2–17.4, which differ in the similarity measure they employ: single-link, complete-link, group-average, and centroid clustering. Section 17.5 looks at automatic labeling of clusters, which is important whenever humans interact with the output of clustering. Finally, we discuss variants of hierarchical clustering algorithms and implementation issues in Sections 17.6 and 17.7.

In principle, the possible applications of flat and hierarchical clustering in information retrieval differ little. In particular, hierarchical clustering is appropriate for any of the applications shown in Table 16.1 (page 253) (see also Section 16.6, page 270). 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 of flat clustering (not enough structure, predetermined number of clusters, no guarantee of optimality) is a concern. In addition, many researchers believe that hierarchical clustering produces better clusters than flat clustering on either internal or external criteria (Section 16.3, page 257). But there is no consensus on this issue (see references in Section 17.8).
17 Hierarchical clustering

17.1 Hierarchical agglomerative clustering

Hierarchical algorithms are either top-down or bottom-up. Bottom-up algorithms merge or agglomerate documents and clusters into larger and larger units. Bottom-up clustering is therefore called hierarchical agglomerative clustering or HAC. Top-down clustering requires a method for splitting a cluster and proceeds by splitting clusters recursively until individual documents are reached (see Section 17.6). HAC is more frequently used than top-down clustering and is the main subject of this chapter.

HAC treats each document as a singleton cluster at the outset and then successively merges pairs of clusters until all clusters have been merged into a single cluster that contains all documents.

An HAC clustering is typically visualized as a dendrogram as shown in Figure 17.1. A merge of two clusters is represented as a horizontal line that connects the vertical lines of the two clusters (where documents are viewed as singleton clusters). The y-axis represents combination similarity, the similarity of the two clusters merged by the horizontal line at a particular y. By moving up from the bottom layer to the top node, we can reconstruct the history of mergers that resulted in the depicted clustering.

A fundamental assumption in HAC is that the merge operation is monotonic. If $s_1, s_2, \ldots, s_{K-1}$ are the successive combination similarities of an HAC clustering, then $s_1 \geq s_2 \geq \ldots \geq s_{K-1}$ must hold. A non-monotonic HAC clustering contains at least one inversion $s_i < s_{i+1}$ and contradicts the fundamental assumption that we found the best possible merger at each step. We will see an example of an inversion in Figure 17.11.

Hierarchical clustering does not require a prespecified number of clusters. But in some applications we want a partition of disjoint clusters just as 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 may want clusters with a minimum combination similarity of 0.4. In this case, we cut the dendrogram at 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 can be viewed as optimal. This strategy is analogous to looking for the knee in the k-means graph in Figure 16.8 (page 264).
A dendrogram of a single-link clustering of 30 documents from Reuters-RCV1. The y-axis represents combination similarity, the similarity of the two component clusters that gave rise to the corresponding merge. For example, the combination similarity of Lloyd’s CEO questioned and Lloyd’s chief / U.S. grilling is ≈ 0.56. Two possible cuts of the dendrogram are shown at 0.4 into 24 clusters and at 0.1 into 12 clusters.
**Given:** \( N \) one-document clusters  

**Compute similarity matrix**  
for \( k = 1 \) to \( N \):  
for \( \ell = 1 \) to \( N \):  
\[
C[k][\ell] = \text{sim}(d_k, d_\ell)
\]

**Initialization**  
\( A = [] \) (for collecting merge sequence)  
for \( k = 1 \) to \( N \):  
\( I[k] = 1 \) (keeps track of active clusters)

**Compute clustering**  
for \( k = 1 \) to \( N - 1 \):  
\[
(\ell, m) = \arg \max_{\ell \neq m, I[\ell] = I[m]} C[\ell][m]
\]
\( A\_\text{append}(\langle \ell, m \rangle) \)
for \( j = 1 \) to \( N \):  
\[
C[\ell][j] = C[j][\ell] = \text{sim}(j, \ell, m)
\]
\( I[m] = 0 \) (deactivate cluster)

**Figure 17.2** A simple, but inefficient HAC algorithm. In each iteration, the two most similar clusters are merged and the rows and columns of the merge cluster \( \ell \) in \( C \) are updated. Ties in this algorithm and in HAC in general are broken randomly. The clustering is stored as a list of mergers in \( A \). \( I \) indicates which clusters are still available to be merged. The function \( \text{sim}(j, \ell, m) \) computes the similarity of cluster \( j \) with the merger of clusters \( \ell \) and \( m \). For some HAC algorithms \( \text{sim}(j, \ell, m) \) is simply a function of \( C[j][\ell] \) and \( C[j][m] \), for example, the maximum of these two values for single-link.

