CHAPTER 4

SIMILARITY AND CLUSTERING

Keyword query processing and response ranking, described in Chapter 3, depend on computing a measure of *similarity* between the query and documents in the collection. Although the query is regarded at par with the documents in the vector-space model, it is usually much shorter and prone to ambiguity (the average Web query is only two to three words long). For example, the query *star* is highly ambiguous, retrieving documents about astronomy, plants and animals, popular media and sports figures, and American patriotic songs. Their vector-space similarity (see Chapter 3) to the single-word query may carry no hint that documents pertaining to these topics are highly dissimilar. However, if the search *clusters* the responses along the lines of these topics, as shown in Figure 4.1, the user can quickly disambiguate the query or drill down into a specific topic.

Apart from visualization of search results, clustering is useful for taxonomy design and similarity search. Topic taxonomies such as Yahoo! and the Open Directory (*dmoz.org/*) are constructed manually, but this process can be greatly facilitated by a preliminary clustering of large samples of Web documents. Clustering can also assist fast similarity search, described in Section 3.3.1. Given a precomputed clustering of the corpus, the search for documents similar to a query document d_q may be efficiently limited to a small number of clusters that are most similar to d_q, quickly eliminating a large number of documents that we can safely surmise would rank poorly.

Similarity, in a rather general way, is fundamental to many search and mining operations on hypertext and is central to most of this book. In this chapter we will study how measures of similarity are used to cluster a collection of documents into

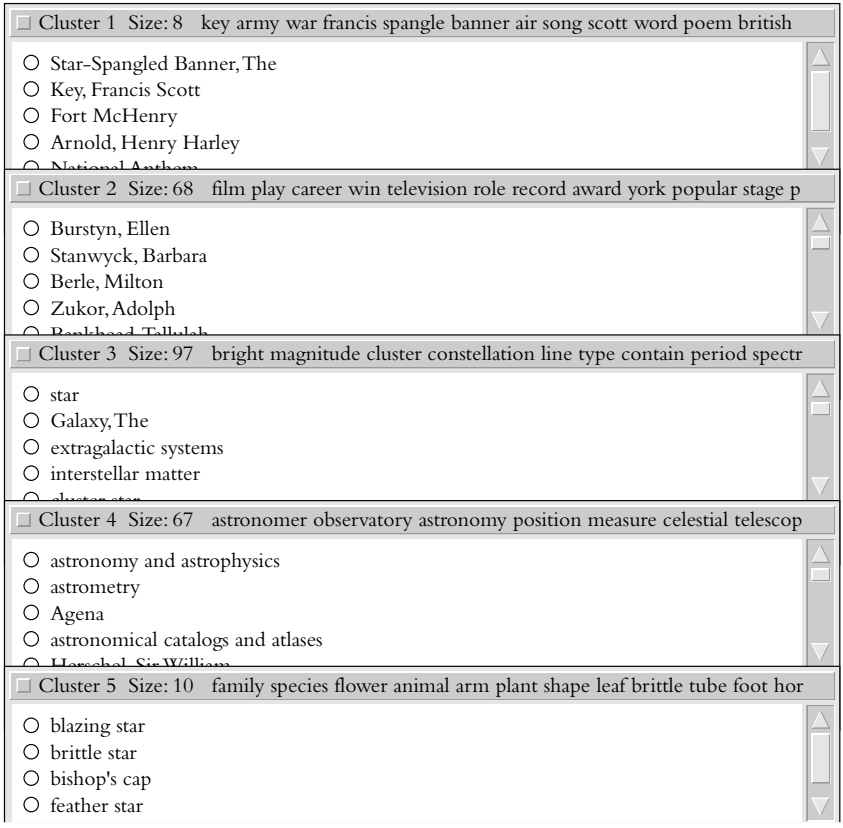


FIGURE 4.1 Scatter/Gather, a text clustering system, can separate salient topics in response to keyword queries. (Image courtesy of Hearst [101].)

groups within which interdocument similarity is large compared to the similarity between documents chosen from different groups. The utility of clustering for text and hypertext information retrieval lies in the so-called *cluster hypothesis*: given a "suitable" clustering of a collection, if the user is interested in document d, she is likely to be interested in other members of the cluster to which d belongs.

The cluster hypothesis is not limited to documents alone. If documents are similar because they share terms, terms can also be represented as bit-vectors representing the documents in which they occur, and these bit-vectors can be used to cluster the terms. As with terms and documents, we can set up a bipartite

relation for people liking documents, and use this to cluster both people and documents, with the premise that similar people like similar documents, and vice versa. This important ramification of clustering is called *collaborative filtering*.

This chapter is organized as follows: I start with an overview of basic formulations and approaches to clustering (Section 4.1). Then I describe two important clustering paradigms: a bottom-up agglomerative technique (Section 4.2.1), which collects similar documents into larger and larger groups, and a top-down partitioning technique (Section 4.2.2), which divides a corpus into topic-oriented partitions. These are followed by a slew of clustering techniques that can be broadly classified as *embeddings* of the corpus in a low-dimensional space so as to bring out the clustering present in the data (Section 4.3). Next, I discuss probabilistic models and algorithms in Section 4.4, and end the chapter with a discussion of collaborative filtering.

4.1 Formulations and Approaches

Formulations of clustering problems range from combinatorial to fuzzy, and no single objective serves all applications. Most of the combinatorial definitions are intractable to optimize. Clustering is a classic applied art where a great deal of experience with data must supplement stock algorithms. It is beyond the scope of a single chapter to cover the entire breadth of the subject, but there are many classic books on it. My goal is to highlight broad classes of algorithms and the specific issues that arise when one seeks to find structure in text and hypertext domains.

I first propose a few formal specifications of the clustering problem and outline some basic approaches to clustering. We are given a collection D of documents (in general, entities to be clustered). Entities either may be characterized by some *internal* property, such as the vector-space model for documents, or they may be characterized only *externally*, via a measure of distance (dissimilarity) $\delta(d_1, d_2)$ or resemblance (similarity) $\rho(d_1, d_2)$ specified between any two pairs of documents. For example, we can use the Euclidean distance between length-normalized document vectors for δ and cosine similarity for ρ. These measures have been discussed earlier, in Chapter 3.

4.1.1 Partitioning Approaches

One possible goal that we can set up for a clustering algorithm is to *partition* the document collection into k subsets or clusters $D_1, \ldots, D_k$ so as to minimize

the intracluster distance $\sum_i \sum_{d_1,d_2 \in D_i} \delta(d_1, d_2)$ or maximize the intracluster resemblance $\sum_i \sum_{d_1,d_2 \in D_i} \rho(d_1, d_2)$. If an internal representation of documents is available, then it is also usual to specify a representation of clusters with regard to that same model. For example, if documents are represented using the vector-space model, a cluster of documents may be represented by the centroid (average) of the document vectors. When a cluster representation is available, a modified goal could be to partition D into $D_1, \ldots, D_k$ so as to minimize $\sum_i \sum_{d \in D_i} \delta(d, \vec{D}_i)$ or maximize $\sum_i \sum_{d \in D_i} \rho(d, \vec{D}_i)$, where $\vec{D}_i$ is the vector-space representation of cluster i.

One could think of assigning document d to cluster i as setting a Boolean variable $z_{d,i}$ to 1. This can be generalized to *fuzzy* or *soft* clustering where $z_{d,i}$ is a real number between zero and one. In such a scenario, one may wish to find $z_{d,i}$ so as to minimize $\sum_i \sum_{d \in D} z_{d,i} \delta(d, \vec{D}_i)$ or maximize $\sum_i \sum_{d \in D} z_{d,i} \rho(d, \vec{D}_i)$.

Partitions can be found in two ways. We can start with each document in a group of its own, and collapse together groups of documents until the number of partitions is suitable; this is called *bottom-up* clustering. Alternatively, we can declare the number of partitions that we want a priori, and assign documents to partitions; this is called *top-down* clustering. I will discuss both variants in Section 4.2.

4.1.2 Geometric Embedding Approaches

The human eye is impressive at noticing patterns and clusters in data presented as points embedded in two or three dimensions, as borne out by the naming of constellations and archipelagoes. If there is natural clustering in the data, and we manage to embed or project the data points to two or three dimensions without losing the clustering property, the resulting "map" may itself be an adequate clustering aid.

I will discuss several approaches to creating clusters in low-dimensional space. In one approach, called *self-organizing maps*, clusters are laid out on a plane in a regular grid, and documents are iteratively assigned to regions of the plane. For this approach we need documents to be specified using an internal description. In another approach, called *multidimensional scaling*, the system input is the pairwise (dis-)similarity between documents. The algorithm seeks to embed the documents as points in 2D to 3D space with the minimum distortion of pairwise distances. Both of these approaches are heuristic in nature; there is no general guarantee that all collections can be rendered well. Another technique, called

Team-Fly®

latent semantic indexing, uses techniques from linear algebra to factor the term-document matrix. The factors can be used to derive a low-dimensional representation for documents as well as terms. This representation can also be used for ad hoc searching.

A different form of partition-based clustering is to identify *dense regions* in space. As an extreme example, we can start with a 1D space with a finite extent and a finite number of points, and claim that a cluster is demarcated by endpoints within which the number of points per unit length (density) is higher than (some multiple of) the average global density. Such a density-based notion of clustering can be readily extended to more dimensions. In particular, there may be no discernible clustering when the points are considered in the original space, but clusters may emerge when the points are projected to a subspace with a smaller number of dimensions. We can look for density-based clusters in a simple bottom-up fashion. The basic observation is that if a region is dense in k dimensions, then all projections of this region are dense. Therefore, the algorithm first finds 1D dense "regions," tries to compose them into 2D regions, discarding those that fail the density test, and so on. Unfortunately this method would not scale to textual data with tens of thousands of dimensions. The only way around seems to be to propose simple *generative distributions* for documents, discussed next.

