14 *Clustering*

CLUSTERING ALGORITHMS partition a set of objects into groups or *clusters*. Figure 14.1 gives an example of a clustering of 22 high-frequency words from the Brown corpus. The figure is an example of a *dendrogram*, a branching diagram where the apparent similarity between nodes at the bottom is shown by the height of the connection which joins them. Each node in the tree represents a cluster that was created by merging two child nodes. For example, *in* and *on* form a cluster and so do *with* and *for*. These two subclusters are then merged into one cluster with four objects. The "height" of the node corresponds to the decreasing similarity of the two clusters that are being merged (or, equivalently, to the order in which the merges were executed). The greatest similarity between any two clusters is the similarity between *in* and *on* – corresponding to the lowest horizontal line in the figure. The least similarity is between *be* and the cluster with the 21 other words – corresponding to the highest horizontal line in the figure.

CLUSTERS

DENDROGRAM

While the objects in the clustering are all distinct as tokens, normally objects are described and clustered using a set of features and values (often known as the *data representation model*), and multiple objects may have the same representation in this model, so we will define our clustering algorithms to work over *bags* – objects like sets except that they allow multiple identical items. The goal is to place similar objects in the same group and to assign dissimilar objects to different groups.

DATA REPRESENTATION MODEL

BAGS

What is the notion of 'similarity' between words being used here? First, the left and right neighbors of tokens of each word in the Brown corpus were tallied. These distributions give a fairly true implementation of Firth's idea that one can categorize a word by the words that occur around it. But now, rather than looking for distinctive collocations, as in

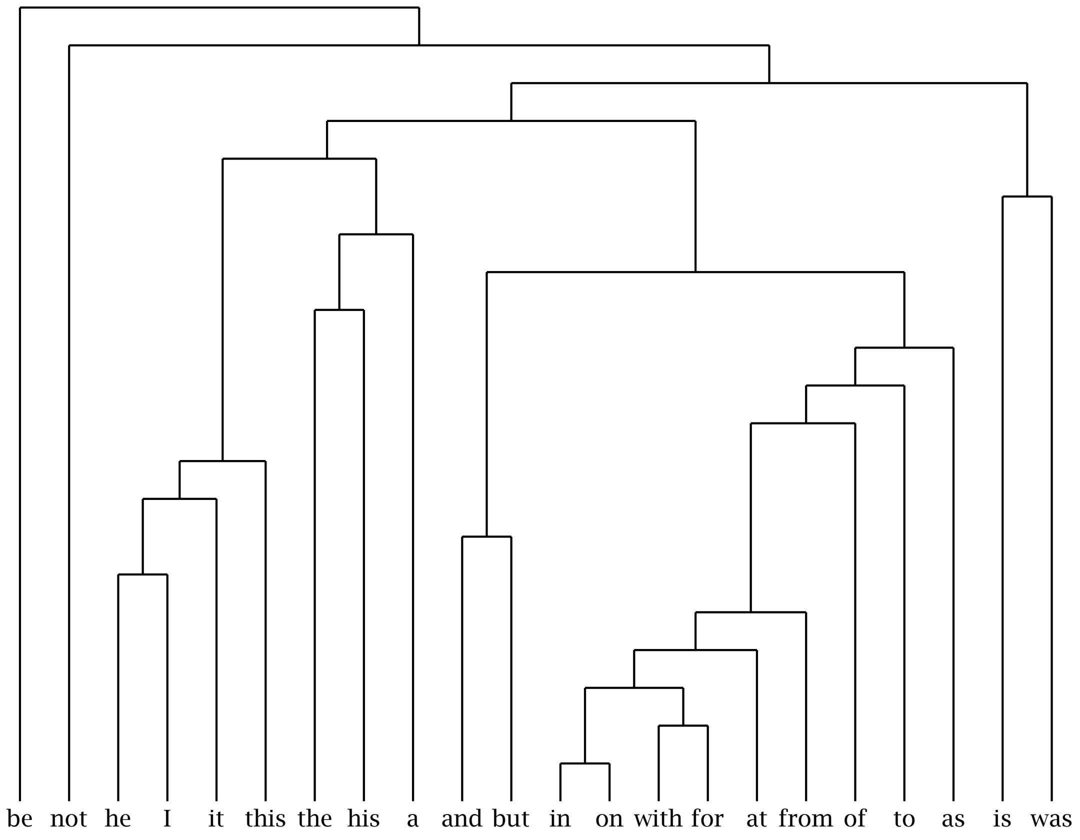

Figure 14.1 A single-link clustering of 22 frequent English words represented as a dendrogram.

chapter 5, we are capturing and using the whole distributional pattern of the word. Word similarity was then measured as the degree of overlap in the distributions of these neighbors for the two words in question. For example, the similarity between *in* and *on* is large because both words occur with similar left and right neighbors (both are prepositions and tend to be followed by articles or other words that begin noun phrases, for instance). The similarity between *is* and *he* is small because they share fewer immediate neighbors due to their different grammatical functions. Initially, each word formed its own cluster, and then at each step in the clustering, the two clusters that are closest to each other are merged into a new cluster.