- Evaluate each of the \( N \) clusterings with respect to a goodness measure \( g \). Select the cutting point corresponding to the clustering with optimal goodness. We first need a goodness measure \( g \) for clusters. An example for \( g \) is \((- \text{RSS}) \) (Chapter 16, page 260). The goodness measure of the clustering is then the sum of (a) the weighted average of \( g(\omega_1), \ldots, g(\omega_K) \) and (b) a penalty for the number of clusters. We need the penalty, because the goodness measure reaches its maximum (0.0 for \((- \text{RSS}) \) for \( N \) one-document clusters. This strategy is similar to AIC and BIC in Chapter 16 (page 265).

- As in flat clustering, we can also prespecify the number of clusters \( K \) and select the cutting point that produces \( K \) clusters.

A simple, generic HAC algorithm is shown in Figure 17.2. It consists of \( N - 1 \) steps of merging the currently most similar clusters. We will now refine this algorithm for the different similarity measures of single-link, complete-link, group-average, and centroid clustering. Single-link clustering merges
### Table 17.1  Comparison of HAC algorithms.

<table>
<thead>
<tr>
<th>Method</th>
<th>Combination similarity</th>
<th>Time compl.</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-link</td>
<td>max sim of any two points</td>
<td>$O(N^2)$</td>
<td>chaining effect</td>
</tr>
<tr>
<td>Complete-link</td>
<td>min sim of any two points</td>
<td>$O(N^2 \log N)$</td>
<td>sensitive to outliers</td>
</tr>
<tr>
<td>Centroid</td>
<td>similarity of centroids</td>
<td>$O(N^2 \log N)$</td>
<td>combination similarity, not recoverable, inversions possible</td>
</tr>
<tr>
<td>Group-average</td>
<td>avg sim of any two points</td>
<td>$O(N^2 \log N)$</td>
<td>optimal in most cases</td>
</tr>
</tbody>
</table>

The two clusters that have the greatest *maximum* similarity between any two members; complete-link clustering merges the two clusters that have the greatest *minimum* similarity between any two members (Section 17.2). Group-average and centroid clustering average similarities between members and thereby avoid the long straggling clusters in single-link clustering (the chaining effect) and the sensitivity to outliers in complete-link clustering (Sections 17.3 and 17.4).

Table 17.1 summarizes the properties of the four HAC algorithms introduced in this chapter. We recommend GAAC because it is generally the method that produces the highest quality clustering. It does not suffer from chaining or sensitivity to outliers and has cleaner semantics than centroid clustering. The only exception to this recommendation is for non-vector representations. In that case, GAAC is not applicable and clustering should typically be performed with the complete-link method.

### 17.2 Single-link and complete-link clustering

In **single-link clustering** or **single-linkage clustering**, the similarity of two clusters is the similarity of their **most similar** members. 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.

In **complete-link clustering** or **complete-linkage clustering**, the similarity of two clusters is the similarity of their **most dissimilar** members. This is equivalent to choosing the cluster pair whose merger has the smallest diameter. This complete-link merge criterion is non-local: the entire structure of the clustering is taken into account. We prefer compact clusters with small diameters over long, straggly clusters. Complete-link clustering is sensitive to outliers. A single document far from the center can increase diameters of candidate merge clusters dramatically and completely change the final clustering.

Figure 17.3 depicts a single-link and a complete-link clustering of eight...
Figure 17.3 A single-link (top) and complete-link (bottom) clustering of eight documents. The ellipses correspond to successive clustering stages. Top: The single-link similarity of the two upper two-point clusters is the similarity (proximity) of \( d_2 \) and \( d_3 \) (solid line), which is greater than the single-link similarity of the two left pairs (dashed line). Bottom: The complete-link similarity of the two upper two-point clusters is the similarity (proximity) of \( d_1 \) and \( d_4 \) (dashed line), which is smaller than the complete-link similarity of the two left pairs (solid line).

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-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. 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.
17.2 Single-link and complete-link clustering

The y-axis represents combination similarity.