4.1.3 Generative Models and Probabilistic Approaches

In the approaches outlined thus far, the measures of (dis-)similarity are provided by the user. Carelessly designed measures can easily damage the quality of clustering. The probabilistic approach seeks to model the document collection as being generated by a random process following a specific set of distributions. For example, we can assume that each cluster that we seek is associated with a distribution over the terms in our lexicon. Given the collection, we must estimate the number of distributions, and the parameters defining these distributions. Indeed, estimating these distributions can be *defined* as the clustering problem. We will study several techniques for estimating cluster distributions. Initially we will assume that each document is generated from exactly one distribution. However, in the common situation that clusters correspond to *topics*, a single-topic-per-document model is not entirely realistic: documents are often *mixtures* of multiple topics. The more advanced techniques that we will study can estimate models in this more general setting. (This part of the chapter is key to understanding many later chapters.)

Estimating a term distribution over documents is difficult. There is little hope of capturing the joint distribution between terms or term sequences, given the large number of terms in the vocabulary (tens to hundreds of thousands for many standard collections). Most practical models need to assume that term occurrences are independent of each other. Even if each term is associated with a simple Boolean event (the term occurs or does not occur in a document), the number of event combinations is astronomical compared to the size of any document collection that we are likely to encounter.

4.2 Bottom-Up and Top-Down Partitioning Paradigms

We will now study one bottom-up clustering technique that will repeatedly merge groups of similar documents until the desired number of clusters is attained, and a top-down technique that will iteratively refine the assignment of documents to a preset number of clusters. The former method is somewhat slower, but may be used on a small sample of the corpus to "seed" the initial clusters before the latter algorithm takes over.

4.2.1 Agglomerative Clustering

Although many formulations of the clustering problem are intractable, a simple, intuitive heuristic is to start with all the documents and successively combine them into groups within which interdocument similarity is high, collapsing down to as many groups as desired. This style is called *bottom-up, agglomerative,* or *hierarchical agglomerative clustering* (HAC) and is characterized by the broad pseudocode shown in Figure 4.2. HAC is widely used in document clustering and other IR applications [180, 213].

```
1: let each document d be in a singleton group {d}
2: let G be the set of groups
3: while |G| > 1 do
4:    choose Γ, Δ ∈ G according to some measure of similarity s(Γ, Δ)
5:    remove Γ and Δ from G
6:    let Φ = Γ ∪ Δ
7:    insert Φ into G
8: end while
```

FIGURE 4.2 Basic template for bottom-up hierarchical agglomerative clustering.

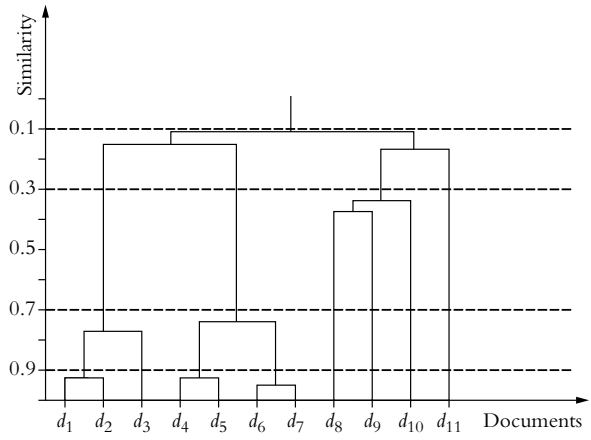


FIGURE 4.3 A dendrogram presents the progressive, hierarchy-forming merging process pictorially. The user can cut across the dendrogram at a suitable level of similarity to get the desired number of clusters. (Taken from [141].)

The hierarchical merging process leads to a tree called a *dendrogram*, drawn in the specific style shown in Figure 4.3. Typically, the earlier mergers happen between groups with a large similarity $s(\Gamma \cup \Delta)$. This value becomes lower and lower for later merges. The user can "cut across" the dendrogram at a suitable level to "read off" any desired number of clusters.

Algorithms differ as to how they compute the figure of merit for merging Γ and Δ. One commonly used measure is the *self-similarity* of $\Gamma \cup \Delta$. The self-similarity of a group of documents Φ is defined as the average pairwise similarity between documents in Φ

$$s(\Phi) = \frac{1}{\binom{|\Phi|}{2}} \sum_{d_1, d_2 \in \Phi} s(d_1, d_2) = \frac{2}{|\Phi|\,(|\Phi| - 1)} \sum_{d_1, d_2 \in \Phi} s(d_1, d_2) \tag{4.1}$$

where the TFIDF cosine measure is commonly used for interdocument similarity $s(d_1, d_2)$. Other merger criteria exist. One may choose to merge that pair of clusters (Γ, Δ), which maximizes $\min_{d_1 \in \Gamma, d_2 \in \Delta} s(d_1, d_2)$, $\max_{d_1 \in \Gamma, d_2 \in \Delta} s(d_1, d_2)$, or $(\sum_{d_1 \in \Gamma, d_2 \in \Delta} s(d_1, d_2))/(|\Gamma|\,|\Delta|)$.

In this section, we will denote a document as d and its corresponding vector-space representation as $\vec{d}$, which we will sometimes simplify to d if there is no

chance for confusion. If documents are already normalized to unit length in the L_2 norm, $s(d_1, d_2)$ is simply the dot-product, $\langle d_1, d_2 \rangle$.

By maintaining carefully chosen statistics, the dendrogram can be computed in about quadratic time and space. For any group Φ of documents, we maintain an unnormalized group profile vector

$$p(\Phi) = \sum_{d \in \Phi} \vec{d} \tag{4.2}$$

which is simply the sum of the document vectors belonging to that group, together with the number of documents in the group. It is easy to verify that

$$s(\Phi) = \frac{\langle p(\Phi), p(\Phi) \rangle - |\Phi|}{|\Phi|(|\Phi| - 1)} \tag{4.3}$$

and

$$p(\Gamma \cup \Delta) = \langle p(\Gamma), p(\Gamma) \rangle + \langle p(\Delta), p(\Delta) \rangle + 2 \langle p(\Gamma), p(\Delta) \rangle \tag{4.4}$$

Thus, in Figure 4.2, to compute $s(\Gamma \cup \Delta)$ from $p(\Gamma)$ and $p(\Delta)$ will cost just the time to do a few dot-products. For the moment we will assume that dimensionality of documents and group profiles are fixed, and therefore we can calculate a dot-product in constant time. (We will return to this issue.) Not much changes on each merge, so we would also like to maintain with each group Γ a *heap* [56] of partner groups Δ ordered by largest $s(\Gamma \cup \Delta)$. Thus, for each group, we can access its best partner (together with the score) in constant time. With n groups to start with, we precompute all pairwise similarities in $O(n^2)$ time, and insert them in heaps in $O(n^2 \log n)$ time. Now we can pick the best pair of groups to merge in $O(n)$ time, delete these groups from each of the heaps in $O(\log n)$ time, compute the similarity of the merger with old groups in $O(n)$ time, and update all heaps in $O(n \log n)$ time. Since there are $n - 1$ merges, the total time is $O(n^2 \log n)$, and all the heaps together consume $O(n^2)$ space.

Earlier we assumed that documents and group profile vectors are embedded in a space with a fixed number of dimensions. This is true, but the number of dimensions is rather large, running into tens of thousands. Moreover, the time taken to compute a dot-product is not really proportional to the number of dimensions, but only the number of nonzero coordinates, assuming a sparse vector representation is used (as it should be). For example, the Reuters collection [139] has about 22,000 financial news articles, a typical article having a few dozen to a few hundred distinct terms, whereas the total number of unique terms in the

collection is over 30,000. As a result, dot-product computations near the leaves of the dendrogram would be very fast, but would get slower as the group profiles become denser, until near the root, profile vectors are almost entirely dense. A simple way to reduce the running time is to *truncate* document and group profile vectors to a fixed number (e.g., 1000) of the largest magnitude coordinates. In theory, this may lead to a clustering output that is different from what would be computed with a full representation, but empirical evidence suggests that the quality of clustering remains unaffected [60, 191].

4.2.2 The *k*-Means Algorithm

Bottom-up clustering, used directly, takes quadratic time and space and is not practical for large document collections. If the user can preset a (small) number k of desired clusters, a more efficient top-down partitioning strategy may be used. The best-known member of this family of algorithms is the k-means algorithm. We will discuss two forms of the k-means algorithm here. One makes "hard" (0/1) assignments of documents to clusters. The other makes "soft" assignments, meaning documents belong to clusters with a fractional score between 0 and 1.

k-means with "hard" assignment

In its common form, k-means uses internal representations for both the objects being clustered and the clusters themselves. For documents, the vector-space representation is used, and the cluster is represented as the centroid of the documents belonging to that cluster.

The initial configuration is arbitrary (or chosen by a heuristic external to the k-means algorithm), consisting of a grouping of the documents into k groups, and k corresponding vector-space centroids computed accordingly. Thereafter, the algorithm proceeds in alternating half-steps, as shown in Figure 4.4.

The basic step in k-means is also called *move-to-nearest,* for obvious reasons. A variety of criteria may be used for terminating the loop. One may exit when the assignment of documents to clusters ceases to change (much), or when cluster centroids move by negligible distances in successive iterations.

k-means with "soft' assignment

Rather than make any specific assignment of documents to clusters, the "soft" variant of k-means represents each cluster c using a vector μ_c in term space. Since there is no explicit assignment of documents to clusters, μ_c is not directly related to documents—for example, it is not necessarily the centroid of some documents.

```
 1: initialize cluster centroids to arbitrary vectors
 2: while further improvement is possible do
 3:    for each document d do
 4:       find the cluster c whose centroid is most similar to d
 5:       assign d to this cluster c
 6:    end for
 7:    for each cluster c do
 8:       recompute the centroid of cluster c based on documents assigned to it
 9:    end for
10: end while
```

FIGURE 4.4 The *k*-means algorithm.

The goal of "soft" *k*-means is to find a μ_c for each c so as to minimize the *quantization error,* $\sum_d \min_c |d - \mu_c|^2$.

