16 *Flat clustering*

CLUSTER

Clustering algorithms group a set of documents into subsets or *clusters*. The algorithms' goal is to create clusters that are coherent internally, but clearly different from each other. In other words, documents within a cluster should be as similar as possible; and documents in one cluster should be as dissimilar as possible from documents in other clusters.

UNSUPERVISED LEARNING

Clustering is the most common form of *unsupervised learning*. No supervision means that there is no human expert who has assigned documents to classes. In clustering, it is the distribution and makeup of the data that will determine cluster membership. A simple example is Figure 16.1. It is visually clear that there are three distinct clusters of points. This chapter and Chapter 17 introduce algorithms that find such clusters in an unsupervised fashion.

The difference between clustering and classification may not seem great at first. After all, in both cases we have a partition of a set of documents into groups. But as we will see the two problems are fundamentally different. Classification is a form of supervised learning (Chapter 13, page 237): Our goal is to replicate a categorical distinction that a human supervisor imposes on the data. In unsupervised learning, of which clustering is the most important example, we have no such teacher to guide us.

The key input to a clustering algorithm is the distance measure. In Figure 16.1, the distance measure is distance in the two-dimensional (2D) plane. This measure suggests three different clusters in the figure. In document clustering, the distance measure is often Euclidean distance. Different distance measures give rise to different clusterings. Thus, the distance measure is an important means by which we can influence the outcome of clustering.

FLAT CLUSTERING

Flat clustering creates a flat set of clusters without any explicit structure that would relate clusters to each other. *Hierarchical clustering* creates a hierarchy of clusters and will be covered in Chapter 17. Chapter 17 also addresses the difficult problem of labeling clusters automatically.

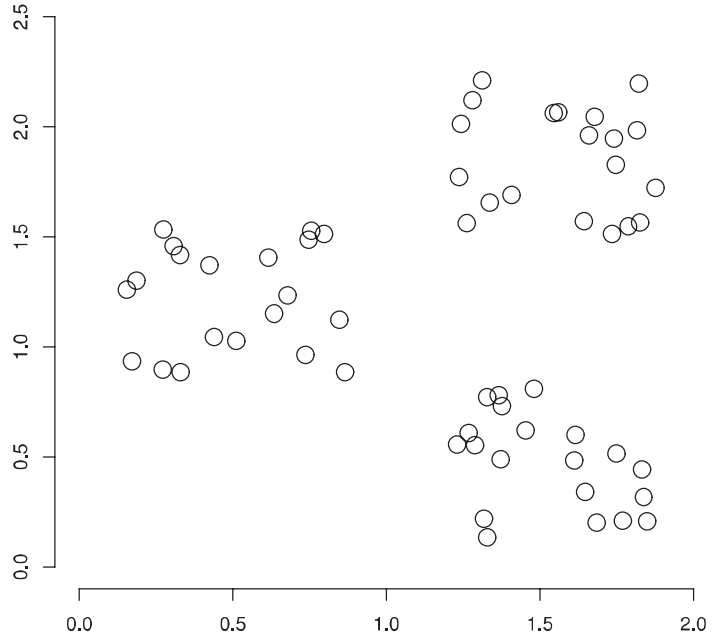


Figure 16.1 An example of a data set with a clear cluster structure.

HARD CLUSTERING

SOFT CLUSTERING

A second important distinction can be made between hard and soft clustering algorithms. *Hard clustering* computes a *hard assignment* – each document is a member of exactly one cluster. The assignment of *soft clustering algorithms* is *soft* – a document's assignment is a distribution over all clusters. In a soft assignment, a document has fractional membership in several clusters. Latent semantic indexing, a form of dimensionality reduction, is a soft clustering algorithm (Chapter 18, page 382).

This chapter motivates the use of clustering in information retrieval by introducing a number of applications (Section 16.1), defines the problem we are trying to solve in clustering (Section 16.2), and discusses measures for evaluating cluster quality (Section 16.3). It then describes two flat clustering algorithms, K-means (Section 16.4), a hard clustering algorithm, and the expectation maximization (or EM) algorithm (Section 16.5), a soft clustering algorithm. K-means is perhaps the most widely used flat clustering algorithm because of its simplicity and efficiency. The EM algorithm is a generalization of K-means and can be applied to a large variety of document representations and distributions.

16.1 Clustering in information retrieval

CLUSTER HYPOTHESIS

The *cluster hypothesis* states the fundamental assumption we make when using clustering in information retrieval (IR).

Cluster hypothesis. Documents in the same cluster behave similarly with respect to relevance to information needs.

Table 16.1 Some applications of clustering in IR.

application	What is clustered?	benefit	example
search result clustering	search results	more effective information presentation to user	Figure 16.2
scatter-gather	(subsets of) collection	alternative user interface: "search without typing"	Figure 16.3
collection clustering	collection	effective information presentation for exploratory browsing	McKeown et al. (2002), http://news.google.com
language modeling	collection	increased precision and/or recall	Liu and Croft (2004)
cluster-based retrieval	collection	higher efficiency: faster search	Salton (1971a)

The hypothesis states that if there is a document from a cluster that is relevant to a search request, then it is likely that other documents from the same cluster are also relevant. This is because clustering puts together documents that share many terms. The cluster hypothesis essentially is the contiguity hypothesis in Chapter 14 (page 266). In both cases, we posit that similar documents behave similarly with respect to relevance.

Table 16.1 shows some of the main applications of clustering in IR. They differ in the set of documents that they cluster – search results, collection, or subsets of the collection – and the aspect of an IR system they try to improve – user experience, user interface, effectiveness or efficiency of the search system. But they are all based on the basic assumption stated by the cluster hypothesis.

SEARCH RESULT CLUSTERING

The first application mentioned in Table 16.1 is *search result clustering* where by search results we mean the documents that were returned in response to a query. The default presentation of search results in IR is a simple list. Users scan the list from top to bottom until they have found the information they are looking for. Instead, search result clustering clusters the search results, so that similar documents appear together. It is often easier to scan a few coherent groups than many individual documents. This is particularly useful if a search term has different word senses. The example in Figure 16.2 is aguar . Three frequent senses on the web refer to the car, the animal, and an Apple operating system. The *Clustered Results* panel returned by the Vivísimo search engine (http://vivisimo.com) can be a more effective user interface for understanding what is in the search results than a simple list of documents.

SCATTER-GATHER

A better user interface is also the goal of *scatter-gather*, the second application in Table 16.1. Scatter-gather clusters the whole collection to get groups of documents that the user can select or *gather*. The selected groups are merged and the resulting set is again clustered. This process is repeated until a cluster of interest is found. An example is shown in Figure 16.3.

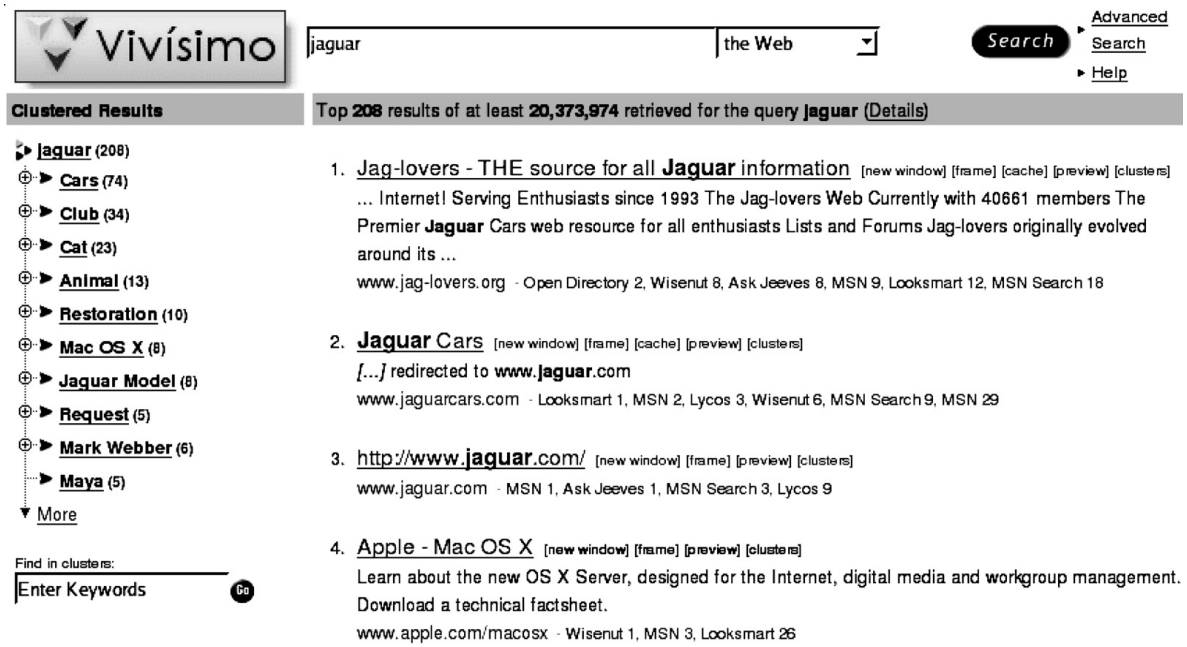


Figure 16.2 Clustering of search results to improve recall. None of the top hits cover the animal sense of aguar , but users can easily access it by clicking on the *Cat Cluster* in the *Clustered Results* panel on the left (third arrow from the top).

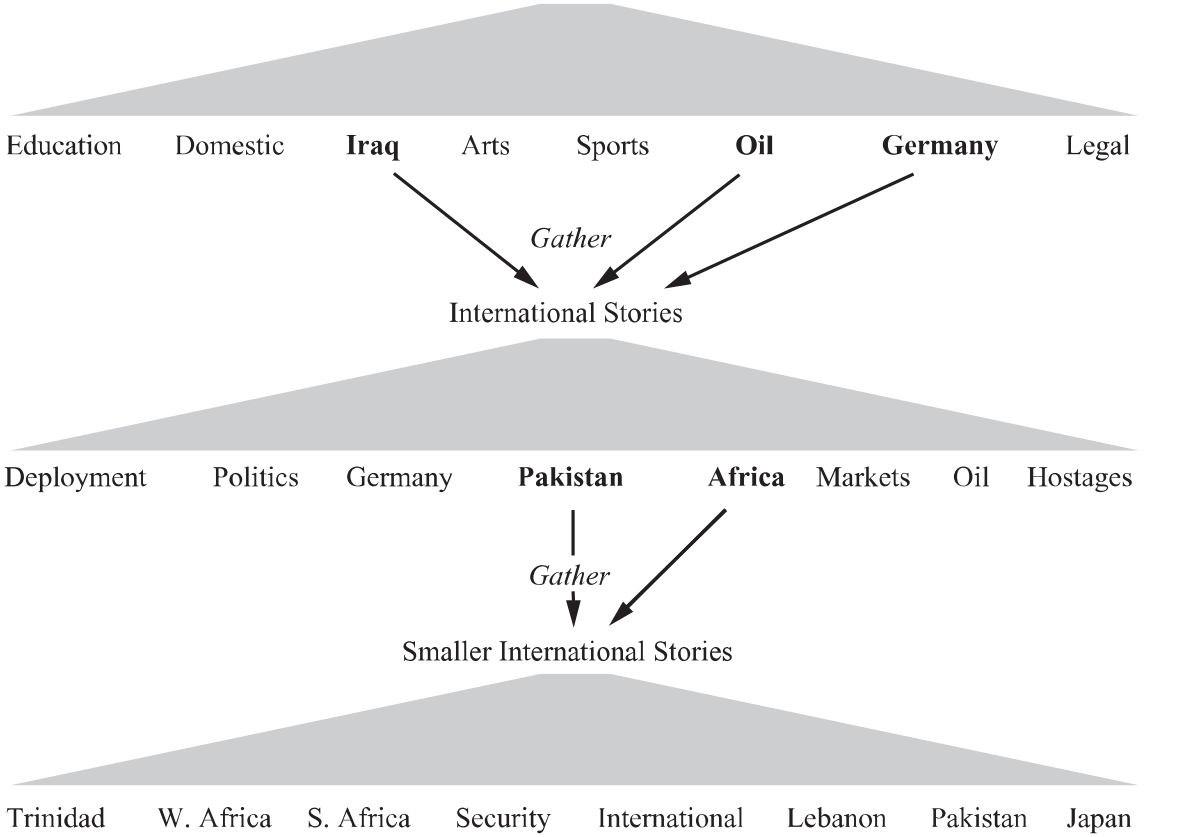


Figure 16.3 An example of a user session in scatter-gather. A collection of *New York Times* news stories is clustered ("scattered") into eight clusters (*top row*). The user manually *gathers* three of these into a smaller collection *International Stories* and performs another scattering operation. This process repeats until a small cluster with *relevant* documents is found (e.g., *Trinidad*).