When cutting the last edge of a complete-link clustering, we obtain two clusters of similar size. When cutting the last edge of a single-link clustering, we obtain clusters of the same size (documents 1-16 and documents 17-30). The y-axis represents combination similarity.

Figure 17.4: A dendrogram of a complete-link clustering of 30 documents from Reuters-RCV1. This complete-link clustering is more balanced than the single-link clustering of the same documents in Figure 17.1.

Lloyd's CEO questioned
Mexican markets
Fed to keep interest rates steady
Fed keeps interest rates steady
Mexican Prime minister / Mexican
Clinton signs law
American Tobacco
Fed to keep interest rates steady
Fed keeps interest rates steady
Fed to keep interest rates steady
Fed keeps interest rates steady
Fed to keep interest rates steady
Fed keeps interest rates steady
Hierarchical clustering

Figure 17.5 Chaining in single-link clustering. The local criterion in single-link clustering can cause undesirable elongated clusters.

set. Single-link clustering is used in Section 19.6.1 (page 321), as part of the union-find algorithm, to identify near duplicate pages on the web.

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 the maximum cliques of $G(s_k)$.

These graph-theoretic interpretations motivate the terms single-link and complete-link clustering. Single-link clusters at step $k$ are maximum sets of points that are linked via at least one link of similarity $s \geq s_k$; complete-link clusters at step $k$ are maximum sets of points that are completely linked among 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, 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 types of clustering are suboptimal. Single-link clustering can produce straggling clusters as shown in Figure 17.5. Since the merge criterion is strictly local, a chain of points can be extended for long 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 12 mergers of the single-link clustering (those above the 0.1 ($= 1 - 0.9$) line) add on single documents or pairs of documents, corresponding to a chain. The complete-link clustering in Figure 17.4 avoids this problem. Documents are split into two groups of roughly equal size. In general, this is a more useful organization of the data than a clustering with chains.

But complete-link clustering has a different problem. It pays too much attention to outliers, points that do not fit well into the global structure of the data. This is a consequence of the way the clusters are formed.

---

2. A connected component is a maximum set of points such that there is a path connecting each pair. A clique is a set of points that are completely linked among each other.
17.2 Single-link and complete-link clustering

![Figure 17.6](image)

*Figure 17.6* Outliers in complete-link clustering. The four points have the coordinates $-3 + 2 \times \epsilon, 0 + 2 \times \epsilon, 2$ and $3 - \epsilon$. Complete-link clustering creates the two clusters shown as ellipses. Intuitively, $\{b, c, d, e\}$ should be one cluster, but it is split by outlier $a$.

the cluster. In the example in Figure 17.6 the four rightmost points are split because of the outlier at the left edge. Complete-link clustering does not find the intuitively correct cluster structure in this example.

17.2.1 Time complexity

The complexity of the generic HAC algorithm in Figure 17.2 is $O(N^3)$ because we exhaustively search matrix $C$ for the largest similarity (which then gives us the next pair to merge). But in single-link clustering, we can do much better by keeping a next-best-merge array (NBM) as shown in Figure 17.7. NBM keeps track of what the best merge is for each cluster. Each of the three $k$-loops in Figure 17.7 and thus the overall complexity of single-link clustering is $O(N^2)$.

The time complexity of complete-link clustering is $O(N^2 \log N)$ as shown in Figure 17.8. The rows of the $N \times N$ matrix $C$ are sorted in decreasing order of similarity in the priority queues $P$. $P[k].\text{max}()$ then returns the element of $C[k]$ that currently has the highest similarity with $k$. After creating the merged cluster of $k_1$ and $k_2$, $k_1$ is used as its representative. The function sim computes the similarity function for potential merger pairs: largest similarity for single-link, smallest similarity for complete-link, average similarity for GAAC (Section 17.3), and centroid similarity for centroid (Section 17.4). We give an example of how a row of $C$ is processed (Figure 17.8, bottom panel). The first $k$-loop is $O(N^2)$, the second and third are $O(N^2 \log N)$ for an implementation of priority queues that supports deletion and insertion in $O(\log N)$. Overall complexity of the algorithm is therefore $O(N^2 \log N)$.