A simple strategy to iteratively reduce the error is to bring the mean vectors closer to the documents that they are closest to. We scan repeatedly through the documents, and for each document d, accumulate a "correction" $\Delta\mu_c$ for that μ_c that is closest to d:

$$\Delta\mu_c = \sum_d \begin{cases} \eta(d - \mu_c), & \text{if } \mu_c \text{ is closest to } d \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

After scanning once through all documents, all the μ_cs are updated in a batch by setting all $\mu_c \leftarrow \mu_c + \Delta\mu_c$. η is called the *learning rate*. It maintains some memory of the past and stabilizes the system. Note that each d moves only one μ_c in each batch.

The contribution from d need not be limited to only that μ_c that is closest to it. The contribution can be shared among many clusters, the portion for cluster c being directly related to the current similarity between μ_c and d. For example, we can soften (4.5) to

$$\Delta\mu_c = \eta \frac{1/|d - \mu_c|^2}{\sum_\gamma 1/|d - \mu_\gamma|^2}(d - \mu_c)$$

or

$$\Delta\mu_c = \eta \frac{\exp(-|d - \mu_c|^2)}{\sum_\gamma \exp(-|d - \mu_\gamma|^2)}(d - \mu_c) \tag{4.6}$$

Many other update rules, similar in spirit, are possible. Soft assignment does not break close ties to make documents contribute to a single cluster that wins narrowly. It is easy to show that some variants of soft k-means are special cases of the EM algorithm (see Section 4.4.2), which can be proved to converge to local optima.

Running time

In both variants of k-means, for each round, n documents have to be compared against k centroids, which will take $O(kn)$ time. The number of rounds is usually not too strongly dependent on n or k, and may be regarded as fixed.

Bottom-up clustering is often used to "seed" the k-means procedure. If k clusters are sought, the strategy is to randomly select $O(\sqrt{kn})$ documents from the collection of n documents and subject them to bottom-up clustering until there are k groups left. This will take $O(kn \log n)$ time. Once this step is over, the centroids of the k clusters, together with the remaining points, are used to seed a k-means procedure, which takes $O(kn)$ time. The total time over the two phases is thus $O(kn \log n)$.

4.3 Clustering and Visualization via Embeddings

Visualization of results from IR systems is a key driving force behind clustering algorithms. Of the two clustering techniques we have studied so far, HAC lends itself more readily to visualization, because trees and hierarchies are ubiquitous as user interfaces. Although k-means collects documents into clusters, it has no mechanism to represent the clusters visually, in a small number of dimensions.

In this section we will study a few clustering approaches that directly represent the documents as points in a given number of dimensions (two to three if direct visualization is desired). We start with *Kohonen* or *self-organizing maps* (SOMs), a close cousin of k-means. Next, we study multidimensional scaling (MDS), which gives an explicit optimization objective: we wish to minimize the error or distortion of interpoint distances in the low-dimensional embedding as compared to the dissimilarity given in the input data. This is a satisfying formulation, but usually intractable to exact optimization. The third category of techniques uses linear transformations to reduce the number of dimensions, and some of these approximately but provably preserve important properties related to interdocument similarity.

In all these cases, the ability to reduce the data to points in a 2D or 3D space that can be visualized directly is very valuable. The human eye is great at detecting clusters in low dimensions, and techniques that transform the data to such a format without losing important similarity information from the original data are very useful for analyzing text collections.

4.3.1 Self-Organizing Maps (SOMs)

Self-organizing, or Kohonen, maps are a close cousin to k-means, except that unlike k-means, which is concerned only with determining the association between clusters and documents, the SOM algorithm also embeds the clusters in a low-dimensional space right from the beginning and proceeds in a way that places related clusters close together in that space.

As in "soft" k-means, the SOM is built by associating a representative vector μ_c with each cluster c, and iteratively refining these representative vectors. Unlike k-means, each cluster is also represented as a point in a low-dimensional space. Clusters might be represented by nodes in a triangular or square grid, for example. Figure 4.5 shows a triangular grid. A large number of clusters can be initialized even if many regions are to remain devoid of documents in the end. In Figure 4.5, the background intensity shows the local density of documents assigned to each grid point. By extracting frequent words and phrases from the documents assigned to each cluster, we can "name" regions of the map as shown in Figure 4.5.

Based on the low-dimensional embedding, a *neighborhood* $N(c)$ is defined for each cluster c; for the square grid, $N(c)$ might be chosen as all nodes within two hops of c. We also design a *proximity* function $h(\gamma, c)$, which tells us how close a node γ is to the node c. $h(c, c) = 1$, and h decays with distance (e.g., the number of links on the shortest path connecting γ and c in the grid). In fact, we don't really need $N(c)$; we can simply let $h(\gamma, c)$ be 0 for $\gamma \notin N(c)$.

The update rule for an SOM will be generalized straight from Equation (4.5) by adding one new feature: if document d matches cluster c_d best, the update contribution from d should apply not only to c_d but to all $\gamma \in N(c_d)$ as well. SOMs are a kind of neural network where data item d "activates" the neuron c_d and some other closely neighboring neurons. The overall algorithm initializes all μ to random vectors and repeatedly picks a random document d from the collection and updates the model at each neuron until the model vectors stop changing significantly. The update rule for node γ under the influence of d is thus written as

$$\mu_\gamma \leftarrow \mu_\gamma + \eta h(\gamma, c_d)(d - \mu_\gamma) \tag{4.7}$$

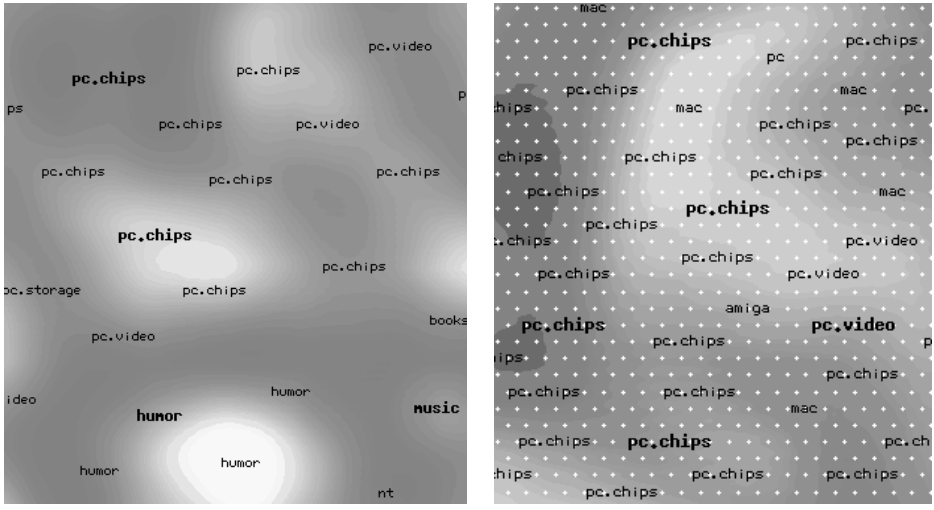
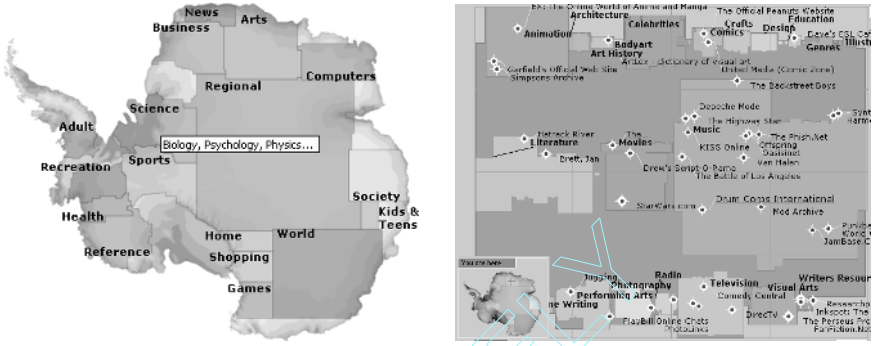


FIGURE 4.5 SOM computed from over a million documents taken from 80 Usenet news groups. Light areas have a high density of documents. The region shown is near groups `pc.chips` and `pc.video`, and closer inspection shows a number of URLs in this region that are about PC videocards.

Here, as before, η is a learning rate, which may be folded into h. An example of an SOM of over a million documents from 80 Usenet news groups is shown in Figure 4.5, together with the result of drilling down into the collection. Another example involving Web documents is shown in Figure 4.6, where the regions chalked out by SOM are in broad agreement with the human catalogers working on the Open Directory Project (*http://dmoz.org/*). The topic names in Figure 4.6 were generated manually once the correspondence with named DMoz.org topics was clear.

4.3.2 Multidimensional Scaling (MDS) and FastMap

In the case of k-means and SOM, documents have a specified internal representation, namely, the vector-space representation. In other applications, documents may be characterized only by a distance to other documents. Even in cases where an internal representation is available, one may use it for generating pairwise distances. Doing this may help in incorporating coarse-grained user feedback in clustering, such as "documents i and j are quite dissimilar" or "document i is more similar to j than k." These can be translated into a distance measure that



(a) (b)

FIGURE 4.6 Another example of SOM at work: the sites listed in the Open Directory Project have been organized within a map of Antarctica, at *antarcti.ca/* (a). Clicking on a region maintains context (inset) and zooms in on more specific topics (b). Documents are located at the cluster to which they are most similar.

overrides that computed from internal representations as the user provides more feedback.

The goal of MDS is to represent documents as points in a low-dimensional space (often 2D to 3D) such that the Euclidean distance between any pair of points is as close as possible to the distance between them specified by the input. Let $d_{i,j}$ be a (symmetric) user-defined measure of distance or dissimilarity between documents i and j, and let $\hat{d}_{i,j}$ be the Euclidean distance between the point representations of documents i and j picked by our MDS algorithm. The *stress* of the embedding is given by

$$\text{stress} = \frac{\sum_{i,j}(\hat{d}_{i,j} - d_{i,j})^2}{\sum_{i,j} d_{i,j}^2} \tag{4.8}$$

We would like to minimize the stress.