Automatically generated clusters like those in Figure 16.3 are not as neatly organized as a manually constructed hierarchical tree like the Open Directory at http://dmoz.org. Also, finding descriptive labels for clusters automatically is a difficult problem (Section 17.7, page 363). But cluster-based navigation is an interesting alternative to keyword searching, the standard IR paradigm. This is especially true in scenarios where users prefer browsing over searching because they are unsure about which search terms to use.

As an alternative to the user-mediated iterative clustering in scatter-gather, we can also compute a static hierarchical clustering of a collection that is not influenced by user interactions ("Collection clustering" in Table 16.1). Google News and its precursor, the Columbia NewsBlaster system, are examples of this approach. In the case of news, we need to frequently recompute the clustering to make sure that users can access the latest breaking stories. Clustering is well suited for access to a collection of news stories because news reading is not really search, but rather a process of selecting a subset of stories about recent events.

The fourth application of clustering exploits the cluster hypothesis directly for improving search results, based on a clustering of the entire collection. We use a standard inverted index to identify an initial set of documents that match the query, but we then add other documents from the same clusters even if they have low similarity to the query. For example, if the query is car and several car documents are taken from a cluster of automobile documents, then we can add documents from this cluster that use terms other than car (automobile, vehicle, etc.). This can increase recall; a group of documents with high mutual similarity is often relevant as a whole.

More recently, this idea has been used for language modeling. Equation (12.10), page 226, showed that to avoid sparse data problems in the language modeling approach to IR, the model of document d can be interpolated with a collection model. But the collection contains many documents with terms untypical of d. By replacing the collection model with a model derived from d's cluster, we get more accurate estimates of the occurrence probabilities of terms in d.

Clustering can also speed up search. As we saw in Section 6.3.2 (page 113), search in the vector space model amounts to finding the nearest neighbors to the query. The inverted index supports fast nearest-neighbor search for the standard IR setting. However, sometimes we may not be able to use an inverted index efficiently, for example, in latent semantic indexing (Chapter 18). In such cases, we could compute the similarity of the query to every document, but this is slow. The cluster hypothesis offers an alternative: Find the clusters that are closest to the query and only consider documents from these clusters. Within this much smaller set, we can compute similarities exhaustively and rank documents in the usual way. Because there are many fewer clusters than documents, finding the closest cluster is fast, and because the documents matching a query are all similar to each other, they

tend to be in the same clusters. Although this algorithm is inexact, the expected decrease in search quality is small. This is essentially the application of clustering that was covered in Section 7.1.6 (page 130).

? **Exercise 16.1** Define two documents as similar if they have at least two proper names like Clinton or Sarkozy in common. Give an example of an information need and two documents, for which the cluster hypothesis does *not* hold for this notion of similarity.

Exercise 16.2 Make up a simple, one-dimensional example (i.e., points on a line) with two clusters where the inexactness of cluster-based retrieval shows up. In your example, retrieving clusters close to the query should do worse than direct nearest neighbor search.

16.2 Problem statement

We can define the goal in hard flat clustering as follows. Given (i) a set of documents $D = \{d_1, \ldots, d_N\}$, (ii) a desired number of clusters K, and (iii) an *objective function* that evaluates the quality of a clustering, we want to compute an assignment $\gamma : D \rightarrow \{1, \ldots, K\}$ that minimizes (or, in other cases, maximizes) the objective function. In most cases, we also demand that γ is surjective, that is, that none of the K clusters is empty.

OBJECTIVE FUNCTION

The objective function is often defined in terms of similarity or distance between documents. Below, we will see that the objective in K-means clustering is to minimize the average distance between documents and their centroids or, equivalently, to maximize the similarity between documents and their centroids. The discussion of similarity measures and distance metrics in Chapter 14 (page 267) also applies to this chapter. As in Chapter 14, we use both similarity and distance to talk about relatedness between documents.

For documents, the type of similarity we want is usually topic similarity or high values on the same dimensions in the vector space model. For example, documents about China have high values on dimensions like Chinese, Beiing , and Mao whereas documents about the UK tend to have high values for London, Britain, and Queen. We approximate topic similarity with cosine similarity or Euclidean distance in vector space (Chapter 6). If we intend to capture similarity of a type other than topic, for example, similarity of language, then a different representation may be appropriate. When computing topic similarity, stop words can be safely ignored, but they are important cues for separating clusters of English (in which the occurs frequently and la infrequently) and French documents (in which the occurs infrequently and la frequently).

PARTITIONAL CLUSTERING

A note on terminology. An alternative definition of hard clustering is that a document can be a full member of more than one cluster. *Partitional*

clustering always refers to a clustering where each document belongs to exactly one cluster. (But in a partitional hierarchical clustering (Chapter 17), all members of a cluster are of course also members of its parent.) On the definition of hard clustering that permits multiple membership, the difference between soft clustering and hard clustering is that membership values in hard clustering are either 0 or 1, whereas they can take on any non-negative value in soft clustering.

EXHAUSTIVE

Some researchers distinguish between *exhaustive* clusterings that assign each document to a cluster and nonexhaustive clusterings, in which some documents will be assigned to no cluster. Nonexhaustive clusterings, in which each document is a member of either no cluster or one cluster, are called *exclusive*. We define clustering to be exhaustive in this book.

16.2.1 Cardinality – The number of clusters

CARDINALITY

A difficult issue in clustering is determining the number of clusters or *cardinality* of a clustering, which we denote by K. Often K is nothing more than a good guess based on experience or domain knowledge. But for K-means, we will also introduce a heuristic method for choosing K and an attempt to incorporate the selection of K into the objective function. Sometimes the application puts constraints on the range of K. For example, the scatter-gather interface in Figure 16.3 could not display more than about $K = 10$ clusters per layer because of the size and resolution of computer monitors in the early 1990s.

Because our goal is to optimize an objective function, clustering is essentially a search problem. The brute force solution would be to enumerate all possible clusterings and pick the best. However, there are exponentially many partitions, so this approach is not feasible.[1] For this reason, most flat clustering algorithms refine an initial partitioning iteratively. If the search starts at an unfavorable initial point, we may miss the global optimum. Finding a good starting point is therefore another important problem we have to solve in flat clustering.

16.3 Evaluation of clustering

Typical objective functions in clustering formalize the goal of attaining high intracluster similarity (documents within a cluster are similar) and low intercluster similarity (documents from different clusters are dissimilar). This is an *internal criterion* for the quality of a clustering. But good scores on an internal criterion do not necessarily translate into good effectiveness in an

INTERNAL CRITERION OF QUALITY

[1] An upper bound on the number of clusterings is $K^N/K!$. The exact number of different partitions of N documents into K clusters is the Stirling number of the second kind. See http://mathworld.wolfram.com/StirlingNumberoftheSecondind.html or Comtet (1974).

application. An alternative to internal criteria is direct evaluation in the application of interest. For search result clustering, we may want to measure the time it takes users to find an answer with different clustering algorithms. This is the most direct evaluation, but it is expensive, especially if large user studies are necessary.

As a surrogate for user judgments, we can use a set of classes in an evaluation benchmark or gold standard (see Section 8.5, page 151, and Section 13.6, page 258). The gold standard is ideally produced by human judges with a good level of inter-judge agreement (see Chapter 8, page 140). We can then compute an *external criterion* that evaluates how well the clustering matches the gold standard classes. For example, we may want to say that the optimal clustering of the search results for aguar in Figure 16.2 consists of three classes corresponding to the three senses *car*, *animal*, and *operating system*. In this type of evaluation, we only use the partition provided by the gold standard, not the class labels.

EXTERNAL CRITERION OF QUALITY

This section introduces four external criteria of clustering quality. *Purity* is a simple and transparent evaluation measure. *Normalized mutual information* can be information-theoretically interpreted. The *Rand index* penalizes both false-positive and false-negative decisions during clustering. The *F measure* in addition supports differential weighting of these two types of errors.

PURITY

To compute *purity*, each cluster is assigned to the class which is most frequent in the cluster, and then the accuracy of this assignment is measured by counting the number of correctly assigned documents and dividing by N. Formally:

$$\text{purity}(\Omega, \mathbb{C}) = \frac{1}{N} \sum_k \max_j |\omega_k \cap c_j| \tag{16.1}$$

where $\Omega = \{\omega_1, \omega_2, \ldots, \omega_K\}$ is the set of clusters and $\mathbb{C} = \{c_1, c_2, \ldots, c_J\}$ is the set of classes. We interpret ω_k as the set of documents in ω_k and c_j as the set of documents in c_j in Equation (16.1).

We present an example of how to compute purity in Figure 16.4.[2] Bad clusterings have purity values close to 0, a perfect clustering has a purity of 1. Purity is compared with the other three measures discussed in this chapter in Table 16.2.

High purity is easy to achieve when the number of clusters is large – in particular, purity is 1 if each document gets its own cluster. Thus, we cannot use purity to trade off the quality of the clustering against the number of clusters.

[2] Recall our note of caution from Figure 14.2 (page 268) when looking at this and other 2D figures in this and the following chapter: these illustrations can be misleading because 2D projections of length-normalized vectors distort similarities and distances between points.

Table 16.2 The four external evaluation measures applied to the clustering in Figure 16.4.

	purity	NMI	RI	F_5
lower bound	0.0	0.0	0.0	0.0
maximum	1.0	1.0	1.0	1.0
value for Figure 16.4	0.71	0.36	0.68	0.46

NORMALIZED MUTUAL INFORMATION

A measure that allows us to make this tradeoff is *normalized mutual information* or *NMI*:

$$\text{NMI}(\Omega, \mathbb{C}) = \frac{I(\Omega;\mathbb{C})}{[H(\Omega) + H(\mathbb{C})]/2}. \tag{16.2}$$

I is mutual information (cf. Chapter 13, page 252):

$$I(\Omega;\mathbb{C}) = \sum_k \sum_j P(\omega_k \cap c_j) \log \frac{P(\omega_k \cap c_j)}{P(\omega_k)P(c_j)} \tag{16.3}$$

$$= \sum_k \sum_j \frac{|\omega_k \cap c_j|}{N} \log \frac{N|\omega_k \cap c_j|}{|\omega_k||c_j|} \tag{16.4}$$

where $P(\omega_k)$, $P(c_j)$, and $P(\omega_k \cap c_j)$ are the probabilities of a document being in cluster ω_k, class c_j, and in the intersection of ω_k and c_j, respectively. Equation (16.4) is equivalent to Equation (16.3) for maximum likelihood estimates (MLE) of the probabilities (i.e., the estimate of each probability is the corresponding relative frequency).