In the definition of the merge functions, $\vec{v}_m$ and $\vec{v}_t$ are the vector sums of $\omega_{k_1} \cup \omega_{k_2}$ and $\omega_{t}$, respectively, and $N_m$ and $N_t$ are the number of documents in $\omega_{k_1} \cup \omega_{k_2}$ and $\omega_{t}$, respectively.
Given: \(N\) one-document clusters

**Compute similarity matrix**

for \(k = 1\) to \(N\):
  for \(\ell = 1\) to \(N\):
    \(C[k][\ell] = \text{sim}(d_k, d_\ell)\)

**Initialization**

\(A = []\)

for \(k = 1\) to \(N\):
  \(I[k] = 1\)
  \(\text{NBM}[k].\text{index} = \arg \max_{i \neq k} C[k][i]\)
  \(\text{NBM}[k].\text{sim} = C[k][\text{NBM}[k].\text{index}]\)

**Compute clustering**

for \(k = 1\) to \(N - 1\):
  \(k_1 = \arg \max_{k, i | k = 1} \text{NBM}[k].\text{sim}\)
  \(k_2 = \text{NBM}[k_1].\text{index}\)
  \(A.\text{append}((k_1, k_2))\)
  \(k_{\min} = \arg \min_{k_1, k_2} (\text{NBM}[k_1].\text{sim}, \text{NBM}[k_2].\text{sim})\)
  \(I[k_{\min}] = 0\)

> **Figure 17.7** Single-link clustering algorithm using an NBM array.

What is the reason for the difference in time complexity between single-link and complete-link clustering? Single-link clustering is *best-merge persistent*. Suppose that the best merge cluster for \(\omega_k\) is \(\omega_j\). Then after merging \(\omega_j\) with a third cluster \(\omega_i \neq \omega_k\), the merger of \(\omega_i\) and \(\omega_j\) will be \(\omega_k\)'s best merge cluster (Exercise 17.2). As a consequence, 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 no update of NBM is necessary after a merger in Figure 17.7.

Figure 17.9 demonstrates that best-merge persistence does not hold for complete-link clustering. After merging \(\omega_k\)'s best merge candidate \(\omega_j\) with the third cluster \(\omega_i \neq \omega_k\), a completely different cluster \(\omega_\ell\) becomes the best merge candidate for \(\omega_k\). This is because the complete-link merge criterion is global and can be affected by points at a great distance from the area where two merge candidates meet.

In practice, the performance penalty of the \(O(N^2 \log N)\) algorithm is small compared to the \(O(N^2)\) single-link algorithm since the computation of one dot product is an order of magnitude slower than the computation of a comparison in sorting. All the HAC algorithms we present are \(O(N^2)\) with respect to dot product computations. So the difference in complexity is rarely a concern in practice when choosing one of the HAC algorithms.
Given: $N$ normalized vectors $\vec{v}_i$

Compute matrix $C$
for $k = 1$ to $N$:
  for $\ell = 1$ to $N$:
    $C[k][\ell].\text{sim} = \vec{v}_k \cdot \vec{v}_\ell$
    $C[k][\ell].\text{index} = \ell$
for $k = 1$ to $N$:
  $P[k] :=$ priority queue for $C[k]$ sorted on sim
  Delete $C[k][k]$ from $P[k]$ (don't want self-similarities)

Initialization
$A = []$
for $k = 1$ to $N$:
  $I[k] = 1$

Compute clustering
for $k = 1$ to $N - 1$:
  $k_1 = \arg\max_i I[i] = 1$ $P[k].\text{max}()$
  $k_2 = P[k_1].\text{max}().\text{index}$
  $A.$append$((k_1, k_2))$
  $I[k_2] = 0$
  $P[k_1] = \varnothing$
  for all $\ell$ with $I[\ell] = 1, \ell \neq k_1$:
    $C[\ell][k_1].\text{sim} = C[k_1][\ell].\text{sim} = \text{sim}(\ell, k_1, k_2)$
    Delete $C[\ell][k_1]$ and $C[\ell][k_2]$ from $P[\ell]$
    Insert $C[\ell][k_1]$ in $P[\ell]$, $C[k_1][\ell]$ in $P[k_1]$