This formulation is very appealing but is not easy to optimize. Iterative stress relaxation, that is, hill climbing, is the most used strategy to minimize the stress. Here I shall talk about documents and points interchangeably. Initially, all points are assigned coordinates randomly or by some external heuristic. Then, each point in turn is moved by a small distance in a direction that locally reduces its stress.

With n points to start with, this procedure involves $O(n)$ distance computations for moving each point, and so $O(n^2)$ distance computations per relaxation

Team-Fly®

step. A much faster approach called FastMap, due to Faloutsos and Lin [76], pretends that the original documents are indeed points in some unknown high-dimensional space, and finds a projection to a space with a smaller number k of dimensions. The heart of the FastMap algorithm is to find a carefully selected line onto which the points are projected to obtain their first dimension, then project the points to a hyperplane perpendicular to the line, and recursively find the remaining $k-1$ coordinates. There are thus three key subproblems: (1) how to find a good direction or line, (2) how to "project" the original points onto the line (given that we have no internal representation of the documents), and (3) how to project the points to the hyperplane.

Because there is no internal representation available, the only way in which a direction or line can be specified is via a pair of points. Let a and b be two points defining the line, called the *pivots*. We can arbitrarily let a be the origin. Consider another point x for which we wish to compute the first coordinate x_1. Using the cosine law, we get

$$\begin{aligned} & d_{b,x}^2 = d_{a,x}^2 + d_{a,b}^2 - 2x_1 d_{a,b} \\ \Rightarrow \qquad & x_1 = \frac{d_{a,x}^2 + d_{a,b}^2 - d_{b,x}^2}{2\, d_{a,b}} \end{aligned} \tag{4.9}$$

This 1D projection does preserve some distance information: all points x close to a will have small x_1 and hence be close to each other in the projected space, and vice versa. This also gives a clue to finding a good line: informally, a line is good if projecting onto it helps spread out the points, that is, the point set has high variance in the direction of the line. This is difficult to ensure without exhaustive checking, so Faloutsos and Lin pick pivots that are far apart as a heuristic.

The next step is to project all points to the hyperplane perpendicular to the pivot line. Again, this "projection" cannot give us an internal representation of the original points, because we have not started with any. The only purpose of the "projection" is to correct interpoint distances by taking into account the component already accounted for by the first pivot line. Consider points x and y with distance $d_{x,y}$, first coordinates x_1 and y_1, and projections x', y' (with unknown internal properties) on the hyperplane. By the Pythagorean theorem, it is easy to see that the new distance d' on the hyperplane is

$$d'_{x',y'} = \sqrt{d_{x,y}^2 - (x_1 - y_1)^2} \tag{4.10}$$

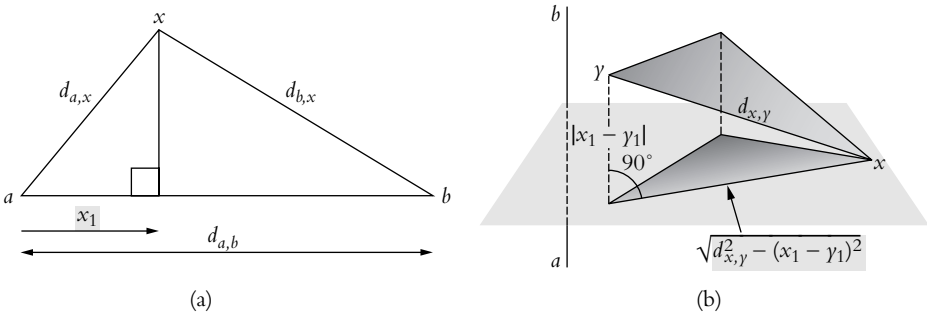


FIGURE 4.7 FastMap: projecting onto the pivot line (a), and projecting to a subspace with one less dimension (b).

At this point, we have derived the first dimension of all points and reduced the problem to one exactly like the original problem, except $k - 1$ additional dimensions remain to be computed. Therefore, we can simply call the routine recursively, until we go down to 1D space where the problem is trivially solved. The end product is a vector $(x_1, \ldots, x_k)$ for each point x in the original data set. It can be verified that FastMap runs in $O(nk)$ time. For visualization tasks, k is usually a small constant. Therefore, FastMap is effectively linear in the size of the point set. Figure 4.8 shows a fly-through 2D rendering of a 3D embedding of documents returned by a search engine in response to the query "tony bennett," which are clearly separated into two clusters. Closer inspection shows those clusters are about "country" and "jazz" music.

4.3.3 Projections and Subspaces

In many of the clustering algorithms we have discussed so far, including HAC and *k*-means style clustering, a significant fraction of the running time is spent in computing (dis-)similarities between documents and clusters. The time taken for one similarity calculation is proportional to the total number of nonzero components of the two vectors involved. One simple technique to speed up HAC or *k*-means is to *truncate* the document vectors to retain only some of the largest components. (We can retain either a fixed number of components or the smallest number of components that make up, say, at least 90% of the original vector's norm.) Truncation was introduced earlier, in Section 4.2.1; it has been

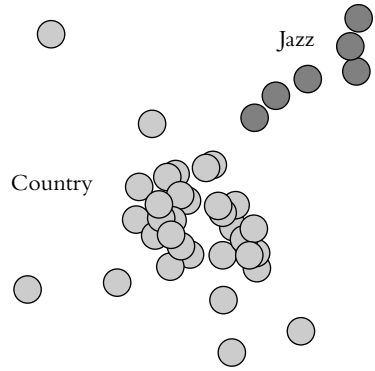


FIGURE 4.8 FastMap in action: clustering documents about country and jazz music.

experimentally evaluated by Schutze and Silverstein [191]. For the clustering task, it turns out that cutting down from tens of thousands to barely 50 dimensions has no significant negative impact on the quality of clusters generated by a clustering algorithm. Truncation to 50 axes per vector was comparable even to a more sophisticated global projection algorithm, discussed in Section 4.3.4.

One problem with orthogonal subspace projection is that one does not know if 50 or 100 coordinates are enough except by judging the outcome of clustering. Certain non-orthogonal projections have provable guarantees that the distortion that they force on inter-document distances are mild [168, 9]. Specifically, for any $0 < \epsilon > 1$ and any integer $n > 0$, choose any

$$k \geq \frac{4}{\epsilon^2/2 - \epsilon^3/3} \ln n.$$

Then for any set V of n vectors in $\mathbb{R}^d$, there is a map $f : \mathbb{R}^d \to \mathbb{R}^k$, computable in randomized polynomial time, such that for all pairs $\vec{x}, \vec{y} \in V$,

$$(1 - \epsilon)||\vec{x} - \vec{y}||^2 \leq ||f(\vec{x}) - f(\vec{y})||^2 \leq (1 + \epsilon)||\vec{x} - \vec{y}||^2.$$

While this represents a powerful theoretical property, the mapping f involves random rotations in the original space, which may destroy sparseness: document vectors which were very sparse in the original space may be mapped to dense vectors by f, reducing the performance gain from the apparently simpler

1: select, say, k^3 documents out of n uniformly at random
2: use HAC or move-to-nearest to cluster these to k^2 clusters
3: note the k^2 centroid vectors
4: for each document d, find the projection of $\vec{d}$ onto each of the centroid vectors
5: use this vector of k^2 real numbers as a representaton of d
6: with the new k^2-dimensional representation of all d, run a conventional clustering algorithm

FIGURE 4.9 Data-sensitive random projections.

representation. For example, with $\epsilon = 1/2$ and $n = 100000$, which could be quite typical in an application, we need $k \geq 32 \ln 100000 \approx 368$. If the average document has fewer than 368 terms, projection may not really simplify our document representation and therefore may not speed up clustering substantially. An effective heuristic to retain sparsity (at the cost of losing the theoretical distortion guarantee) is shown in Figure 4.9.

Hopefully, if the average document density is more than k^2, the second-round clustering will be much faster because of the speedup in distance computation. Note that this transformation is not linear. The intuition is that a uniform random selection will pick more documents from dense regions and few from unpopulated ones, with the result that fewer directions for projections will be needed to keep the clusters apart.

4.3.4 Latent Semantic Indexing (LSI)

Projections to orthogonal subspaces, that is, a subset of dimensions, may not reveal clustering structure in the best possible way. For example, the clusters may be formed by multiple correlated attributes. In this section I will characterize attribute redundancy more systematically in terms of linear algebraic operations on the term-document matrix.

Let the term-document matrix be A where the entry $A[t, d]$ may be a 0/1 value denoting the occurrence or otherwise of term t in document d. More commonly, documents are transformed into TFIDF vectors and each column of A is a document vector.

In the vector-space model, we allocated a distinct orthogonal direction for each token. The obvious intuition is that there is no need for so many (tens of thousands) of orthogonal directions because there are all sorts of latent relationships between the corresponding tokens. *Car* and *automobile* are likely to occur in similar documents, as are *cows* and *sheep*. Thus, documents as points in

this space are not likely to nearly "use up" all possible regions, but are likely to occupy semantically meaningful subspaces of it. Another way of saying this is that A has a much lower rank than $\min\{|D|, |T|\}$. (See the standard text by Golub and van Loan [91] for definitions of rank and matrix factoring and decomposition.)

One way to reveal the rank of A is to compute its *singular value decomposition* (SVD). Without going into the details of how the SVD is computed, which is standard, I will write down the decomposed form of A as

$$A_{|T|\times|D|} = U_{|T|\times r} \begin{pmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_r \end{pmatrix} V^T_{r\times|D|} \tag{4.11}$$

where r is the rank of A, U and V are column-orthonormal ($U^TU = V^TV = \mathbf{I}$, the identity matrix), and the diagonal matrix $\mathbf{\Sigma}$ in the middle can be organized (by modifying U and V) such that $\sigma_1 \geq \ldots \geq \sigma_r > 0$.

The standard cosine measure of similarity between documents can be applied to the A matrix: the entries of $(A^TA)_{|D|\times|D|}$ may be interpreted as the pairwise document similarities in vector space. The situation is completely symmetric with regard to terms, and we can regard the entries of $(AA^T)_{|T|\times|T|}$ as the pairwise term, similarity based on their co-occurrence in documents. (In Chapter 7, I will return to defining similarity using such matrix products, where the matrices will be node adjacency matrices of hyperlink graphs.)