H is entropy as defined in Chapter 5 (page 91):

$$H(\Omega) = -\sum_k P(\omega_k) \log P(\omega_k) \tag{16.5}$$

$$= -\sum_k \frac{|\omega_k|}{N} \log \frac{|\omega_k|}{N} \tag{16.6}$$

where, again, the second equation is based on MLEs of the probabilities.

$I(\Omega;\mathbb{C})$ in Equation (16.3) measures the amount of information by which our knowledge about the classes increases when we are told what the clusters

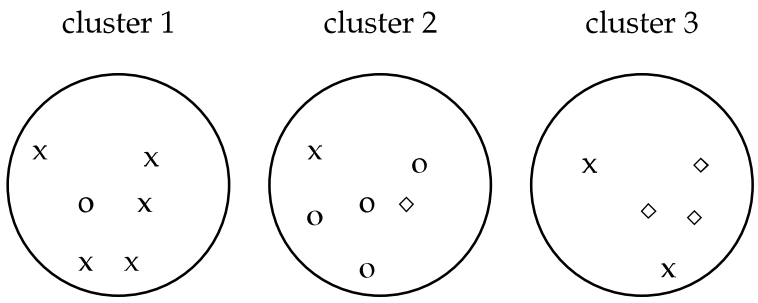


Figure 16.4 Purity as an external evaluation criterion for cluster quality. Majority class and number of members of the majority class for the three clusters are: x, 5 (cluster 1); o, 4 (cluster 2); and ⋄, 3 (cluster 3). Purity is $(1/17) \times (5 + 4 + 3) \approx 0.71$.

are. The minimum of $I(\Omega;\mathbb{C})$ is 0 if the clustering is random with respect to class membership. In that case, knowing that a document is in a particular cluster does not give us any new information about what its class might be. Maximum mutual information is reached for a clustering Ω_{exact} that perfectly recreates the classes – but also if clusters in Ω_{exact} are further subdivided into smaller clusters (Exercise 16.7). In particular, a clustering with $K = N$ one-document clusters has maximum MI. So MI has the same problem as purity: It does not penalize large cardinalities and thus does not formalize our bias that, other things being equal, fewer clusters are better.

The normalization by the denominator $[H(\Omega) + H(\mathbb{C})]/2$ in Equation (16.2) fixes this problem; entropy tends to increase with the number of clusters. For example, $H(\Omega)$ reaches its maximum $\log N$ for $K = N$, which ensures that NMI is low for $K = N$. Because NMI is normalized, we can use it to compare clusterings with different numbers of clusters. The particular form of the denominator is chosen because $[H(\Omega) + H(\mathbb{C})]/2$ is a tight upper bound on $I(\Omega;\mathbb{C})$ (Exercise 16.8). Thus, NMI is always a number between 0 and 1.

An alternative to this information-theoretic interpretation of clustering is to view it as a series of decisions, one for each of the $N(N-1)/2$ pairs of documents in the collection. We want to assign two documents to the same cluster if and only if they are similar. A true-positive (TP) decision assigns two similar documents to the same cluster, a true-negative (TN) decision assigns two dissimilar documents to different clusters. There are two types of errors we can commit. A false-positive (FP) decision assigns two dissimilar documents to the same cluster. A false-negative (FN) decision assigns two similar documents to different clusters. The *Rand index (RI)* measures the RI percentage of decisions that are correct. That is, it is simply accuracy (Section 8.3, page 143).

RAND INDEX (RI)

$$\text{RI} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

As an example, we compute RI for Figure 16.4. We first compute TP + FP. The three clusters contain six, six, and five points, respectively, so the total number of "positives" or pairs of documents that are in the same cluster is:

$$\text{TP} + \text{FP} = \binom{6}{2} + \binom{6}{2} + \binom{5}{2} = 40.$$

Of these, the x pairs in cluster 1, the o pairs in cluster 2, the $\diamond$ pairs in cluster 3, and the x pair in cluster 3 are true positives:

$$\text{TP} = \binom{5}{2} + \binom{4}{2} + \binom{3}{2} + \binom{2}{2} = 20.$$

Thus, $\text{FP} = 40 - 20 = 20$.

FN and TN are computed similarly, resulting in the following contingency table:

	same cluster	different clusters
same class	TP = 20	FN = 24
different classes	FP = 20	TN = 72

RI is then $(20 + 72)/(20 + 20 + 24 + 72) \approx 0.68$.

The RI gives equal weight to FPs and FNs. Separating similar documents is sometimes worse than putting pairs of dissimilar documents in the same cluster. We can use the *F measure* (Section 8.3, page 142) to penalize FNs more strongly than FPs by selecting a value $\beta > 1$, thus giving more weight to recall.

F MEASURE

$$P = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad R = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}.$$

Based on the numbers in the contingency table, $P = 20/40 = 0.5$ and $R = 20/44 \approx 0.455$. This gives us $F_1 \approx 0.48$ for $\beta = 1$ and $F_5 \approx 0.456$ for $\beta = 5$. In IR, evaluating clustering with F has the advantage that the measure is already familiar to the research community.

? **Exercise 16.3** Replace every point d in Figure 16.4 with two identical copies of d in the same class. (i) Is it less difficult, equally difficult or more difficult to cluster this set of thirty-four points as opposed to the seventeen points in Figure 16.4? (ii) Compute purity, NMI, RI, and F_5 for the clustering with thirty-four points. Which measures increase and which stay the same after doubling the number of points? (iii) Given your assessment in (i) and the results in (ii), which measures are best suited to compare the quality of the two clusterings?

16.4 *K*-means

K-means is the most important flat clustering algorithm. Its objective is to minimize the average squared Euclidean distance (Chapter 6, page 121) of documents from their cluster centers where a cluster center is defined as the mean or *centroid* $\vec{\mu}$ of the documents in a cluster ω:

CENTROID

$$\vec{\mu}(\omega) = \frac{1}{|\omega|} \sum_{\vec{x} \in \omega} \vec{x}.$$

The definition assumes that documents are represented as length-normalized vectors in a real-valued space in the familiar way. We used centroids for Rocchio classification in Chapter 14 (page 269). They play a similar role here. The ideal cluster in K-means is a sphere with the centroid as its center of gravity. Ideally, the clusters should not overlap. Our desiderata for classes in Rocchio classification were the same. The difference is that we have no labeled training set in clustering for which we know which documents should be in the same cluster.

K-MEANS($\{\vec{x}_1, \ldots, \vec{x}_N\}$, K)
1 $(\vec{s}_1, \vec{s}_2, \ldots, \vec{s}_K) \leftarrow$ SELECTRANDOMSEEDS($\{\vec{x}_1, \ldots, \vec{x}_N\}$, K)
2 **for** $k \leftarrow 1$ **to** K
3 **do** $\vec{\mu}_k \leftarrow \vec{s}_k$
4 **while** stopping criterion has not been met
5 **do for** $k \leftarrow 1$ **to** K
6 **do** $\omega_k \leftarrow \{\}$
7 **for** $n \leftarrow 1$ **to** N
8 **do** $j \leftarrow \arg\min_{j'} |\vec{\mu}_{j'} - \vec{x}_n|$
9 $\omega_j \leftarrow \omega_j \cup \{\vec{x}_n\}$ *(reassignment of vectors)*
10 **for** $k \leftarrow 1$ **to** K
11 **do** $\vec{\mu}_k \leftarrow \frac{1}{|\omega_k|} \sum_{\vec{x} \in \omega_k} \vec{x}$ *(recomputation of centroids)*
12 **return** $\{\vec{\mu}_1, \ldots, \vec{\mu}_K\}$

Figure 16.5 The K-means algorithm. For most IR applications, the vectors $\vec{x}_n \in \mathbb{R}^M$ should be length normalized. Alternative methods of seed selection and initialization are discussed on page 334.

A measure of how well the centroids represent the members of their clusters is the *residual sum of squares* or *RSS*, the squared distance of each vector from its centroid summed over all vectors:

RESIDUAL SUM OF SQUARES

$$\text{RSS}_k = \sum_{\vec{x} \in \omega_k} |\vec{x} - \vec{\mu}(\omega_k)|^2$$

$$\text{RSS} = \sum_{k=1}^{K} \text{RSS}_k \tag{16.7}$$

RSS is the objective function in K-means and our goal is to minimize it. Because N is fixed, minimizing RSS is equivalent to minimizing the average squared distance, a measure of how well centroids represent their documents.

The first step of K-means is to select as initial cluster centers K randomly selected documents, the *seeds*. The algorithm then moves the cluster centers around in space to minimize RSS. As shown in Figure 16.5, this is done iteratively by repeating two steps until a stopping criterion is met: Reassigning documents to the cluster with the closest centroid and recomputing each centroid based on the current members of its cluster. Figure 16.6 shows snapshots from nine iterations of the K-means algorithm for a set of points. The "centroid" column of Table 17.2 (page 364) shows examples of centroids.

SEED

We can apply one of the following termination conditions.

- A fixed number of iterations I has been completed. This condition limits the runtime of the clustering algorithm, but in some cases the quality of the clustering will be poor because of an insufficient number of iterations.