<table>
<thead>
<tr>
<th>clustering algorithm</th>
<th>sim($\ell, k_1, k_2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>single-link</td>
<td>max(sim($\ell, k_1$), sim($\ell, k_2$))</td>
</tr>
<tr>
<td>complete-link</td>
<td>min(sim($\ell, k_1$), sim($\ell, k_2$))</td>
</tr>
<tr>
<td>centroid</td>
<td>$(\frac{1}{N_m} \bar{\vec{v}}<em>m) \cdot (\frac{1}{N</em>\ell} \bar{\vec{v}}_\ell)$</td>
</tr>
<tr>
<td>group-average</td>
<td>$\frac{1}{(N_m + N_\ell)(N_m + N_\ell - 1)} [(\bar{\vec{v}}<em>m + \bar{\vec{v}}</em>\ell)^2 - (N_m + N_\ell)]$</td>
</tr>
</tbody>
</table>

compute $C[5]$
create $P[5]$ (by sorting)
merge 2 and 3, update similarity of 2, delete 3
reinsert 2

Figure 17.8 A generic HAC algorithm. Top: The algorithm. Center: Four different similarity measures. Bottom: An example of a sequence of processing steps for a priority queue. This is a made up example showing $P[5]$ for a $5 \times 5$ matrix $C$. 

Preliminary draft (c) 2007 Cambridge UP
17 Hierarchical clustering

Figure 17.9 Complete-link clustering is not best-merge persistent. At first, \( \omega'_j \) is the best-merge cluster for \( \omega_k \). But after merging \( \omega_i \) and \( \omega_j \), \( \omega_\ell \) becomes \( \omega_k \)'s best-merge candidate. In a best-merge persistent algorithm like single-link, the best-merge cluster would be \( \omega_i \cup \omega_j \).

17.3 Group-average agglomerative clustering

For clustering in a vector space, there is a clustering method that evaluates cluster quality based on all similarities between documents, thus avoiding the pitfalls of the single-link and complete-link criteria: group-average agglomerative clustering or GAAC. It is also called group-average clustering and average-link clustering. GAAC computes the average similarity \( \text{sim-ga} \) of all pairs, including pairs of documents from the same cluster. Self-similarities are not included in the average.

\[
\text{sim-ga}(\omega_i, \omega_j) = \frac{1}{(N_i + N_j)(N_i + N_j - 1)} \sum_{d_k \in \omega_i \cup \omega_j} \sum_{d_\ell \in \omega_i \cup \omega_j, d_\ell \neq d_k} \vec{d}_k \cdot \vec{d}_\ell
\]

where \( \vec{d} \) is the normalized vector of document \( d \), \( \cdot \) denotes the scalar or dot product, and \( N_i \) and \( N_j \) are the number of documents in \( \omega_i \) and \( \omega_j \), respectively.

The measure \( \text{sim-ga} \) can be computed efficiently because the sum of individual vector similarities is equal to the similarities of their sums:

\[
\sum_{d_k \in \omega_i} \sum_{d_\ell \in \omega_j} (\vec{d}_k \cdot \vec{d}_\ell) = (\sum_{d_k \in \omega_i} \vec{d}_k) \cdot (\sum_{d_\ell \in \omega_j} \vec{d}_\ell)
\]

With (17.2), we have:

\[
\text{sim-ga}(\omega_i, \omega_j) = \frac{1}{(N_i + N_j)(N_i + N_j - 1)} \left[ (\sum_{d_k \in \omega_i \cup \omega_j} \vec{d}_k)^2 - (N_i + N_j) \right]
\]

The term \((N_i + N_j)\) 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 \( \sum \vec{d} \) instead of in \( O(N_i N_j) \). Note that for two single-document clusters, 17.3 is equivalent to the dot product.

Equation (17.2) relies on the distributivity of the scalar 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 non-vector representations of documents.

The merge algorithm for GAAC is the same as Figure 17.8 for complete-link except that we use as the similarity function 17.3. So the overall time complexity of GAAC is the same as for complete-link clustering: \( O(N^2 \log N) \). Like complete-link clustering, GAAC is not best-merge persistent (Exercise 17.2). This means that there is no \( O(N^2) \) algorithm for GAAC that would be analogous to the \( O(N^2) \) algorithm for single-link.