The tth row of A may therefore be regarded as a $|D|$-dimensional representation of term t, just as the dth column of A is the $|T|$-dimensional vector-space representation of document d. Because A has redundancy revealed by the SVD operation, we can now use a "better" way to compute document-to-document similarities as $(V\mathbf{\Sigma}^2V^T)_{|D|\times|D|}$ and term-to-term similarities as $(U\mathbf{\Sigma}^2U^T)_{|T|\times|T|}$. In other words, the tth row of U is a refined representation of term t, and the dth row of V is a refined representation of document d. Interestingly, both representations are vectors in an r-dimensional subspace, and we can therefore talk about the similarity of a term with a document in this subspace.

In *latent semantic indexing* (*LSI*), the corpus is first used to precompute the matrices U, $\mathbf{\Sigma}$, and V. A query is regarded as a document. When a query "q" is submitted, it is first projected to the r-dimensional "LSI space" using the transformation

$$\hat{q} = \mathbf{\Sigma}^{-1}_{r\times r} U^T_{r\times|T|} q_{|T|} \tag{4.12}$$

At this point $\hat{q}$ becomes comparable with the r-dimensional document representations in LSI space. Now one can look for document vectors close to the transformed query vector.

In LSI implementations, not all r singular values are retained. A smaller number k, roughly 200 to 300, of the top singular values are retained—that is, A is approximated as

$$A_k = \sum_{1 \le i \le k} \vec{u}_i \sigma_i \vec{v}_i^T \tag{4.13}$$

where $\vec{u}_i$ and $\vec{v}_i$ are the ith columns of U and V. How good an approximation is A_k? The Frobenius norm of A is given by

$$|A|_F = \sqrt{\sum_{t,d} A[t,d]^2} \tag{4.14}$$

It can be shown that

$$|A|_F^2 = \sigma_1^2 + \cdots + \sigma_r^2, \tag{4.15}$$

and

$$\min_{\text{rank}(B)=k} |A - B|_F^2 = |A - A_k|_F^2 = \sigma_{k+1}^2 + \cdots + \sigma_r^2 \tag{4.16}$$

That is, A_k is the best rank-k approximation to A under the Frobenius norm.

The above results may explain why retrieval based on LSI may be close to vector-space quality, despite reduced space and perhaps query time requirements (although the preprocessing involved is quite time-consuming). Interestingly, in practice, LSI does *better,* in terms of recall/precision, than TFIDF retrieval. Heuristic explanations may be sought in signal-processing practice, where SVD has been used for decades, with the experience that the dominating singular values capture the "signal" in A, leaving the smaller singular values to account for the "noise." In IR terms, LSI maps synonymous and related words to similar vectors, potentially bridging the "syntax gap" in traditional IR and thus improving recall. Although a complete discussion is outside our scope here, LSI may also be able to exploit correlations between terms to resolve polysemy in some situations, improving precision as well.

More rigorous theories seeking to explain the improved accuracy of LSI have been proposed by Papadimitriou et al. [170] and by Azar et al. [9]. Papadimitriou et al. assume that documents are generated from a set of topics with disjoint

vocabularies, and after the resulting low-rank block matrix A is slightly perturbed, LSI can recover the block structure and hence the topic information. Azar et al. generalized this result to the case where A is not necessarily close to a block matrix but is approximated well by some low-rank matrix.

Thus far, we have discussed LSI/SVD as a device for dimensionality reduction, noise filtering, and ad hoc retrieval. But it can also be used for visualization (choose $k = 2$ or 3) or clustering, by using any of the other algorithms in this chapter after applying SVD. An example of a 2D embedding via LSI is shown in Figure 4.10. LSI can run in minutes to hours on corpora in the rough range of 10^3 to 10^4 documents, but is not very practical at the scale of the Web. At the time of this writing, I know of no public-domain SVD package that can work efficiently without storing the whole input matrix in memory. This can lead to an unacceptable memory footprint for a large collection.

4.4 Probabilistic Approaches to Clustering

Although the vector-space representation has been very successful for ad hoc retrieval, using it for clustering leaves a few unresolved issues. Consider HAC as discussed in Section 4.2.1. The document and group profile vectors were determined by a single IDF computation before the agglomerative process. Perhaps it makes more sense to compute IDF with regard to $\Gamma \cup \Delta$, not the entire corpus, when evaluating the self-similarity of $\Gamma \cup \Delta$. However, such a policy would interfere with the optimizations I have described.

Given a corpus with various salient topics, documents are likely to include terms highly indicative of one or relatively few topics, together with noise terms selected from a common set. A major function of IDF is to downplay noise-words, but we may get the same effect by identifying that a document is composed of these separate distributions, and attribute similarity only to overlap in terms generated from distributions other than the noise distribution. Continuing on this line of thought, documents assigned to some node c in a topic taxonomy such as Yahoo! may be thought of as picking up vocabulary from distributions associated with nodes on the path from the root up to c, inclusive. Note that the notion of a noise-word becomes context-dependent in the hierarchical setting: the word *can*, used largely as a verb, has low information content at the root node of Yahoo!, but in the subtree rooted at `/Environment/Recycling`, *can* is used mostly as a noun and should not be attributed to the noise distribution.

Label	Titles
B1	A Course on Integral Equations
B2	Attractors for Semigroups and Evolution Equations
B3	Automatic Differentiation of Algorithms: Theory, Implementation, and Application
B4	Geometrical Aspects of Partial Differential Equations
B5	Ideals, Varieties, and Algorithms –An Introduction to Computational Algebraic Geometry and Commutative Algebra
B6	Introduction to Hamiltonian Dynamical Systems and the *N*-Body Problem
B7	Knapsack Problems: Algorithms and Computer Implementations
B8	Methods of Solving Singular Systems of Ordinary Differential Equations
B9	Nonlinear Systems
B10	Ordinary Differential Equations
B11	Oscillation Theory for Neutral Differential Equations with Delay
B12	Oscillation Theory of Delay Differential Equations
B13	Pseudodifferential Operators and Nonlinear Partial Differential Equations
B14	Sync Methods for Quadrature and Differential Equations
B15	Stability of Stochastic Differential Equations with Respect to Semi-Martingales
B16	The Boundary Integral Approach to Static and Dynamic Contact Problems
B17	The Double Mellin-Barnes Type Integrals and Their Applications to Convolution Theory

(a)

FIGURE 4.10 Subtopics are clearly separated by LSI in this collection of mathematical abstracts (a). The query "application theory" and a cone around it is shown shaded on page 101 (b). (Image courtesy Berry et al. [15].)

In this section, we are interested in proposing and validating generative models of documents that move away from the paradigm of assigning importance to terms, or defining similarity or distance measures, by fiat. Instead, we propose random processes that generate documents, and characterize clustering as *discovering* the random processes and associated parameters that are (most) likely to have generated a given collection of documents. Several desirable ramifications will follow:

- There will be no need for IDF to determine the importance of a term.
- Some of the models we will study can directly and naturally capture the notion of stopwords vs. content-bearing words.
- There is no need to define distances or similarities between entities.
- Assignment of entities to clusters need not be "hard"; it is probabilistic.

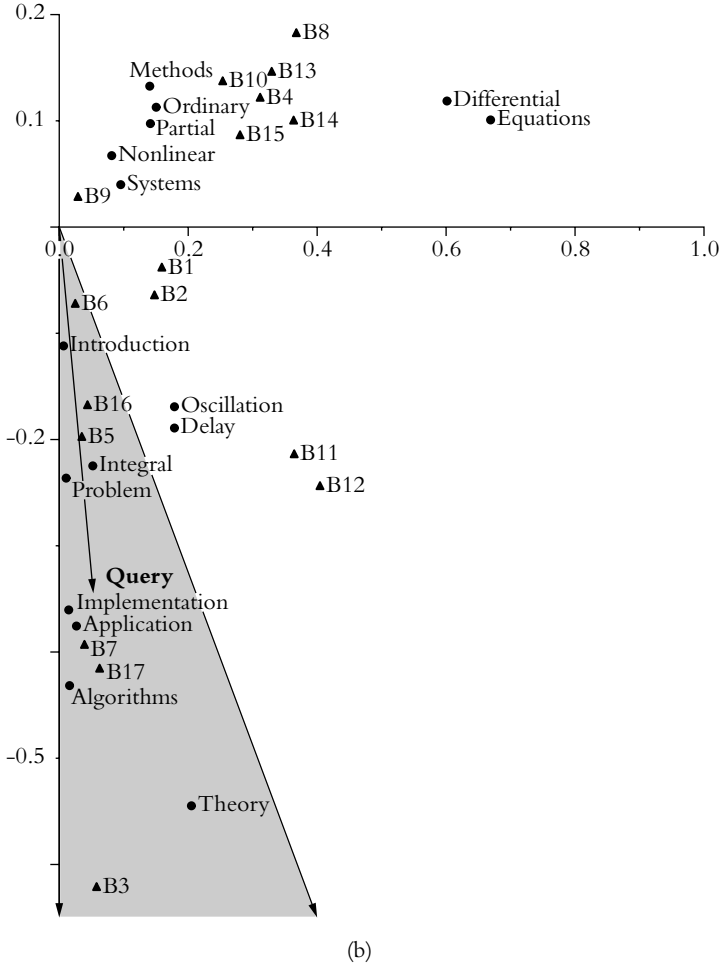


FIGURE 4.10 *(continued)*

4.4.1 Generative Distributions for Documents

Statistical pattern recognition and IR algorithms are built on the premise that the patterns (documents, images, audio) that we observe are generated by random processes that follow specific distributions. The observations let us estimate various

parameters pertaining to those distributions, which in turn let us design strategies for analyzing the patterns, by way of clustering, indexing, or classification.

This may sound clean and appealing, but proposing credible distributions that can generate natural language is very difficult. Even if some limited success can be achieved in this quest, the computation involved is usually heavy-duty. We must be content to model only a few aspects of the observed data, hoping that they will suffice for the application at hand.