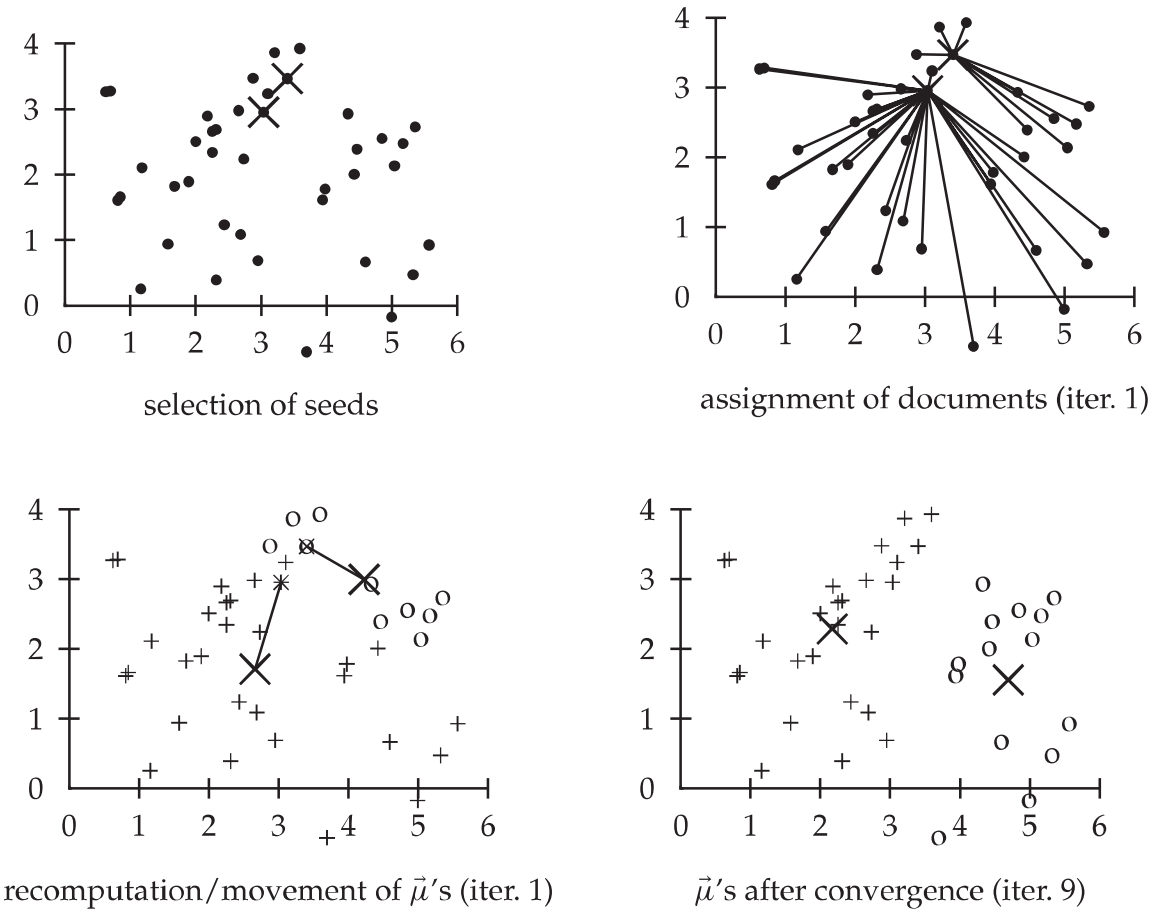


Figure 16.6 A K-means example for $K = 2$ in $\mathbb{R}^2$. The position of the two centroids ($\vec{\mu}$'s shown as X's in the top four panels) converges after nine iterations.

- Assignment of documents to clusters (the partitioning function γ) does not change between iterations. Except for cases with a bad local minimum, this produces a good clustering, but runtime may be unacceptably long.
- Centroids $\vec{\mu}_k$ do not change between iterations. This is equivalent to γ not changing (Exercise 16.5).

- Terminate when RSS falls below a threshold. This criterion ensures that the clustering is of a desired quality after termination. In practice, we need to combine it with a bound on the number of iterations to guarantee termination.
- Terminate when the decrease in RSS falls below a threshold θ. For small θ, this indicates that we are close to convergence. Again, we need to combine it with a bound on the number of iterations to prevent very long runtimes.

We now show that K-means converges by proving that RSS monotonically decreases in each iteration. We will use *decrease* in the meaning *decrease or does not change* in this section. First, RSS decreases in the reassignment step; each vector is assigned to the closest centroid, so the distance it contributes to RSS decreases. Second, it decreases in the recomputation step because the new centroid is the vector $\vec{v}$ for which RSS_k reaches its minimum.

$$\text{RSS}_k(\vec{v}) = \sum_{\vec{x} \in \omega_k} |\vec{v} - \vec{x}|^2 = \sum_{\vec{x} \in \omega_k} \sum_{m=1}^{M} (v_m - x_m)^2 \tag{16.8}$$

$$\frac{\partial \text{RSS}_k(\vec{v})}{\partial v_m} = \sum_{\vec{x} \in \omega_k} 2(v_m - x_m) \tag{16.9}$$

where x_m and v_m are the m^{th} components of their respective vectors. Setting the partial derivative to zero, we get:

$$v_m = \frac{1}{|\omega_k|} \sum_{\vec{x} \in \omega_k} x_m \tag{16.10}$$

which is the componentwise definition of the centroid. Thus, we minimize RSS_k when the old centroid is replaced with the new centroid. RSS, the sum of the RSS_k, must then also decrease during recomputation.

Because there is only a finite set of possible clusterings, a monotonically decreasing algorithm will eventually arrive at a (local) minimum. Take care, however, to break ties consistently, for example, by assigning a document to the cluster with the lowest index if there are several equidistant centroids. Otherwise, the algorithm can cycle forever in a loop of clusterings that have the same cost.

Although this proves the convergence of K-means, there is unfortunately no guarantee that a *global minimum* in the objective function will be reached.

OUTLIER This is a particular problem if a document set contains many *outliers*, documents that are far from any other documents and therefore do not fit well into any cluster. Frequently, if an outlier is chosen as an initial seed, then no other vector is assigned to it during subsequent iterations. Thus, we end up

SINGLETON CLUSTER with a *singleton cluster* (a cluster with only one document) even though there is probably a clustering with lower RSS. Figure 16.7 shows an example of a suboptimal clustering resulting from a bad choice of initial seeds.

Another type of suboptimal clustering that frequently occurs is one with empty clusters (Exercise 16.11).

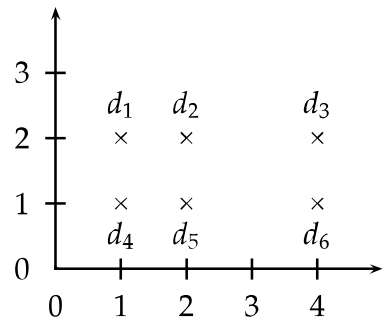


Figure 16.7 The outcome of clustering in K-means depends on the initial seeds. For seeds d_2 and d_5, K-means converges to $\{\{d_1, d_2, d_3\}, \{d_4, d_5, d_6\}\}$, a suboptimal clustering. For seeds d_2 and d_3, it converges to $\{\{d_1, d_2, d_4, d_5\}, \{d_3, d_6\}\}$, the global optimum for $K = 2$.

Effective heuristics for seed selection include (i) excluding outliers from the seed set; (ii) trying out multiple starting points and choosing the clustering with lowest cost; and (iii) obtaining seeds from another method such as hierarchical clustering. Because deterministic hierarchical clustering methods are more predictable than K-means, a hierarchical clustering of a small random sample of size iK (e.g., for $i = 5$ or $i = 10$) often provides good seeds (see the description of the Buckshot algorithm, Chapter 17, page 366).

Other initialization methods compute seeds that are not selected from the vectors to be clustered. A robust method that works well for a large variety of document distributions is to select i (e.g., $i = 10$) random vectors for each cluster and use their centroid as the seed for this cluster. See Section 16.6 for more sophisticated initializations.

What is the time complexity of K-means? Most of the time is spent on computing vector distances. One such operation costs $\Theta(M)$. The reassignment step computes KN distances, so its overall complexity is $\Theta(KNM)$. In the recomputation step, each vector gets added to a centroid once, so the complexity of this step is $\Theta(NM)$. For a fixed number of iterations I, the overall complexity is therefore $\Theta(IKNM)$. Thus, K-means is linear in all relevant factors: iterations, number of clusters, number of vectors, and dimensionality of the space. This means that K-means is more efficient than the hierarchical algorithms in Chapter 17. We had to fix the number of iterations I, which can be tricky in practice. But in most cases, K-means quickly reaches either complete convergence or a clustering that is close to convergence. In the latter case, a few documents would switch membership if further iterations were computed, but this has a small effect on the overall quality of the clustering.

There is one subtlety in the preceding argument. Even a linear algorithm can be quite slow if one of the arguments of $\Theta(\ldots)$ is large, and M usually is large. High dimensionality is not a problem for computing the distance of two documents. Their vectors are sparse, so that only a small fraction of the theoretically possible M componentwise differences need to be computed. Centroids, however, are dense; they pool all terms that occur in any of

the documents of their clusters. As a result, distance computations are time consuming in a naive implementation of K-means. But there are simple and effective heuristics for making centroid–document similarities as fast to compute as document–document similarities. Truncating centroids to the most significant k terms (e.g., $k = 1{,}000$) hardly decreases cluster quality while achieving a significant speedup of the reassignment step (see references in Section 16.6).

K-MEDOIDS

The same efficiency problem is addressed by *K-medoids*, a variant of K-means that computes medoids instead of centroids as cluster centers. We

MEDOID

define the *medoid* of a cluster as the document vector that is closest to the centroid. Since medoids are sparse document vectors, distance computations are fast.

16.4.1 Cluster cardinality in K-means

We stated in Section 16.2 that the number of clusters K is an input to most flat clustering algorithms. What do we do if we cannot come up with a plausible guess for K?

A naive approach would be to select the optimal value of K according to the objective function, namely, the value of K that minimizes RSS. Defining $\text{RSS}_{\text{min}}(K)$ as the minimal RSS of all clusterings with K clusters, we observe that $\text{RSS}_{\text{min}}(K)$ is a monotonically decreasing function in K (Exercise 16.13), which reaches its minimum 0 for $K = N$ where N is the number of documents. We would end up with each document being in its own cluster. Clearly, this is not an optimal clustering.

A heuristic method that gets around this problem is to estimate $\text{RSS}_{\text{min}}(K)$ as follows. We first perform i (e.g., $i = 10$) clusterings with K clusters (each with a different initialization) and compute the RSS of each. Then we take the minimum of the i RSS values. We denote this minimum by $\widehat{\text{RSS}}_{\text{min}}(K)$. Now we can inspect the values $\widehat{\text{RSS}}_{\text{min}}(K)$ as K increases and find the "knee" in the curve – the point where successive decreases in $\widehat{\text{RSS}}_{\text{min}}$ become noticeably smaller. There are two such points in Figure 16.8, one at $K = 4$, where the gradient flattens slightly, and a clearer flattening at $K = 9$. This is typical: There is seldom a single best number of clusters. We still need to employ an external constraint to choose from a number of possible values of K (four and nine in this case).

A second type of criterion for cluster cardinality imposes a penalty for each new cluster – where conceptually we start with a single cluster containing all documents and then search for the optimal number of clusters K by successively increasing K. To determine the cluster cardinality in this way, we create a generalized objective function that combines two elements:

DISTORTION

distortion, a measure of how much documents deviate from the prototype

MODEL

of their clusters (e.g., RSS for K-means); and a measure of *model complexity*.

COMPLEXITY

We interpret a clustering here as a model of the data. Model complexity in clustering is usually the number of clusters or a function thereof. For

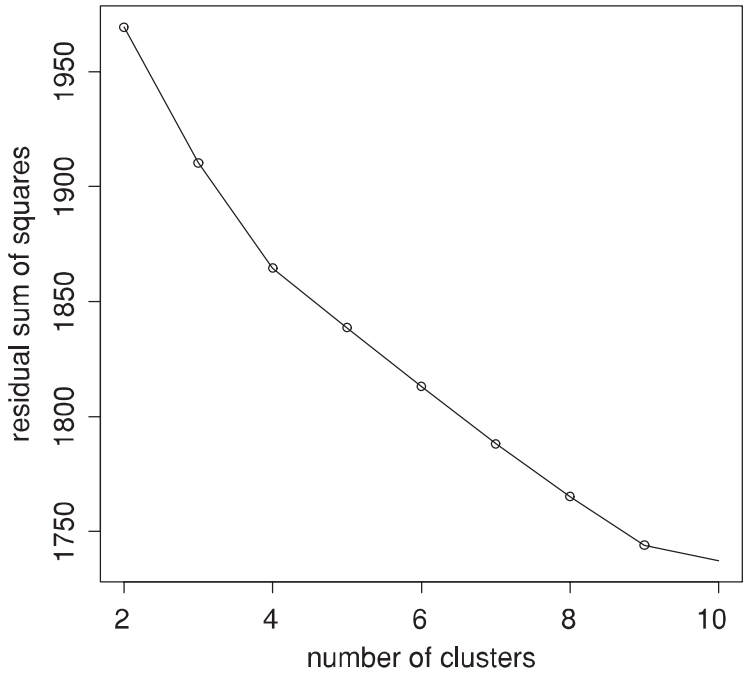


Figure 16.8 Estimated minimal residual sum of squares ($\widehat{\text{RSS}}_{\text{min}}$) as a function of the number of clusters in K-means. In this clustering of 1203 Reuters-RCV1 documents, there are two points where the $\widehat{\text{RSS}}_{\text{min}}$ curve flattens: at four clusters and at nine clusters. The documents were selected from the categories *China*, *Germany*, *Russia*, and *Sports*, so the $K = 4$ clustering is closest to the RCV1-Reuters classification.