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

\[
\text{sim-ga}'(\omega_i, \omega_j) = \frac{1}{(N_i + N_j)^2} \left( \sum_{d_k \in \omega_i \cup \omega_j} \vec{d}_k \right)^2 = \frac{1}{N_i + N_j} \sum_{d_k \in \omega_i \cup \omega_j} \vec{d}_k \cdot \vec{\mu}(\omega_i \cup \omega_j)
\]

where the centroid \( \vec{\mu}(\omega) \) is defined as in Equation (14.1) (page 220). This definition is equivalent to the intuitive definition of cluster quality as average similarity of documents \( \vec{d}_k \) to the cluster’s centroid \( \vec{\mu}_k \). Self-similarities are always equal to 1.0, the maximum possible value for 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 with a similarity \( s \), this means that merging them generates a cluster with a similarity measure of \( (1 + s)/2 \). The sim-ga similarity of the combined clusters according to Equation (17.3) is \( s \leq (1 + s)/2 \), which is the same as in single-link, complete-link and centroid clustering. For these reasons, we prefer the definition in Equation (17.3), which excludes self-similarities from the average.

### 17.4 Centroid clustering

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

\[
\text{sim-cent}(\omega_i, \omega_j) = \left( \frac{1}{N_i} \sum_{d_k \in \omega_i} \vec{d}_k \right) \cdot \left( \frac{1}{N_j} \sum_{d_\ell \in \omega_j} \vec{d}_\ell \right) = \frac{1}{N_i N_j} \sum_{d_k \in \omega_i} \sum_{d_\ell \in \omega_j} \vec{d}_k \cdot \vec{d}_\ell
\]
Figure 17.10  Centroid clustering. Each step merges the two clusters whose centroids are closest as measured by the dot product.

that GAAC considers all pairs of documents in computing average pairwise similarity whereas centroid clustering excludes pairs from the same cluster.

Figure 17.10 shows two iterations of centroid clustering.

Like GAAC, centroid clustering is not best-merge persistent and therefore \( O(N^2 \log N) \) (Exercise 17.2). But centroid clustering has one unique property that makes it inferior to the other three HAC algorithms. A good HAC algorithm creates the optimal cluster \( \omega \) in each step for some criterion of optimality. For single-link, complete-link and group-average clustering, these optimality criteria directly correspond to the combination similarity of the two components that were merged into \( \omega \): maximum distance of any point to its nearest neighbor in \( \omega \) (single-link), maximum distance of any two points in \( \omega \) (complete-link), and average of all pairwise similarities in \( \omega \) (GAAC). So there is equivalence for these three methods between the combination similarity computed on the two components and an optimality criterion computed directly on \( \omega \) (Exercise 17.3). But there is no such optimality score computable directly on the merged cluster \( \omega \) for centroid clustering: each split of \( \omega \) will in general produce a different similarity value of the centroids of the two parts. This is worrying because it means that centroid clustering makes no guarantees that the clusters it creates have good properties.

Centroid clustering does not have an optimality criterion because it is not monotonic. So-called inversions can occur: Similarity can increase during clustering as in the example in Figure 17.11. This means that we cannot be sure that after merging two clusters with similarity \( s \) in centroid clustering, we have found all clusters whose “quality” is better than \( s \).

Despite this problem, centroid clustering is often used because its similarity measure – the similarity of two centroids – is conceptually simpler.
Figure 17.11  Centroid clustering is not monotonic. The points a at \((1 + \varepsilon, 1)\), b at \((5, 1)\), and c at \((3, 1 + 2\sqrt{3})\) are almost equidistant, with a and b closer to each other than to c. In the first merger, the proximity of a and b is \(\approx 4\). In the second merger, the proximity of the centroid of a and b (the circle) and c is \(\approx \cos(\pi/6) \times 4 = \sqrt{3}/2 \times 4 \approx 3.46 < 4\). 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.

than the average of all pairwise similarities in GAAC. Figure 17.10 is all one needs to understand centroid clustering. There is no equally simple graph that would explain how GAAC works.

17.5 Cluster labeling

In many applications of clustering, particularly in analysis tasks and in user interfaces (see applications in Table 16.1, page 253), human users interact with clusters. In such settings, we must label clusters, so that users can see what a cluster is about. In Scatter-Gather (Figure 16.3, page 255), each cluster is represented by titles of typical documents – those closest to the cluster centroid – and by a list of words with high weights in the centroid of the cluster. Titles are natural cluster labels because they are written by authors as one-line summaries that can be quickly scanned. Highly weighted words (or, even better, phrases, especially noun phrases) are often more representative of the cluster as a whole than a few titles can be. But a list of phrases takes more time to digest for users than a well crafted title. On the web, anchor text can play a role similar to a title since the anchor text pointing to a page can serve as a concise summary of its contents.