The aspects that are almost always axed by the need for simplicity and efficiency are *dependencies* and *ordering* between terms. To appreciate why dependencies are difficult to capture, consider the Reuters collection [139], which has about 22,000 news articles, using over 30,000 unique tokens. Even if each attribute (axis) in the vector space had just two possible values (0/1), there would be $2^{30,000} \approx 10^{10,000}$ possible documents. Clustering is intimately tied to estimating the density of the distribution being sampled, so the chances of finding a decent estimate in a space this size with only 22,000 documents is out of the question.

In a bid to reduce the severity of the problem, we can make the drastic assumption that term occurrences are *independent* events. To start with, let us make the further assumption that term counts are unimportant, that is, the event associated with a term and a document is a 0/1 random variable. This is called the *multivariate binary* model, or the *binary model* for short. A document *event* is just a bit-vector with a 0/1 slot for each term in the vocabulary W, and the bit corresponding to a term t is flipped on with probability ϕ_t, and off with probability $1 - \phi_t$. All the ϕ_ts are collected into the parameter set for this model, called Φ. Given Φ, the probability of generating document d is given by

$$\Pr(d|\Phi) = \prod_{t \in d} \phi_{c,t} \prod_{t \in W, t \notin d} (1 - \phi_{c,t}) \tag{4.17}$$

Since typically $|W| \gg |d|$, short documents are discouraged by this model. Also, the second product makes strong independence assumptions and is likely to greatly underestimate $\Pr(d|\Phi)$ for many not-so-unlikely documents. On the other hand, assuming all $\phi_t > 0$ and $\phi_t < 1$, all the $2^{|W|}$ possible documents, some of them essentially impossible in real life, have positive probability. Thus, this model smooths out the probability over too large a space, depriving the more likely regions.

In our second attempt, we will model term counts. The modified generative process is as follows: the writer first decides the total term count (including

Team-Fly®

repetitions) of the document d to be generated by drawing a random positive integer L from a suitable distribution $\Pr(\ell)$; suppose the instantiated event is ℓ_d. Next, the writer gets a die: it has $|W|$ faces, one face for each term in the vocabulary. When tossed, the face corresponding to term t comes up with probability θ_t ($\sum_t \theta_t = 1$). We represent by Θ all parameters needed to capture the length distribution and all θ_ts. The author tosses this die ℓ_d times and writes down the terms that come up. Suppose term t turns up $n(d,t)$ times, with $\sum_\tau n(d,\tau) = \ell_d$. The document *event* in this case comprises ℓ_d and the set of counts $\{n(d,t)\}$. The probability of this compound event is given by

$$\begin{aligned}\Pr(\ell_d, \{n(d,t)\}|\Theta) &= \Pr(L = \ell_d|\Theta)\ \Pr(\{n(d,t)\}|\ell_d, \Theta) \\ &= \Pr(L = \ell_d|\Theta)\binom{\ell_d}{\{n(d,t)\}}\prod_{t\in d}\theta_t^{n(d,t)} \qquad (4.18)\end{aligned}$$

where $\binom{\ell_d}{\{n(d,t)\}} = \frac{\ell_d!}{n(d,t_1)!\,n(d,t_2)!\cdots}$ is the multinomial coefficient. We will abbreviate the compound event on the lhs by $\Pr(d|\Theta)$. This is called the *multinomial* model. The length distribution is vital; without it, the empty document would have probability 1.

The multinomial model does not fix the term-independence assumption. In fact, it assumes that occurrences of a given term are also independent of each other, which is another assumption that is clearly wrong. Reading a document from left to right, if you see the word *Pentium* five times, you are not really surprised to see it a sixth time, unlike what the additional factor of θ_t in Equation (4.18) suggests. Even so, the multinomial model, in preserving term count information, turns out to be somewhat superior for most text mining tasks.

From a linguist's perspective, such models are insufferably crude: there is not a shade of grammar or semantic sense in these characterizations; there is not even a hint of the strong short-range dependence that is commonly seen between terms. (For example, the word *spite* is quite likely to follow the word *in* and precede the word *of*.) We offer no defense, but note that these models are approximations to physical reality, make parameter estimation tractable, and produce acceptable experimental results for machine learning tasks in the text domain.

4.4.2 Mixture Models and Expectation Maximization (EM)

The notion of generative distributions makes it easy and elegant to express the clustering problem, perhaps a little more elegantly than the formulations in Section 4.1. Consider a given collection of documents, for example, a set of

pages crawled from the Web. It is possible to estimate Θ_{Web} for this collection, and then calculate the probability $\Pr(d|\Theta_{\text{Web}})$ of all Web documents d with regard to Θ_{Web}. But we may not really believe that a single multinomial model suffices for the whole Web. Suppose a set of topics, such as arts, science, and politics, were given to us ahead of time, and we can identify which topic a document talks about. (Until Section 4.4.3 we will assume that a document is about exactly one topic.) We can estimate, in lieu of a single Θ_{Web}, specialized parameter sets $\Theta_{\text{arts}}, \Theta_{\text{science}}, \Theta_{\text{politics}}$, and so on, and for a document belonging to a topic γ, evaluate $\Pr(d|\Theta_{\gamma})$. Intuitively, we would expect this to be generally much larger than $\Pr(d|\Theta_{\text{Web}})$, because Θ_{γ} may correctly capture that some terms are rare or frequent in documents about topic γ, compared to a random document from the Web at large.

The preceding discussion reveals the essence of the clustering problem and leads us to the following *mixture model* for document generation. Suppose there are m topics (also called *components* or *clusters*). The author of a page has to first decide what topic he wishes to write about. This may be done using a multinomial m-way *selector* distribution with probabilities $\alpha_1, \ldots, \alpha_m$, where $\alpha_1 + \cdots + \alpha_m = 1$. Once a topic γ is decided upon, the author uses Θ_{γ}, the distribution for that topic, to generate the document. For example, we can use the binary or the multinomial distribution for each component. We can even use different distributions for different components.

In preparation for the rest of this section, I will simplify the notation. For each component γ, there are many parameters $\theta_{\gamma,t}$, one for each term t. I collect all these parameters, all the α_is, as well as the number of clusters m, into a global parameter space. I reuse Θ to name this parameter space:

$$\Theta = (m; \alpha_1, \ldots, \alpha_m; \{\theta_{\gamma,t}\ \forall \gamma, t\})$$

It is conventional to denote the data points as x rather than d, which I will follow for the rest of this section. Lastly, I will illustrate the EM algorithm not with the multinomial component distribution but with a simple distribution characterized by just one parameter per component: the Poisson distribution with mean μ characterized by $\Pr(X = x) = e^{-\mu}\mu^x/x!$, for $x = 0, 1, 2, \ldots$. Accordingly, we will have m parameters $\mu_1, \ldots, \mu_m$, and our simplified parameter space will be

$$\Theta = (m;\ \alpha_1, \ldots, \alpha_m;\ \mu_1, \ldots, \mu_m)$$

With the setup as described so far, we see that

$$\Pr(x|\Theta) = \sum_{j=1}^{m} \alpha_j \Pr(x|\mu_j) \tag{4.19}$$

Note that x is multivariate in general—certainly for text—although for the Poisson distribution, it is just a real number.

For the clustering task, we are given n independent, identically distributed (iid.) observations $X = \{x_1, \ldots, x_n\}$, and we need to estimate Θ—that is, we would like to find Θ so as to maximize

$$\Pr(X|\Theta) = \prod_{i=1}^{n} \Pr(x_i|\Theta) \quad \triangleq \quad L(\Theta|X) \tag{4.20}$$

and thus

$$\log L(\Theta|X) = \sum_{i} \log \left(\sum_{j} \alpha_j \Pr(x_i|\mu_j) \right) \tag{4.21}$$

For the moment we will assume that m is provided as an input. Our estimation of Θ will therefore concern the α and μ parameters.

If the component y_i ($1 \leq y_i \leq m$) from which each observation x_i has been generated were known, this would be a trivial problem. The challenge is that $Y = \{y_i\}$ is a set of *hidden* random variables. The component distributions define the clusters, and Y indicates the cluster to which each data point belongs.

Since Y is unknown, it must be modeled as a random variable, and we can assign data points to clusters only in a probabilistic sense. The classic approach to solving the problem is to maximize L (see Equation (4.21)) explicitly with regard to both X and Y: $L(\Theta|X, Y) = \Pr(X, Y|\Theta)$. Since we do not know Y, we must take the expectation of L over Y. Unfortunately, estimating the distribution of Y requires knowledge of Θ. To break the cycle, we start with a suitable guess Θ^g. Let the "complete data likelihood" be

$$Q(\Theta, \Theta^g) = E_Y \left(\log L(\Theta|X, Y) \middle| X, \Theta^g \right) \tag{4.22}$$

$$= \sum_Y \{\Pr(Y|X, \Theta^g)\}\{\log \Pr(X, Y|\Theta)\} \tag{4.23}$$

$$= \sum_Y \{\Pr(Y|X, \Theta^g)\}\{\log(\Pr(Y|\Theta) \Pr(X|Y, \Theta))\} \tag{4.24}$$

$$= \sum_{\gamma_1=1}^{m} \cdots \sum_{\gamma_n=1}^{m} \left\{ \prod_{j=1}^{n} \Pr(\gamma_j|x_j, \Theta^g) \right\} \left\{ \sum_{i=1}^{n} \log(\alpha_{\gamma_i} \Pr(x_i|\mu_{\gamma_i})) \right\} \tag{4.25}$$

The last expression can be simplified to

$$Q(\Theta, \Theta^g) = \sum_{\ell=1}^{m} \sum_{i=1}^{n} \Pr(\ell|x_i, \Theta^g) \ \log \left(\alpha_\ell \Pr(x_i|\mu_\ell) \right) \tag{4.26}$$

Because Q is an expectation over Y, this step is called the *expectation,* or E, step.