K-means, we then get this selection criterion for K:

$$K = \arg\min_K [\text{RSS}_{\text{min}}(K) + \lambda K] \tag{16.11}$$

where λ is a weighting factor. A large value of λ favors solutions with few clusters. For $\lambda = 0$, there is no penalty for more clusters and $K = N$ is the best solution.

The obvious difficulty with Equation (16.11) is that we need to determine λ. Unless this is easier than determining K directly, then we are back to square one. In some cases, we can choose values of λ that have worked well for similar data sets in the past. For example, if we periodically cluster news stories from a newswire, there is likely to be a fixed value of λ that gives us the right K in each successive clustering. In this application, we would not be able to determine K based on past experience because K changes.

AKAIKE INFORMATION CRITERION

A theoretical justification for Equation (16.11) is the *Akaike information criterion* or AIC, an information-theoretic measure that trades off distortion against model complexity. The general form of AIC is:

$$\textbf{AIC:} \quad K = \arg\min_K [-2L(K) + 2q(K)] \tag{16.12}$$

where $-L(K)$, the negative maximum log-likelihood of the data for K clusters, is a measure of distortion and $q(K)$, the number of parameters of a model with K clusters, is a measure of model complexity. We will not attempt to derive the AIC here, but it is easy to understand intuitively. The first property of a good model of the data is that each data point is modeled well by

the model. This is the goal of low distortion. But models should also be small (i.e., have low model complexity); a model that merely describes the data (and therefore has zero distortion) is worthless. AIC provides a theoretical justification for one particular way of weighting these two factors, distortion and model complexity, when selecting a model.

For K-means, the AIC can be stated as follows:

$$\textbf{AIC:} \quad K = \arg\min_K [\mathrm{RSS}_{\min}(K) + 2MK] \tag{16.13}$$

Equation (16.13) is a special case of Equation (16.11) for $\lambda = 2M$.

To derive Equation (16.13) from Equation (16.12), observe that $q(K) = KM$ in K-means because each element of the K centroids is a parameter that can be varied independently; and that $L(K) = -(1/2)\mathrm{RSS}_{\min}(K)$ (modulo a constant) if we view the model underlying K-means as a Gaussian mixture with hard assignment, uniform cluster priors and identical spherical covariance matrices (see Exercise 16.19).

The derivation of AIC is based on a number of assumptions, for example, that the data are independent and identically distributed. These assumptions are only approximately true for data sets in information retrieval. As a consequence, the AIC can rarely be applied without modification in text clustering. In Figure 16.8, the dimensionality of the vector space is $M \approx 50{,}000$. Thus, $2\mathrm{MK} > 50{,}000$ dominates the smaller RSS-based term ($\widehat{\mathrm{RSS}}_{\min}(1) < 5000$, not shown in the figure) and the minimum of the expression is reached for $K = 1$. But as we know, $K = 4$ (corresponding to the four classes *China*, *Germany*, *Russia*, and *Sports*) is a better choice than $K = 1$. In practice, Equation (16.11) is often more useful than Equation (16.13) – with the caveat that we need to come up with an estimate for λ.

? **Exercise 16.4** Why are documents that do not use the same term for the concept *car* likely to end up in the same cluster in K-means clustering?

Exercise 16.5 Two of the possible termination conditions for K-means were (i) assignment does not change, (ii) centroids do not change (page 332). Do these two conditions imply each other?

16.5 Model-based clustering

In this section, we describe a generalization of K-means, the EM algorithm. It can be applied to a larger variety of document representations and distributions than K-means.

In K-means, we attempt to find centroids that are good representatives. We can view the set of K centroids as a model that generates the data. Generating a document in this model consists of first picking a centroid at random and then adding some noise. If the noise is normally distributed, this procedure will result in clusters of spherical shape. *Model-based clustering* assumes that

MODEL-BASED CLUSTERING

the data were generated by a model and tries to recover the original model from the data. The model that we recover from the data then defines clusters and an assignment of documents to clusters.

A commonly used criterion for estimating the model parameters is maximum likelihood. In K-means, the quantity exp(−RSS) is proportional to the likelihood that a particular model (i.e., a set of centroids) generated the data. For K-means, maximum likelihood and minimal RSS are equivalent criteria. We denote the model parameters by Θ. In K-means, $\Theta = \{\vec{\mu}_1, \ldots, \vec{\mu}_K\}$.

More generally, the maximum likelihood criterion is to select the parameters Θ that maximize the log-likelihood of generating the data D:

$$\Theta = \arg\max_{\Theta} L(D|\Theta) = \arg\max_{\Theta} \log \prod_{n=1}^{N} P(d_n|\Theta) = \arg\max_{\Theta} \sum_{n=1}^{N} \log P(d_n|\Theta).$$

$L(D|\Theta)$ is the objective function that measures the goodness of the clustering. Given two clusterings with the same number of clusters, we prefer the one with higher $L(D|\Theta)$.

This is the same approach taken in Chapter 12 (page 218) for language modeling and in Section 13.1 (page 245) for text classification. In text classification, we chose the class that maximizes the likelihood of generating a particular document. Here, we choose the clustering Θ that maximizes the likelihood of generating a given set of documents. Once we have Θ, we can compute an assignment probability $P(d|\omega_k; \Theta)$ for each document–cluster pair. This set of assignment probabilities defines a soft clustering.

An example of a soft assignment is that a document about Chinese cars may have a fractional membership of 0.5 in each of the two clusters *China* and *automobiles*, reflecting the fact that both topics are pertinent. A hard clustering like K-means cannot model this simultaneous relevance to two topics.

Model-based clustering provides a framework for incorporating our knowledge about a domain. K-means and the hierarchical algorithms in Chapter 17 make fairly rigid assumptions about the data. For example, clusters in K-means are assumed to be spheres. Model-based clustering offers more flexibility. The clustering model can be adapted to what we know about the underlying distribution of the data, be it Bernoulli (as in the example in Table 16.3), Gaussian with nonspherical variance (another model that is important in document clustering) or a member of a different family.

EXPECTATION-MAXIMIZATION ALGORITHM

A commonly used algorithm for model-based clustering is the *expectation-maximization algorithm* or *EM algorithm*. EM clustering is an iterative algorithm that maximizes $L(D|\Theta)$. EM can be applied to many different types of probabilistic modeling. We will work with a mixture of multivariate Bernoulli distributions here, the distribution we know from Section 11.3 (page 204) and Section 13.3 (page 243):

$$P(d|\omega_k; \Theta) = \left(\prod_{t_m \in d} q_{mk}\right)\left(\prod_{t_m \notin d} (1 - q_{mk})\right) \tag{16.14}$$

where $\Theta = \{\Theta_1, \ldots, \Theta_K\}$, $\Theta_k = (\alpha_k, q_{1k}, \ldots, q_{Mk})$, and $q_{mk} = P(U_m = 1|\omega_k)$ are the parameters of the model.[3] $P(U_m = 1|\omega_k)$ is the probability that a document from cluster k contains term t_m. The probability α_k is the prior of cluster ω_k: the probability that a document d is in ω_k if we have no information about d.

The mixture model then is:

$$P(d|\Theta) = \sum_{k=1}^{K} \alpha_k \left(\prod_{t_m \in d} q_{mk} \right) \left(\prod_{t_m \notin d} (1 - q_{mk}) \right). \tag{16.15}$$

In this model, we generate a document by first picking a cluster ω_k with probability α_k and then generating the terms of the document according to the parameters q_{mk}. Recall that the document representation of the multivariate Bernoulli is a vector of M Boolean values (and not a real-valued vector).

How do we use EM to infer the parameters of the clustering from the data? That is, how do we choose parameters Θ that maximize $L(D|\Theta)$? EM is similar to K-means in that it alternates between an *expectation step*, corresponding to reassignment, and a *maximization step*, corresponding to recomputation of the parameters of the model. The parameters of K-means are the centroids, the parameters of the instance of EM in this section are the α_k and q_{mk}.

EXPECTATION STEP

MAXIMIZATION STEP

The maximization step recomputes the conditional parameters q_{mk} and the priors α_k as follows:

$$\textbf{Maximization step:} \quad q_{mk} = \frac{\sum_{n=1}^{N} r_{nk} I(t_m \in d_n)}{\sum_{n=1}^{N} r_{nk}} \qquad \alpha_k = \frac{\sum_{n=1}^{N} r_{nk}}{N} \tag{16.16}$$

where $I(t_m \in d_n) = 1$ if $t_m \in d_n$ and 0 otherwise and r_{nk} is the soft assignment of document d_n to cluster k as computed in the preceding iteration. (We'll address the issue of initialization in a moment.) These are the maximum likelihood estimates for the parameters of the multivariate Bernoulli from Table 13.3 (page 248) except that documents are assigned fractionally to clusters here. These maximum likelihood estimates maximize the likelihood of the data given the model.

The expectation step computes the soft assignment of documents to clusters given the current parameters q_{mk} and α_k:

$$\textbf{Expectation step:} \quad r_{nk} = \frac{\alpha_k (\prod_{t_m \in d_n} q_{mk})(\prod_{t_m \notin d_n} (1 - q_{mk}))}{\sum_{k=1}^{K} \alpha_k (\prod_{t_m \in d_n} q_{mk})(\prod_{t_m \notin d_n} (1 - q_{mk}))}. \tag{16.17}$$

This expectation step applies Equations (16.14) and (16.15) to computing the likelihood that ω_k generated document d_n. It is the classification procedure

[3] U_m is the random variable we defined in Section 13.3 (page 246) for the Bernoulli Naive Bayes model. It takes the values 1 (term t_m is present in the document) and 0 (term t_m is absent in the document).

Table 16.3 The EM clustering algorithm. The table shows a set of documents (a) and parameter values for selected iterations during EM clustering (b). Parameters shown are prior α_1, soft assignment scores $r_{n,1}$ (both omitted for cluster 2), and lexical parameters $q_{m,k}$ for a few terms. The authors initially assigned document 6 to cluster 1 and document 7 to cluster 2 (iteration 0). EM converges after 25 iterations. For smoothing, the r_{nk} in Equation (16.16) were replaced with $r_{nk} + \epsilon$ where $\epsilon = 0.0001$.