Consider the representation of the Germany cluster in one Scatter-Gather experiment:
### Table 17.2

Automatically computed cluster labels for a k-means clustering ($K = 10$) of the first 10,000 documents in Reuters-RCV. Only three of the ten clusters (4, 9, and 10) are shown. The three last columns show cluster summaries computed by three labeling methods: most highly weighted words in centroid (centroid), mutual information, and the title of the document closest to the centroid of the cluster (title). Words selected by only one of the first two methods are in bold.

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

The two labeling methods applied here (titles of typical documents and words prominent in the centroid) are **cluster-internal**. Another cluster-internal method simply selects the most frequent words in the cluster. Cluster-internal methods are efficient, but they fail to distinguish words that are frequent in the collection as a whole from those that are frequent only in the cluster. The word *year* has a high frequency both in the cluster and in the collection and is therefore not helpful in understanding the contents of the Germany cluster.

**Differential cluster labeling** selects cluster labels by comparing the distribution of words in one cluster with that of other clusters. The feature selection methods we introduced in Section 13.5 (page 204) can all be used for differential cluster labeling. In fact, selecting the most frequent words is also a feature selection technique we discussed in Section 13.5, albeit a non-differential one. In particular, mutual information (MI) (Section 13.5.1, page 204) or, equivalently, information gain and the $\chi^2$-test (Section 13.5.2, page 206) will identify cluster labels that characterize one cluster in contrast to other clusters. A combination of a differential test with a penalty for rare words often gives the best labeling results because rare words 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 was almost no difference between MI and $\chi^2$. We therefore omit the latter. The centroid method selects a few more uninformative words (food, court, cents, september) than MI (forces, desk), but most of the words selected by either method are good descriptors. We get a good sense of the documents in a cluster from scanning the selected words.

The title of the document closest to the centroid provides a complementary way of characterization. It is easier to read than a list of words. A full title can also contain important context that didn’t make it into the top 10 terms selected by MI. The title for cluster 9 suggests that many of its documents are about the Chechnya conflict, a fact the MI words do not reveal. But a single document cannot be representative of all documents in a cluster as in the case of cluster 4, where the 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.

Additional complications in labeling clusters arise when we want to label a cluster hierarchy instead of a flat clustering. Not only do we need to distinguish an internal node 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 a naive differential method cannot be used 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.

17.6 Variants

So far we’ve 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 algorithm. This procedure is applied recursively until the desired number of clusters has been computed or some other stopping criterion is met.

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 don’t generate a 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 $O(N^2)$ or $O(N^2 \log N)$ bottom-up algorithms.

There is evidence that divisive algorithms produce more accurate hierarchies than bottom-up algorithms in some circumstances (see reference to
Hierarchical clustering

Another combination of flat and hierarchical clustering uses hierarchical clustering to create good seeds for k-means. K-means requires a set of seeds as initialization (Figure 16.5, page 261). If these seeds are badly chosen, then the resulting clustering will be of poor quality. If the HAC algorithm is applied to a subset of size \( O(\sqrt{N}) \) to generate \( K \) seeds for k-means, then the overall run time of k-means cum HAC seed generation is \( O(N) \). This is because the application of a quadratic algorithm to a sample of size \( O(\sqrt{N}) \) has an overall complexity of \( O(N) \). An appropriate adjustment can be made for an \( O(N^2 \log N) \) algorithm to guarantee linearity. This algorithm is referred to as the Buckshot algorithm.

**17.7 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 don’t share any words 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 17.7). Because of the curse of dimensionality, no more efficient algorithms are known in higher dimensions. We confronted the same problem in kNN classification (see Section 14.4, page 234).

When computing a GAAC of a large document set in high dimensions, we have to take care to avoid dense centroids. For dense centroids, clustering can take time \( O(|V|N^2 \log N) \) where \(|V|\) is the size of the vocabulary whereas complete-link clustering is \( O(M_dN^2 \log N) \) where \( M_d \) is the average vocabulary of a document. So for large vocabularies complete-link 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 263) and suggested two solutions: truncating centroids (keeping only highly weighted terms) and representing clusters by means of sparse medoids instead of dense centroids. This optimization can also be applied to GAAC and centroid clustering.
17.8 References and further reading

An excellent general review of clustering is (Jain et al. 1999). Early references 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). A reference for minimum spanning tree algorithms (as an alternative way of computing a single-link hierarchy, Exercise 17.1) is (Cormen et al. 1990).