How should we pick a refined value of Θ? It seems reasonable to choose the next estimate of Θ so as to maximize $Q(\Theta, \Theta^g)$. This step is called the *maximization,* or M, step. There is one constraint to the maximization, namely, $\sum_i \alpha_i = 1$, and we perform a standard Lagrangian optimization:

$$\frac{\partial}{\partial \alpha_k} \left[\sum_{\ell=1}^{m} \sum_{i=1}^{n} \{\log \alpha_i + \cdots\} \Pr(\ell|x_i, \Theta_g) - \lambda \sum_i \alpha_i \right] = 0 \tag{4.27}$$

which yields

$$\alpha_k = \frac{1}{\lambda} \sum_{i=1}^{n} \Pr(k|x_i, \Theta^g) \tag{4.28}$$

From the constraint we can now show that $\lambda = n$.

We must also find the new values of μ_i, $i = 1, \ldots, m$. For concreteness I have picked a specific one-parameter distribution, a Poisson distribution with mean μ_i for the ith component. (It is not necessary for all components to follow the same distribution for the algorithm to work.) The Poisson distribution is characterized as $\Pr(x|\mu) = e^{-\mu}\mu^x/x!$ for integer $x = 0, 1, 2, \ldots$. Thus, our second set of derivatives is

$$\frac{\partial}{\partial \mu_k} \left[\sum_{\ell=1}^{m} \sum_{i=1}^{n} \Pr(\ell|x_i, \Theta^g) \left(-\mu_\ell + x_i \log \mu_\ell \right) \right] = 0 \tag{4.29}$$

which yields

$$\sum_{i=1}^{n} \left(-1 + \frac{x_i}{\mu_k} \right) \Pr(k|x_i, \Theta^g) = 0 \tag{4.30}$$

Simplifying,

$$\mu_k = \frac{\sum_{i=1}^{n} x_i \Pr(k|x_i, \Theta^g)}{\sum_{i=1}^{n} \Pr(k|x_i, \Theta^g)} \tag{4.31}$$

The complete algorithm, called *expectation maximization* (EM), is shown in Figure 4.11. It can be shown that the maximization step guarantees that $L(\Theta)$ never decreases, and must therefore reach a local maximum.

In general, finding a suitable value of m is a nontrivial task. For some applications, m may be known, and in addition, for some documents i, y_i (the cluster to which that document belongs) may also be specified. A common example would be the assignment of Web documents to Yahoo!-like clusters, where a few documents have been manually assigned to clusters but most documents are not assigned. This is an instance of a *semisupervised* learning problem, which we will study in Chapter 6. Completely supervised learning or classification is the topic of Chapter 5. There the classifier is given a fixed set of labels or classes and sample documents with each class.

When m is not specified, there are two broad techniques to estimate it. The first is to hold out some of the data, build the mixture model on the rest, then find the likelihood of the held-out data given the mixture parameters. This process is repeated while increasing the number of clusters until the likelihood ceases to increase. (Note that this would not work without the held-out data; if training data were used, the system would prefer an inordinately large value of m, a phenomenon called *overfitting*, discussed in Section 5.5.) This approach has been proposed by Smyth [197].

1: Initialize $\Theta^{(0)}$, $i = 0$
2: **while** $L(\Theta|X, Y)$ can be increased **do**
3: Estimate $\vec{\alpha}^{(i+1)}$ using (4.28)
4: Estimate $\vec{\mu}^{(i+1)}$ using (4.31)
5: $i \leftarrow i + 1$
6: **end while**

FIGURE 4.11 The EM algorithm.

A different approach is to constrain the model complexity using a *prior distribution* over the model parameters that makes complex models unlikely (see Sections 4.4.5 and 5.6.1 for more details on prior distributions). This is the approach adopted in the well-known AutoClass clustering package by Cheeseman and others [47].

A criticism of the standard mixture model as applied to text is that many documents are relevant to multiple topics. In fact, the term *mixture model* may be misleading in this context, because after a generating distribution is selected probabilistically, a data point is generated from only one distribution after all. Operationally, this means that each distribution has to "compete on its own" with other distributions for a share of α, that is, they cannot collude to generate documents. In the next two sections, I will discuss two approaches to address this limitation.

4.4.3 Multiple Cause Mixture Model (MCMM)

If a document is (partially or wholly) about a topic, the topic *causes* certain words to become more likely to appear in the document. Let c be the topics or clusters and t be terms. Let $\gamma_{c,t}$ $(0 \leq \gamma_{c,t} \leq 1)$ denote a normalized measure (not to be interpreted as a probability) of causation of t by c. Suppose the extent to which topic c is "activated" in writing a given document d is $a_{d,c}$ $(0 \leq a_{d,c} \leq 1)$. Then the belief that term t will appear in the document d is given by a *soft disjunction*, also called a *noisy OR*:

$$b_{d,t} = 1 - \prod_{c}(1 - a_{d,c}\,\gamma_{c,t}), \tag{4.32}$$

That is, the term does not appear only if it is not activated by any of the classes under consideration. Let the document d be represented by the binary model where $n(d, t)$, the number of times term t appears in it, is either zero or one. Then the goodness of the beliefs in various term activations is defined as a log likelihood:

$$\begin{aligned} g(d) &= \log\left(\prod_{t\in d} b_{d,t} \prod_{t\notin d}(1 - b_{d,t})\right) \\ &= \sum_{t} \log\left(n(d,t)\, b_{d,t} + (1 - n(d,t))(1 - b_{d,t})\right) \end{aligned} \tag{4.33}$$

For a document collection $\{d\}$ the aggregate goodness is $\sum_d g(d)$.

Like other iterative clustering algorithms, we somehow set a number of clusters, and the iterations proceed in pairs of half-steps. In each iteration, the first half-step fixes $\gamma_{c,t}$ and improves on the choice of $a_{d,c}$. The second half-step fixes $a_{d,t}$ and improves on the choice of $\gamma_{c,t}$. In both half-steps, the search for improved parameter values is done by local hill climbing, that is, finding $\partial \sum_d g(d)/\partial a_{d,c}$ or $\partial \sum_d g(d)/\partial \gamma_{c,t}$ and taking a short step along the gradient.

MCMMs can be used in a supervised learning setting, too (see Chapter 5); in that case, the activations $a_{d,c}$ are provided for documents d in the training set, and the system needs to estimate only the coupling matrix $\gamma_{c,t}$. When given a new document q, the coupling matrix is kept fixed and $a_{q,c}$ estimated so as to maximize $g(q)$. This information can be used to tag documents with labels from a predefined set of labels with examples that have been used to train or supervise the system.

MCMMs are thus a very flexible and simple model, useful for both unsupervised and supervised learning. Their only drawback is speed. The representation of the coupling matrix is dense, and hill climbing is slow. With a few hundred terms, a thousand documents, and about 10 clusters, supervised runs take a few minutes and unsupervised runs take a few hours on stock hardware. For larger document collections with tens of thousands of terms, aggressive elimination of terms (see Section 5.5) is required.

4.4.4 Aspect Models and Probabilistic LSI

Hofmann has proposed a new generative model for multitopic documents [109, 110]. We start with the raw term counts in a given document collection, in the form of a matrix in which entry $n(d, t)$ denotes the frequency of term t in document d. Put another way, each pair (d, t) has a *binary event* associated with it. The *number of times* this event occurs is the observed data $n(d, t)$. Note the subtle distinction between this model and the multinomial model discussed in Section 4.4.1. In the multinomial model, given a document length, the frequencies of individual terms apportion this quota of total count. Here the total event count over all (d, t) is set in advance, and the (d, t) events must apportion this total count. This means that the corpus must be fixed in advance and that analyzing a new document from outside the corpus takes some special steps, unlike the multinomial model.

When an author starts composing a document, she induces a probability distribution $\Pr(c)$ over topics or clusters. For example, she may set a probability of 0.3 for writing (using terms) about politics and 0.7 for petroleum. Different

clusters *cause* event (d, t) with different probabilities. To find the overall $\Pr(d, t)$, we condition and sum over clusters:

$$\Pr(d, t) = \sum_c \Pr(c) \Pr(d, t|c) \tag{4.34}$$

The main approximation in the aspect model is to assume conditional independence between d and t given c, which gives us

$$\Pr(d, t) = \sum_c \Pr(c) \Pr(d|c) \Pr(t|c) \tag{4.35}$$

The important parameters of this characterization are $\Pr(c)$, $\Pr(d|c)$, and $\Pr(t|c)$. An EM-like procedure can be used to estimate these parameters, together with the E-step parameter $\Pr(c|d, t)$, which may be interpreted as a grade of evidence that event (d, t) was caused by cluster c.

$$\begin{aligned}\Pr(c|d, t) &= \frac{\Pr(c, d, t)}{\Pr(d, t)} \\ &= \frac{\Pr(c) \Pr(d, t|c)}{\sum_\gamma \Pr(\gamma, d, t)} \\ &= \frac{\Pr(c) \Pr(d|c) \Pr(t|c)}{\sum_\gamma \Pr(\gamma) \Pr(d|\gamma) \Pr(t|\gamma)} \qquad (4.36)\end{aligned}$$

$$\Pr(c) = \frac{\sum_{d,t} n(d, t) \Pr(c|d, t)}{\sum_\gamma \sum_{d,t} n(d, t) \Pr(\gamma|d, t)} \tag{4.37}$$

$$\Pr(d|c) = \frac{\sum_t n(d, t) \Pr(c|d, t)}{\sum_\delta \sum_t n(\delta, t) \Pr(c|\delta, t)} \tag{4.38}$$

$$\Pr(t|c) = \frac{\sum_d n(d, t) \Pr(c|d, t)}{\sum_\tau \sum_d n(d, \tau) \Pr(c|d, \tau)} \tag{4.39}$$

As in EM, the user has to fix the number of clusters ahead of time or use validation with a held-out set. Hofmann also describes an enhanced EM procedure. The number of clusters is akin to the number of singular values retained in the LSI (SVD) decomposition discussed in Section 4.3.4; we may use held-out data for cross-validation to determine a suitable number of clusters.