(a)

docID	document text	docID	document text
1	hot chocolate cocoa beans	7	sweet sugar
2	cocoa ghana africa	8	sugar cane brazil
3	beans harvest ghana	9	sweet sugar beet
4	cocoa butter	10	sweet cake icing
5	butter truffles	11	cake black forest
6	sweet chocolate		

(b)

	iteration of clustering							
parameter	0	1	2	3	4	5	15	25
α_1		0.50	0.45	0.53	0.57	0.58	0.54	0.45
$r_{1,1}$		1.00	1.00	1.00	1.00	1.00	1.00	1.00
$r_{2,1}$		0.50	0.79	0.99	1.00	1.00	1.00	1.00
$r_{3,1}$		0.50	0.84	1.00	1.00	1.00	1.00	1.00
$r_{4,1}$		0.50	0.75	0.94	1.00	1.00	1.00	1.00
$r_{5,1}$		0.50	0.52	0.66	0.91	1.00	1.00	1.00
$r_{6,1}$	1.00	1.00	1.00	1.00	1.00	1.00	0.83	0.00
$r_{7,1}$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$r_{8,1}$		0.00	0.00	0.00	0.00	0.00	0.00	0.00
$r_{9,1}$		0.00	0.00	0.00	0.00	0.00	0.00	0.00
$r_{10,1}$		0.50	0.40	0.14	0.01	0.00	0.00	0.00
$r_{11,1}$		0.50	0.57	0.58	0.41	0.07	0.00	0.00
$q_{\text{africa},1}$		0.000	0.100	0.134	0.158	0.158	0.169	0.200
$q_{\text{africa},2}$		0.000	0.083	0.042	0.001	0.000	0.000	0.000
$q_{\text{brazil},1}$		0.000	0.000	0.000	0.000	0.000	0.000	0.000
$q_{\text{brazil},2}$		0.000	0.167	0.195	0.213	0.214	0.196	0.167
$q_{\text{cocoa},1}$		0.000	0.400	0.432	0.465	0.474	0.508	0.600
$q_{\text{cocoa},2}$		0.000	0.167	0.090	0.014	0.001	0.000	0.000
$q_{\text{sugar},1}$		0.000	0.000	0.000	0.000	0.000	0.000	0.000
$q_{\text{sugar},2}$		1.000	0.500	0.585	0.640	0.642	0.589	0.500
$q_{\text{sweet},1}$		1.000	0.300	0.238	0.180	0.159	0.153	0.000
$q_{\text{sweet},2}$		1.000	0.417	0.507	0.610	0.640	0.608	0.667

for the multivariate Bernoulli in Table 13.3. Thus, the expectation step is nothing else but Bernoulli Naive Bayes classification (including normalization, i.e. dividing by the denominator, to get a probability distribution over clusters).

We clustered a set of eleven documents into two clusters using EM in Table 16.3. After convergence in iteration 25, the first five documents are assigned to cluster 1 ($r_{i,1} = 1.00$) and the last six to cluster 2 ($r_{i,1} = 0.00$). Somewhat atypically, the final assignment is a hard assignment here. EM

usually converges to a soft assignment. In iteration 25, the prior α_1 for cluster 1 is $5/11 \approx 0.45$ because five of the eleven documents are in cluster 1. Some terms are quickly associated with one cluster because the initial assignment can "spread" to them unambiguously. For example, membership in cluster 2 spreads from document 7 to document 8 in the first iteration because they share sugar. ($r_{8,1} = 0$ in iteration 1.) For parameters of terms occurring in ambiguous contexts, convergence takes longer. Seed documents 6 and 7 both contain sweet. As a result, it takes 25 iterations for the term to be unambiguously associated with cluster 2 ($q_{\text{sweet},1} = 0$ in iteration 25).

Finding good seeds is even more critical for EM than for K-means. EM is prone to get stuck in local optima if the seeds are not chosen well. This is a general problem that also occurs in other applications of EM.[4] Therefore, as with K-means, the initial assignment of documents to clusters is often computed by a different algorithm. For example, a hard K-means clustering may provide the initial assignment, which EM can then "soften up."

? **Exercise 16.6** We saw above that the time complexity of K-means is $\Theta(IKNM)$. What is the time complexity of EM?

Exercise 16.7 Let Ω be a clustering that exactly reproduces a class structure $\mathbb{C}$ and Ω' a clustering that further subdivides some clusters in Ω. Show that $I(\Omega;\mathbb{C}) = I(\Omega';\mathbb{C})$.

Exercise 16.8 Show that $I(\Omega;\mathbb{C}) \leq [H(\Omega) + H(\mathbb{C})]/2$.

Exercise 16.9 Mutual information is symmetric in the sense that its value does not change if the roles of clusters and classes are switched: $I(\Omega;\mathbb{C}) = I(\mathbb{C};\Omega)$. Which of the other three evaluation measures are symmetric in this sense?

Exercise 16.10 Compute RSS for the two clusterings in Figure 16.7.

Exercise 16.11 (i) Give an example of a set of points and three initial centroids (which need not be members of the set of points) for which 3-means converges to a clustering with an empty cluster. (ii) Can a clustering with an empty cluster be the global optimum with respect to RSS?

Exercise 16.12 Download Reuters-21578. Discard documents that do not occur in one of the ten classes *acquisitions*, *corn*, *crude*, *earn*, *grain*, *interest*, *money-fx*, *ship*, *trade*, and *wheat*. Discard documents that occur in two of these ten classes. (i) Compute a K-means clustering of this subset into ten clusters. There are a number of software packages that implement K-means, such as, WEKA (Witten and Frank 2005) and R (R Development Core Team 2005). (ii) Compute purity, normalized mutual information, F_1

[4] For example, this problem is common when EM is used to estimate parameters of hidden Markov models, probabilistic grammars, and machine translation models in natural language processing (Manning and Schütze 1999).

and RI for the clustering with respect to the ten classes. (iii) Compile a confusion matrix (Table 14.5, page 283) for the ten classes and ten clusters. Identify classes that give rise to FPs and FNs.

Exercise 16.13 Prove that $\text{RSS}_{\text{min}}(K)$ is monotonically decreasing in K.

Exercise 16.14 There is a soft version of K-means that computes the fractional membership of a document in a cluster as a monotonically decreasing function of the distance Δ from its centroid, for example, as $e^{-\Delta}$. Modify reassignment and recomputation steps of hard K-means for this soft version.

Exercise 16.15 In the last iteration in Table 16.3, document 6 is in cluster 2 even though it was the initial seed for cluster 1. Why does the document change membership?

Exercise 16.16 The values of the parameters q_{mk} in iteration 25 in Table 16.3 are rounded. What are the exact values that EM will converge to?

Exercise 16.17 Perform a K-means clustering for the documents in Table 16.3. After how many iterations does K-means converge? Compare the result with the EM clustering in Table 16.3 and discuss the differences.

Exercise 16.18 [⋆ ⋆ ⋆] Modify the expectation and maximization steps of EM for a Gaussian mixture. The maximization step computes the maximum likelihood parameter estimates α_k, $\vec{\mu}_k$, and Σ_k for each of the clusters. The expectation step computes for each vector a soft assignment to clusters (Gaussians) based on their current parameters. Write down the equations for Gaussian mixtures corresponding to Equations (16.16) and (16.17).

Exercise 16.19 [⋆ ⋆ ⋆] Show that K-means can be viewed as the limiting case of EM for Gaussian mixtures if variance is very small and all covariances are 0.

WITHIN-POINT SCATTER

Exercise 16.20 [⋆ ⋆ ⋆] The *within-point scatter* of a clustering is defined as $\sum_k \frac{1}{2} \sum_{\vec{x}_i \in \omega_k} \sum_{\vec{x}_j \in \omega_k} |\vec{x}_i - \vec{x}_j|^2$. Show that minimizing RSS and minimizing within-point scatter are equivalent.

Exercise 16.21 [⋆ ⋆ ⋆] Derive an AIC criterion for the multivariate Bernoulli mixture model from Equation (16.12).

16.6 References and further reading

Berkhin (2006b) gives a general up-to-date survey of clustering methods with special attention to scalability. The classic reference for clustering in pattern recognition, covering both K-means and EM, is (Duda et al. 2000).

Rasmussen (1992) introduces clustering from an information retrieval perspective. Anderberg (1973) provides a general introduction to clustering for applications. In addition to Euclidean distance and cosine similarity, Kullback-Leibler divergence is often used in clustering as a measure of how (dis)similar documents and clusters are (Xu and Croft 1999; Muresan and Harper 2004; Kurland and Lee 2004).

The cluster hypothesis is due to Jardine and van Rijsbergen (1971) who state it as follows: *Associations between documents convey information about the relevance of documents to requests.* Salton (1971a, 1975), Croft (1978), Voorhees (1985a), Can and Ozkarahan (1990), Cacheda et al. (2003), Can et al. (2004), Singitham et al. (2004), and Altingövde et al. (2008) investigate the efficiency and effectiveness of cluster-based retrieval. Although some of these studies show improvements in effectiveness, efficiency, or both, there is no consensus that cluster-based retrieval works well consistently across scenarios. Cluster-based language modeling was pioneered by Liu and Croft (2004).

There is good evidence that clustering of search results improves user experience and search result quality (Hearst and Pedersen 1996; Zamir and Etzioni 1999; Tombros et al. 2002; Käki 2005; Toda and Kataoka 2005); although not as much as search result structuring based on carefully edited category hierarchies (Hearst 2006). The scatter-gather interface for browsing collections was presented by Cutting et al. (1992). A theoretical framework for analyzing the properties of scatter-gather and other information seeking user interfaces is presented by Pirolli (2007). Schütze and Silverstein (1997) evaluate LSI (Chapter 18) and truncated representations of centroids for efficient K-means clustering.

The Columbia NewsBlaster system (McKeown et al. 2002), a forerunner to the now much more famous and refined Google News (http://news.google.com), used hierarchical clustering (Chapter 17) to give two levels of news topic granularity. See Hatzivassiloglou et al. (2000) for details, and Chen and Lin (2000) and Radev et al. (2001) for related systems. Other applications of clustering in information retrieval are duplicate detection (Yang and Callan (2006), Section 19.6, page 400), novelty detection (see references in Section 17.9, page 367) and metadata discovery on the semantic web (Alonso et al. 2006).

The discussion of external evaluation measures is partially based on Strehl (2002). Dom (2002) proposes a measure Q_0 that is better motivated theoretically than NMI. Q_0 is the number of bits needed to transmit class memberships assuming cluster memberships are known. The Rand index is due to Rand (1971). Hubert and Arabie (1985) propose an *adjusted Rand index* that ranges between -1 and 1 and is 0 if there is only chance agreement between clusters and classes (similar to κ in Chapter 8, page 152). Basu et al. (2004) argue that the three evaluation measures NMI, Rand index, and F measure give very similar results. Stein et al. (2003) propose *expected edge density* as an

ADJUSTED RAND INDEX

internal measure and give evidence that it is a good predictor of the quality of a clustering. Kleinberg (2002) and Meilă (2005) present axiomatic frameworks for comparing clusterings.

Authors that are often credited with the invention of the K-means algorithm include Lloyd (1982) (first distributed in 1957), Ball (1965), MacQueen (1967), and Hartigan and Wong (1979). Arthur and Vassilvitskii (2006) investigate the worst-case complexity of K-means. Bradley and Fayyad (1998), Pelleg and Moore (1999), and Davidson and Satyanarayana (2003) investigate the convergence properties of K-means empirically and how it depends on initial seed selection. Dhillon and Modha (2001) compare K-means clusters with SVD-based clusters (Chapter 18). The K-medoid algorithm was presented by Kaufman and Rousseeuw (1990). The EM algorithm was originally introduced by Dempster et al. (1977). An in-depth treatment of EM is (McLachlan and Krishnan 1996). See Section 18.5 (page 383) for publications on latent analysis, which can also be viewed as soft clustering.