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 occasionally converge to a local optimum of poor quality. Assuming ties are handled appropriately, the hierarchical algorithms we have presented here are deterministic and thus more reliable.

The complexity of complete-link, group-average and centroid clustering is sometimes given as $O(N^2)$ (Day and Edelsbrunner 1984, Voorhees 1985b) or even as less than quadratic (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).

An important HAC technique not discussed here is Ward’s method (Ward 1963, El-Hamdouchi and Willett 1986), also called minimum variance clustering. In each step, it selects the merger with the smallest RSS (the sum of all squared distances of each point from its centroid), as defined in Chapter 16 (page 260). 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 from a combination of $\chi^2$ and term frequency. Glover et al. (2002b) use information gain for labeling clusters of web pages. 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). We
know of no comprehensive study that evaluates the quality of such “label-
based” clusters compared to the clustering algorithms discussed here and in
Chapter 16. A problem related to cluster labeling occurs in marking areas
of a self-organizing map (or Kohonen map) with typical words or phrases
(Azcarraga and Yap Jr. 2001). In principle, work on multi-document sum-
marization (McKeown and Radev 1995) is also applicable to cluster labeling,
but multi-document summaries are usually longer than the short text needed
when labeling clusters.

The bisecting k-means algorithm is due to Steinbach et al. (2000).

Although this introduction to hierarchical clustering focuses on document
clustering, hierarchical clustering can be readily applied to clustering other
objects in information retrieval. For example, terms can be clustered in docu-
ment space, where documents are axes and terms are represented as vectors
in document space (Qiu and Frei 1993).

Unlike k-means and EM, most hierarchical clustering algorithms do not
have a probabilistic interpretation. Model-based hierarchical clustering (Vaithyanathan
and Dom 2000) is an exception.

The evaluation methodology described in Section 16.3 (page 257) is also
applicable to hierarchical clustering. Specialized evaluation measures for hi-
erarchies are discussed by Larsen and Aone (1999) and Sahoo et al. (2006).

R (R Development Core Team 2005) offers good support for hier archi-
clustering. The R function hclust implements single-link (single),
complete-link (complete), group-average (average), and centroid (centroid)
clustering; and Ward’s method (ward). Another option provided is median
clustering which represents each cluster by its medoid (cf. k-medoids in Chap-
ter 16, page 263).

17.9 Exercises

Exercise 17.1

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 lengths of the edges of a minimum spanning tree, then these edges
correspond to the \( k-1 \) merges in constructing a single-link clustering.

Exercise 17.2

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

Exercise 17.3

Given a cluster \( \omega \) that was produced in a) single-link, b) complete-link, c) group-
average clustering, how can one compute the combination similarity of \( \omega \)? (One way
Figure 17.12  Single-link clustering of points on a line. In low dimensions, single-link clustering can be implemented more efficiently than the general algorithm in Figure 17.2.

to do this is to restart the clustering and keep track of the combination similarities. The intent of the exercise is to read the combination similarity directly off of the cluster.)

Exercise 17.4
Apply group-average clustering to the points in Figures 17.5 and 17.6. Map them onto the surface of the unit sphere in a three-dimensional space to get normalized vectors. Is the group-average clustering different from the single-link and complete-link clusterings?

Exercise 17.5
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 17.6
Suppose a run of HAC finds the clustering with $K = 7$ to have the highest value on some pre-chosen goodness measure of clustering. Have we found the highest-value clustering amongst all clusterings with $K = 7$?

Exercise 17.7
Show that for a single-link clustering of points on a line (Figure 17.12) only a linear number of similarities has to be computed. What is the overall complexity of single-link clustering of points on a line?

Exercise 17.8
Prove that single-link, complete-link, and group-average are monotonic in the sense defined on page 276.

Exercise 17.9
For $N$ points, there are $K^N/K!$ different flat clusterings into $K$ clusters. How many different hierarchical clusterings (or dendrograms) are there for $N$ points? Which number is greater?

Exercise 17.10
For a 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?