The factor model can be used as a probabilistic version of LSI, dubbed *probabilistic LSI,* or *PLSI*. A text collection is first subjected to the PLSI analysis

and the four sets of parameters estimated as specified. Now for each document d and each cluster c, we precompute

$$\Pr(c|d) = \frac{\Pr(c)\,\Pr(d|c)}{\sum_\gamma \Pr(\gamma)\,\Pr(d|\gamma)} \tag{4.40}$$

where all the quantities on the right-hand side are estimated parameters. A query q (regarded as a bag of words, like documents) has to be *folded* into the system. The precalculated parameters are frozen, and new parameters $\Pr(c|q, t)$ for all c, t, $\Pr(q|c)$ for all c are estimated as

$$\Pr(c|q, t) = \frac{\Pr(c)\underline{\Pr(q|c)}\,\Pr(t|c)}{\sum_\gamma \Pr(\gamma)\underline{\Pr(q|\gamma)}\,\Pr(t|\gamma)} \tag{4.41}$$

$$\Pr(q|c) = \frac{\sum_t n(q, t)\underline{\Pr(c|q, t)}}{\sum_t n(q, t)\underline{\Pr(c|q, t)} + \sum_d \sum_t n(d, t)\,\Pr(c|d, t)} \tag{4.42}$$

This is itself an iterative procedure with the coupling shown by the underlined variables.

Once $\Pr(c|q)$ and $\Pr(c|d)$ are known for all d, one may use the vector of posterior class probabilities as a surrogate representation, just as the projection via U or V^T is used in LSI. That is, the similarity between q and d may be defined in a number of reasonable ways, for example, $\sum_c \Pr(c|q)\,\Pr(c|d)$, or $\sum_c \Pr(c)\,\Pr(d|c)\,\Pr(q|c)$, for both of which the similarity-finding operations remains a dot-product of vectors.

PLSI has been evaluated using four standard IR collections: MED (1033 abstracts from medical journals), CRAN (1400 documents on aeronautics), CACM (3204 abstracts from a computer science periodical), and CISI (1460 abstracts related to library science). As shown in Figure 4.12, PLSI compares favorably in terms of recall precision with standard TFIDF cosine-based ranking. The vector-space ranking used in comparison is a simple one-shot process. The best vector-space–based contenders today use two enhancements. First, it is a two-shot process: some number of top-ranking results are assumed to be relevant, and a second query is generated including certain words from those top-ranking documents; the final response set is the result of this second query. Second, the TFIDF weights are adjusted to reflect diverse document lengths; this may lead to favorable scoring of documents that match a query in only a few local regions.

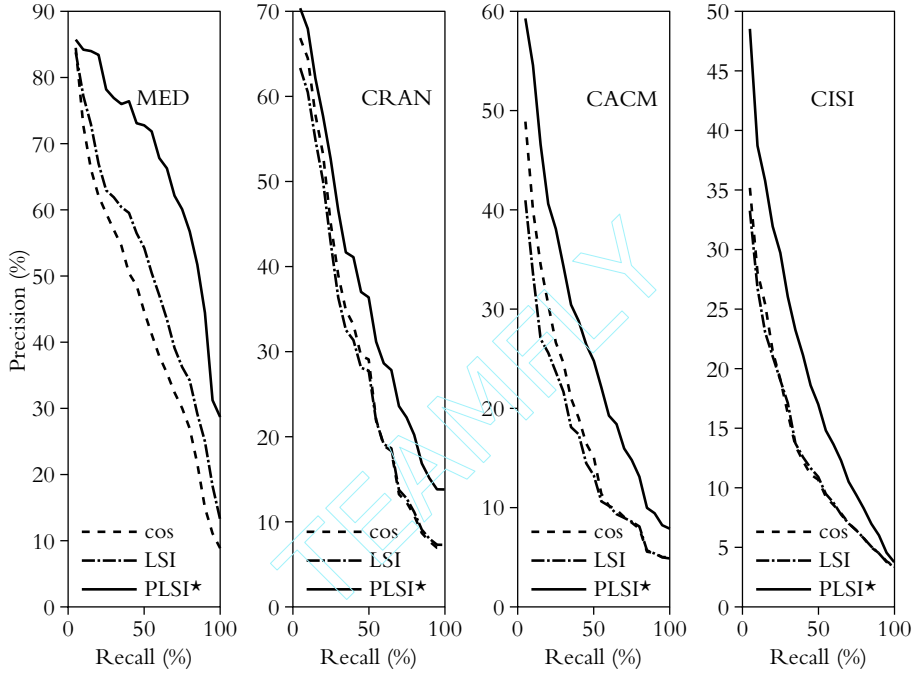


FIGURE 4.12 PLSI shows a significant improvement beyond standard one-shot TFIDF vector-space retrieval as well as standard LSI for several well-known data sets.

It would be interesting to compare PLSI with the length-adjusted, two-round variants of TFIDF search engines.

4.4.5 Model and Feature Selection

Clustering is also called *unsupervised learning* because topic-based clusters emerge as a result of the learning process, and are not specified ahead of time. As we have seen, (dis-)similarity measures are central to many forms of unsupervised learning. With a large number of dimensions where many dimensions are noisy and correlated, the similarity measure gets distorted rather easily. For example, noise-words or stopwords are integral to any language. A precompiled list of stopwords, or even a corpus-dependent IDF weighting, may fail to capture semantic emptiness in certain terms. Failing to eliminate or play down these

Team-Fly®

dimensions sufficiently results in all similarity scores being inflated by some random, noisy amount. In bottom-up clustering, this noise often manifests itself in unbalanced, stringy dendrograms—once a cluster becomes large, there is no stopping it from gathering more mass.

A possible approach is to launch a search for a subset of terms that appears to be "noisy" in the sense that the clusters found all share a common distribution over these terms, together with per-cluster distribution over useful "signal" terms. Let D be the set of documents and D^T, D^N, and D^S be the representation of documents in the entire term space T, a noisy subset of term N, and the complement signal space $S = T \setminus N$. Using standard independence assumptions, we can approximate

$$\begin{aligned} \Pr(D^T|\Theta) &= \Pr(D^N|\Theta)\Pr(D^S|\Theta) && (4.43)\\ &= \prod_{d\in D} \Pr(d^N|\Theta^N)\Pr(d^S|\Theta^S) \\ &= \prod_{d\in D} \Pr(d^N|\Theta^N)\Pr(d^S|\Theta^S_{c(d)}) && (4.44) \end{aligned}$$

where Θ^S is composed of per-cluster parameters Θ^S_c for a set of clusters $\{c\}$, $c(d)$ is the cluster to which d belongs, and d^N (respectively, d^S) are documents projected to the noise (respectively, signal) attributes.

If $|T|$ ranges into tens of thousands, it can be daunting to partition it every way into N and S. An obvious technique is to cluster the terms in T according to their occurrence in documents, the process being called *distributional clustering*. (One can use the U matrix in LSI for doing this, for instance.) The hope is to collect the term set into a manageable number of groups, each of which is then tested for membership or otherwise in S as a whole.

MCMM and PLSI may achieve the same effect via a slightly different route. In MCMM, we can designate one cluster node to take care of the noise terms, and we can do likewise with one factor in PLSI. We can seed these clusters suitably (with, say, known stopwords) so that they gravitate toward becoming a generator of noise terms. It would be interesting to compare how well MCMM and PLSI achieve signal and noise separation compared to the feature subset search approach.

There is another useful way to look at the search for S and N in Equation (4.43): we would like to maximize $\Pr(D^T|\Theta)$, while at the same time *share* the cost of the parameters for the N subspace over all clusters—there is only one set

of parameters Θ^N, whereas the Θ^S is diversified over each cluster. In other words, we wish to *factor out* Θ^N from all the clusters. Why is this desirable?

The medieval English philosopher and Franciscan monk William of Ockham (c. 1285–1349) proposed that "plurality should not be posited without necessity" ("*pluralitas non est ponenda sine neccesitate*"). This utterance has since been called *Occam's razor*, *the principle of parsimony*, and *the principle of simplicity*, and it has had profound influence on statistical analysis of noisy data.

In data analysis, Occam's razor would favor the simplest model that "explains" the data as well as any other. More formally, if under some assumptions about the space of generative models, two models generate the data with equal probability, then we should prefer the simpler model. This is not merely a normative stand. As we shall see in Chapter 5, picking simple models helps us generalize what we have learned from limited samples to yet-unseen data. If we do not control the complexity of the models we accept, we are in danger of learning chance artifacts from our sample data, a phenomenon called *overfitting*.

The *Minimum Description Length* (MDL) principle [182] is a ramification of Occam's razor that helps us control model complexity. MDL expresses the goodness of fit of models to data by composing a cost measure that has two components: *model cost* and *data cost*. The model cost is the number of bits $L(\Theta)$ needed to express an efficient encoding of the model Θ. The data cost $L(\mathbf{x}|\Theta)$ is the number of bits needed to express the data $\mathbf{x}$ with regard to a specified model (not necessarily a mixture model). Shannon's classic work on information and coding theory [57] lets us approximate $L(\mathbf{x}|\Theta) \approx -\log \Pr(\mathbf{x}|\Theta)$, the entropy lower bound, in most cases. Clustering thus amounts to finding

$$\begin{aligned}\Theta^* &= \arg\min_{\Theta} \{L(\Theta) + L(\mathbf{x}|\Theta)\} \\ &= \arg_{\Theta} \min \left\{L(\Theta) - \log \Pr(\mathbf{x}|\Theta)\right\}\end{aligned} \tag{4.45}$$

$L(\Theta)$ is the coding cost for the model and its parameters. The coding cost of parameters that take values from finite, discrete sets is easily determined by assuming a prior distribution over the parameters. For example, we may assume a prior distribution for m, the number of components in a mixture model (see Section 4.4.2) of the form $\Pr(M = m) = 2^{-m}$ for $m \geq 1$. Now we can use Shannon's theorem again to encode the parameter with regard to the prior distribution with a cost close to the entropy of the prior distribution. For continuous-valued parameters, some form of discretization is needed. A complete description of continuous parameter spaces is beyond this book's scope.