AIC is due to Akaike (1974) (see also Burnham and Anderson (2002)). An alternative to AIC is BIC, which can be motivated as a Bayesian model selection procedure (Schwarz 1978). Fraley and Raftery (1998) show how to choose an optimal number of clusters based on BIC. An application of BIC to K-means is (Pelleg and Moore 2000). Hamerly and Elkan (2003) propose an alternative to BIC that performs better in their experiments. Another influential Bayesian approach for determining the number of clusters (simultaneously with cluster assignment) is described by Cheeseman and Stutz (1996). Two methods for determining cardinality without external criteria are presented by Tibshirani et al. (2001).

We only have space here for classical completely unsupervised clustering. An important current topic of research is how to use prior knowledge to guide clustering (e.g., Ji and Xu (2006)) and how to incorporate interactive feedback during clustering (e.g., Huang and Mitchell (2006)). Fayyad et al. (1998) propose an initialization for EM clustering. For algorithms that can cluster very large data sets in one scan through the data see Bradley et al. (1998).

The applications in Table 16.1 all cluster documents. Other information retrieval applications cluster words (e.g., Crouch 1988), contexts of words (e.g., Schütze and Pedersen 1995) or words and documents simultaneously (e.g., Tishby and Slonim 2000, Dhillon 2001, Zha et al. 2001). Simultaneous clustering of words and documents is an example of *co-clustering* or *biclustering*.

CO-CLUSTERING

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 information retrieval (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 335).

This chapter first introduces *agglomerative* hierarchical clustering (Section 17.1) and presents four different agglomerative algorithms, in Sections 17.2 through 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 323; see also Section 16.6, page 343). In fact, the example we gave for collection

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

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, nondeterminism) 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.

HIERARCHICAL AGGLOMERATIVE CLUSTERING (HAC)

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.

Before looking at specific similarity measures used in HAC in Sections 17.2 through 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

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).

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 twenty-nine 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-

INVERSION

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.

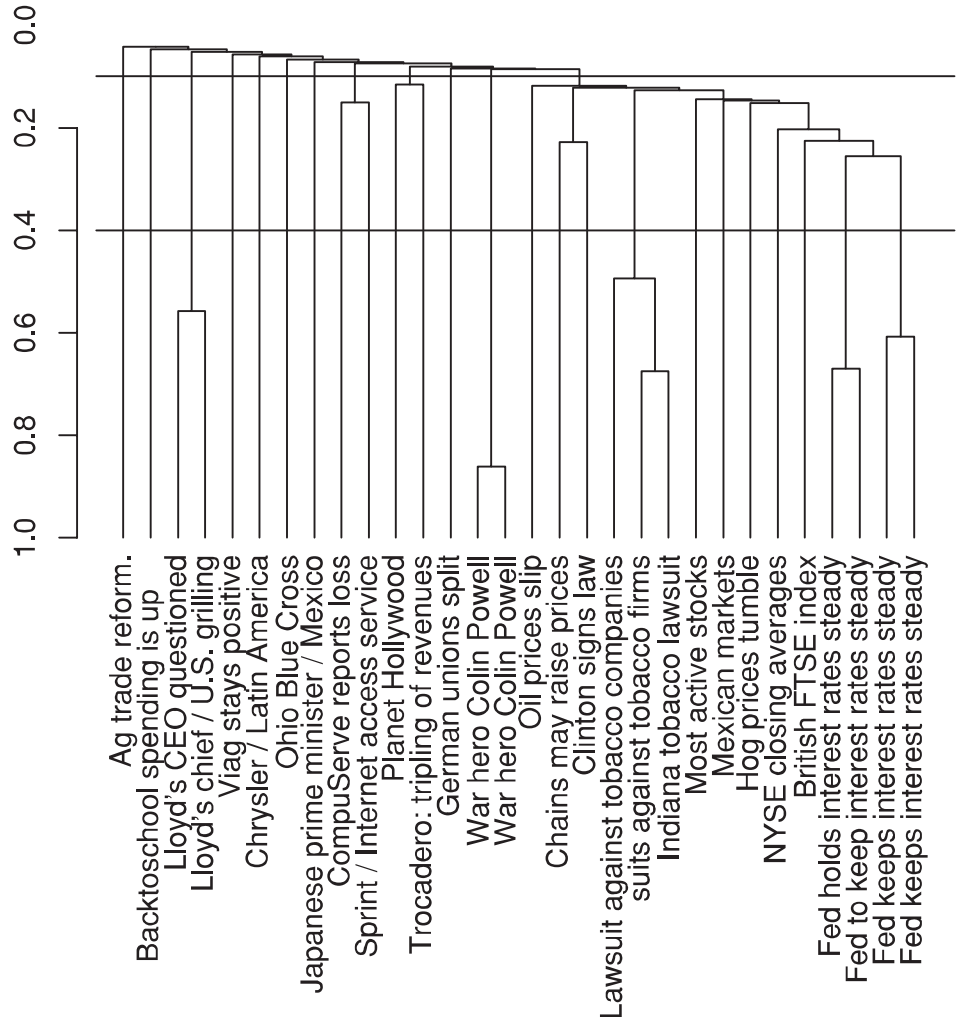


Figure 17.1 A dendrogram of a single-link clustering of thirty documents from Reuters-RCV1. Two possible cuts of the dendrogram are shown: at 0.4 into twenty-four clusters and at 0.1 into twelve clusters.

Hierarchical clustering does not require a prespecified number of clusters. However, 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 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 twenty-four clusters (grouping only documents with high similarity together) and cutting it at $y = 0.1$ yields twelve clusters (one large financial news cluster and eleven 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 337).

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 $A \leftarrow []$ *(collects 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.

- Apply Equation (16.11) (page 337):

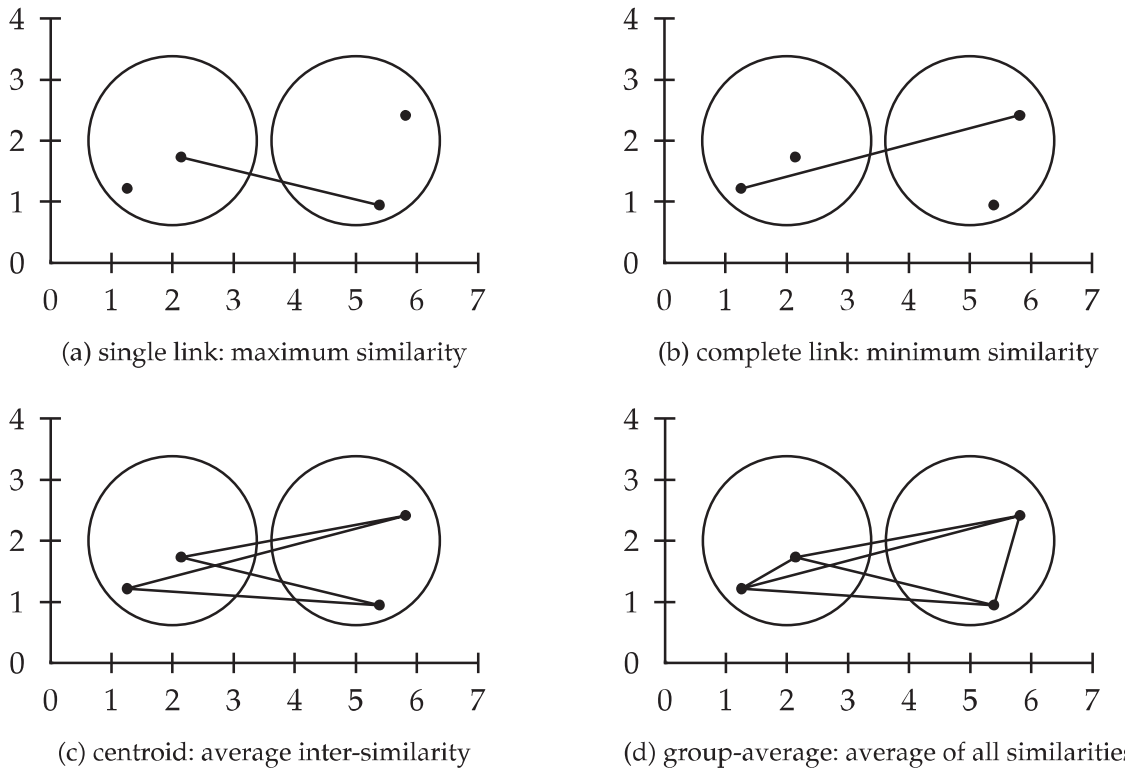
$$K = \arg\min_{K'} [\text{RSS}(K') + \lambda K']$$

  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 SIM(i, m, j) computes the similarity of cluster j with the merge of clusters i and m. For some HAC algorithms, 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.



(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.

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 nonlocal; 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

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

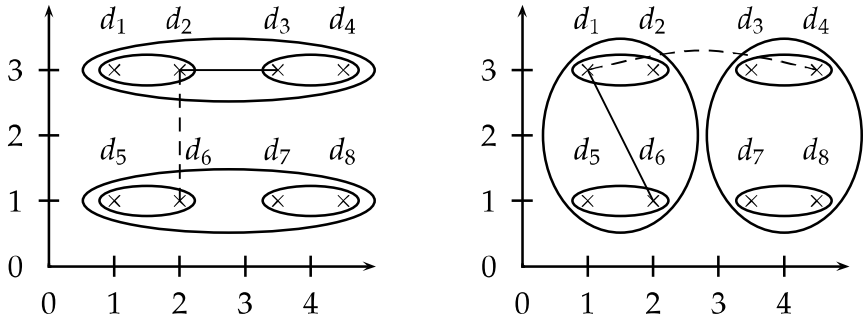


Figure 17.4 A single-link (*left*) and complete-link (*right*) clustering of eight documents. The ellipses correspond to successive clustering stages. *Left*: The single-link similarity of the two upper two-point clusters is the similarity of d_2 and d_3 (*solid line*), which is greater than the single-link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

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.[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

[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.

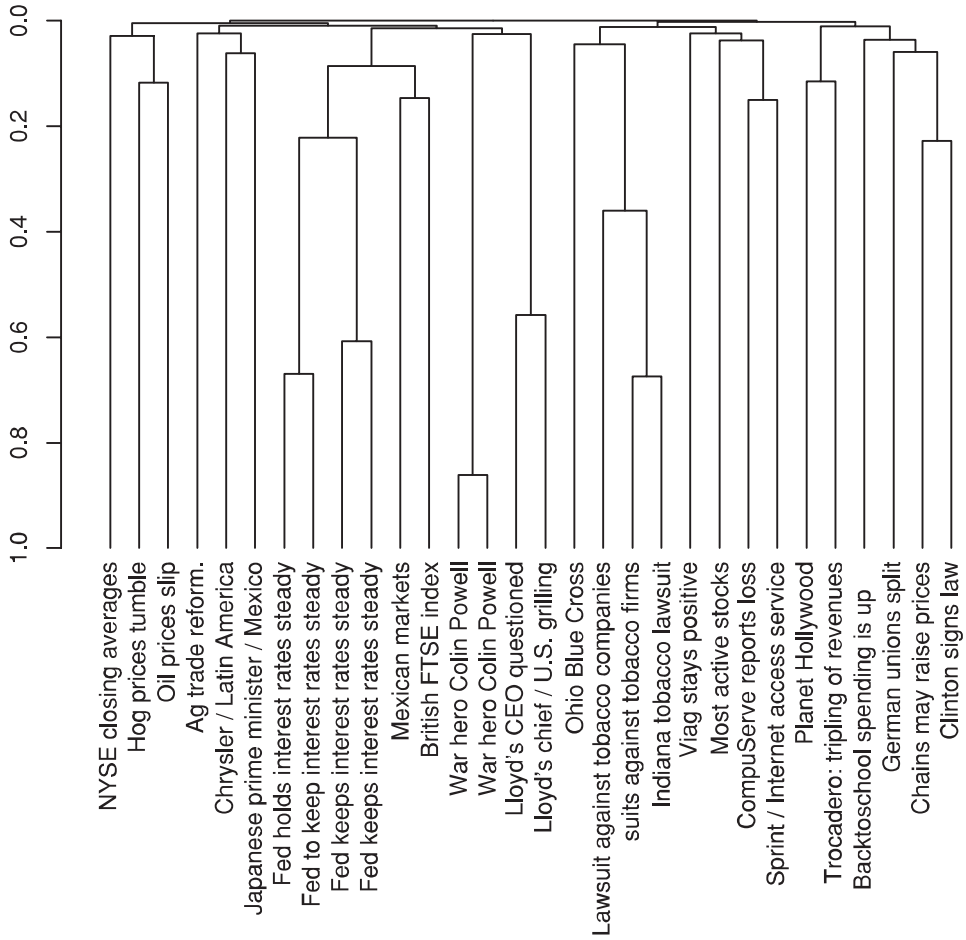


Figure 17.5 A dendrogram of a complete-link clustering. The same thirty documents were clustered with single-link clustering in Figure 17.1.

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. Because 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*.

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

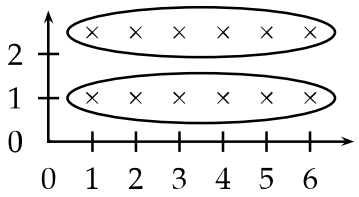


Figure 17.6 Chaining in single-link clustering. The local criterion in single-link clustering can cause undesirable elongated clusters.

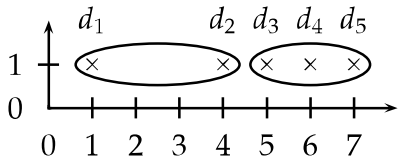


Figure 17.7 Outliers in complete-link clustering. The five documents have the x-coordinates $1 + 2\epsilon$, 4, $5 + 2\epsilon$, 6 and $7 - \epsilon$. Complete-link clustering creates the two clusters shown as ellipses. The most intuitive two-cluster clustering is $\{\{d_1\}, \{d_2, d_3, d_4, d_5\}\}$,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

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 has 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

The complexity of the naive HAC algorithm in Figure 17.2 is $\Theta(N^3)$ because we exhaustively search 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 (Section 17.4). We give an example of how a row of C is processed (Figure 17.8, bottom panel). Both high-level loops (lines 1–7 and 9–21) are $\Theta(N^2 \log N)$ 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.

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 $P[n]$.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.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** $P[i]$.DELETE($C[i][k_1]$)
17 $P[i]$.DELETE($C[i][k_2]$)
18 $C[i][k_1].\text{sim} \leftarrow \text{SIM}(i, k_1, k_2)$
19 $P[i]$.INSERT($C[i][k_1]$)
20 $C[k_1][i].\text{sim} \leftarrow \text{SIM}(i, k_1, k_2)$
21 $P[k_1]$.INSERT($C[k_1][i]$)
22 **return** A

clustering algorithm	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($d_1, \ldots, d_N$)
1 **for** $n \leftarrow 1$ **to** N
2 **do for** $i \leftarrow 1$ **to** N
3 **do** $C[n][i].\text{sim} \leftarrow \text{SIM}(d_n, d_i)$
4 $C[n][i].\text{index} \leftarrow i$
5 $I[n] \leftarrow n$
6 $NBM[n] \leftarrow \arg\max_{X \in \{C[n][i]: n \neq i\}} X.\text{sim}$
7 $A \leftarrow []$
8 **for** $n \leftarrow 1$ **to** $N - 1$
9 **do** $i_1 \leftarrow \arg\max_{\{i: I[i]=i\}} NBM[i].\text{sim}$
10 $i_2 \leftarrow I[NBM[i_1].\text{index}]$
11 A.APPEND($\langle i_1, i_2 \rangle$)
12 **for** $i \leftarrow 1$ **to** N
13 **do if** $I[i] = i \wedge i \neq i_1 \wedge i \neq i_2$
14 **then** $C[i_1][i].\text{sim} \leftarrow C[i][i_1].\text{sim} \leftarrow \max(C[i_1][i].\text{sim}, C[i_2][i].\text{sim})$
15 **if** $I[i] = i_2$
16 **then** $I[i] \leftarrow i_1$
17 $NBM[i_1] \leftarrow \arg\max_{X \in \{C[i_1][i]: I[i]=i \wedge i \neq i_1\}} X.\text{sim}$
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 $NBM[i_1]$.

The argument of EFFICIENTHAC in Figure 17.8 is a set of vectors (as opposed to a set of generic documents) because group-average agglomerative clustering 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 $\Theta(N^2)$.

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

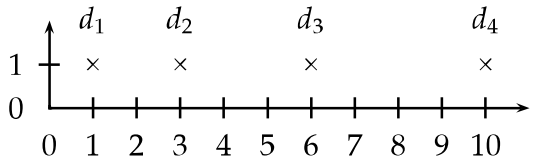


Figure 17.10 Complete-link clustering is not best-merge persistent. At first, d_2 is the best-merge cluster for d_3. But after merging d_1 and d_2, d_4 becomes d_3's best-merge candidate. In a best-merge persistent algorithm like single-link, d_3's best-merge cluster would be $\{d_1, d_2\}$.

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 nonlocal 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 because computing the similarity between two documents (e.g., as a dot product) is an order of magnitude slower than a comparison of two values 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:

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

With (17.2), we have:

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

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 implimentations 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. Because 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. Although 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 267), 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 for computing the 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. So 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:

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

where the centroid $\vec{\mu}(\omega)$ is defined as in Equation (14.1) (page 269). 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; they will have proportionally more self-similarities. For two documents d_1, d_2 with a similarity s, we have SIM-GA$'(d_1, d_2) = (1 + s)/2$. In contrast, SIM-GA$(d_1, d_2) = s \leq (1 + s)/2$. This similarity 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 threedimensional space to get length-normalized vectors. Is the group-average clustering different from the single-link and complete-link clusterings?

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).

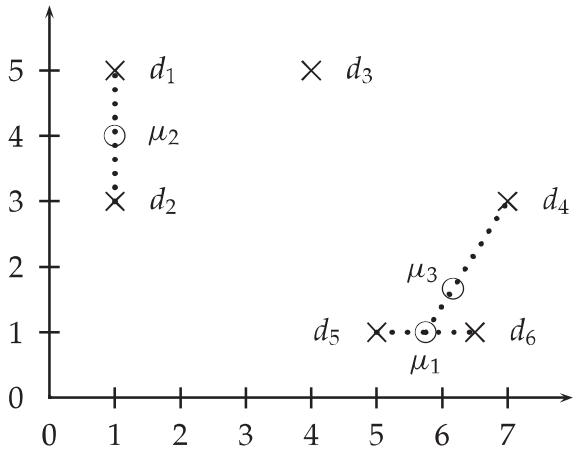


Figure 17.11 Three iterations of centroid clustering. Each iteration merges the two clusters whose centroids are closest.

INVERSION

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

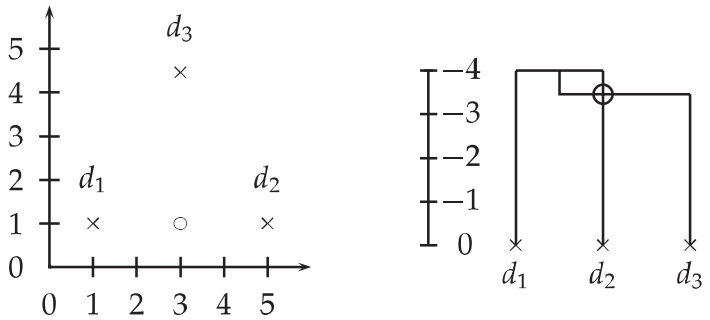


Figure 17.12 Centroid clustering is not monotonic. The documents d_1 at $(1+\epsilon, 1)$, d_2 at $(5, 1)$, and d_3 at $(3, 1+2\sqrt{3})$ are almost equidistant, with d_1 and d_2 closer to each other than to d_3. The non-monotonic inversion in the hierarchical clustering of the three points appears as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

? **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 hierarchical agglomerative clustering

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 347).

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 because 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

COMBINATION SIMILARITY

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.

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 possible 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.

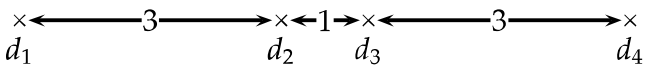
The inductive basis of the proof is that a clustering with $K = N$ clusters has combination similarity 1.0, which is the largest value possible. The induction hypothesis is that a single-link clustering Ω_K with K clusters is optimal: COMB-SIM(Ω_K) > COMB-SIM(Ω'_K) for all Ω'_K. Assume for contradiction 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 COMB-SIM(Ω'_{K-1}) > COMB-SIM(Ω_{K-1}).

Case 1: The two documents linked by s = COMB-SIM(Ω'_{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 $s >$ COMB-SIM(Ω_K). Thus, COMB-SIM(Ω'_{K-1}) $= s >$ COMB-SIM(Ω_K) > COMB-SIM(Ω'_K) > COMB-SIM(Ω'_{K-1}). Contradiction.

Case 2: The two documents linked by s = COMB-SIM(Ω'_{K-1}) are not in the same cluster in Ω_K. But s = COMB-SIM(Ω'_{K-1}) > COMB-SIM(Ω_{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

Table 17.1 Comparison of HAC algorithms.

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

because it is generally the method that produces the clustering with the best 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 nonvector 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 403) 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 347 and the static definition on page 360.

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 algorithm.

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; 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 323), 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 251) can all be used for differential cluster labeling.[5] In particular, mutual information (MI) (Section 13.5.1, page 252) or, equivalently, information gain and the χ^2 test (Section 13.5.2, page 255) 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 did not make it into the top ten terms selected by MI. On the web, anchor text

[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.

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.

		labeling method		
	# docs	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

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.

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 selected 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 k nearest neighbor (kNN) classification (see Section 14.7, page 291).

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 336) 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 332). 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

BUCKSHOT ALGORITHM

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.

?

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, and (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 four 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 347.

Exercise 17.12 For N points, there are $\leq N^K$ different flat clusterings into K clusters (Section 16.2, page 327). 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?

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 332). 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 multidocument summarization (McKeown and Radev 1995) is also applicable to cluster labeling, but multidocument summaries are usually longer than the short text fragments needed when labeling clusters (cf. Section 8.7, page 157). Presenting clusters in a way that users can understand is a UI problem. We recommend reading (Baeza-Yates and Ribeiro-Neto 1999, Chapter 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 327) 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 336). Support for clustering vectors in high-dimensional spaces is provided by the software package CLUTO (http://glaros.dtc.umn.edu/gkhome/views/cluto).