EXPLORATORY DATA ANALYSIS

There are two main uses for clustering in Statistical NLP. The figure demonstrates the use of clustering for *exploratory data analysis* (EDA). Somebody who does not know English would be able to derive a crude grouping of words into parts of speech from figure 14.1 and this insight may make subsequent analysis easier. Or we can use the figure to evaluate neighbor overlap as a measure of part-of-speech similarity, assuming we know what the correct parts of speech are. The clustering makes apparent both strengths and weaknesses of a neighbor-based representation. It works well for prepositions (which are all grouped together), but seems inappropriate for other words such as *this* and *the* which are not grouped together with grammatically similar words.

Exploratory data analysis is an important activity in any pursuit that deals with quantitative data. Whenever we are faced with a new problem and want to develop a probabilistic model or just understand the basic characteristics of the phenomenon, EDA is the first step. It is always a mistake to not first spend some time getting a feel for what the data at hand look like. Clustering is a particularly important technique for EDA in Statistical NLP because there is often no direct pictorial visualization for linguistic objects. Other fields, in particular those dealing with numerical or geographic data, often have an obvious visualization, for example, maps of the incidence of a particular disease in epidemiology. Any technique that lets one visualize the data better is likely to bring to the fore new generalizations and to stop one from making wrong assumptions about the data.

There are other well-known techniques for displaying a set of objects in a two-dimensional plane (such as pages of books); see section 14.3 for references. When used for EDA, clustering is thus only one of a number of techniques that one might employ, but it has the advantage that it can produce a richer hierarchical structure. It may also be more convenient to work with since visual displays are more complex. One has to worry about how to label objects shown on the display, and, in contrast to clustering, cannot give a comprehensive description of the object next to its visual representation.

GENERALIZATION

The other main use of clustering in NLP is for *generalization*. We referred to this as *forming bins* or *equivalence classes* in section 6.1. But there we grouped data points in certain predetermined ways, whereas here we induce the bins from data.

As an example, suppose we want to determine the correct preposition to use with the noun *Friday* for translating a text from French into English. Suppose also that we have an English training text that contains the phrases *on Sunday*, *on Monday*, and *on Thursday*, but not *on Friday*. That *on* is the correct preposition to use with *Friday* can be inferred as follows. If we cluster English nouns into groups with similar syntactic and semantic environments, then the days of the week will end up in the same cluster. This is because they share environments like "*until* day-of-the-week," "*last* day-of-the-week," and "day-of-the-week *morning*." Under the assumption that an environment that is correct for one member of the cluster is also correct for the other members of the cluster, we can infer the correctness of *on Friday* from the presence of *on Sunday*, *on Monday* and *on Thursday*. So clustering is a way of *learning*. We group objects into clusters and generalize from what we know about some members of the cluster (like the appropriateness of the preposition *on*) to others.

LEARNING

CLASSIFICATION

Another way of partitioning objects into groups is *classification*, which is the subject of chapter 16. The difference is that classification is *supervised* and requires a set of labeled training instances for each group. Clustering does not require training data and is hence called *unsupervised* because there is no "teacher" who provides a training set with class labels. The result of clustering only depends on natural divisions in the data, for example the different neighbors of prepositions, articles and pronouns in the above dendrogram, not on any pre-existing categorization scheme. Clustering is sometimes called automatic or unsupervised classification, but we will not use these terms in order to avoid confusion.

HIERARCHICAL
FLAT
NON-HIERARCHICAL
ITERATIVE

There are many different clustering algorithms, but they can be classified into a few basic types. There are two types of structures produced by clustering algorithms, *hierarchical clusterings* and *flat* or *non-hierarchical clusterings*. Flat clusterings simply consist of a certain number of clusters and the relation between clusters is often undetermined. Most algorithms that produce flat clusterings are *iterative*. They start with a set of initial clusters and improve them by iterating a reallocation operation that reassigns objects.

A hierarchical clustering is a hierarchy with the usual interpretation that each node stands for a subclass of its mother's node. The leaves of the tree are the single objects of the clustered set. Each node represents the cluster that contains all the objects of its descendants. Figure 14.1 is an example of a hierarchical cluster structure.

Another important distinction between clustering algorithms is whether they perform a *soft clustering* or *hard clustering.* In a hard assignment, each object is assigned to one and only one cluster. Soft assignments allow degrees of membership and membership in multiple clusters. In a probabilistic framework, an object x_i has a probability distribution $P(\cdot|x_i)$ over clusters c_j where $P(c_j|x_i)$ is the probability that x_i is a member of c_j. In a vector space model, degree of membership in multiple clusters can be formalized as the similarity of a vector to the center of each cluster. In a vector space, the center of the M points in a cluster c, otherwise known as the *centroid* or *center of gravity* is the point:

SOFT CLUSTERING
HARD CLUSTERING
CENTROID
CENTER OF GRAVITY

$$\vec{\mu} = \frac{1}{M} \sum_{\vec{x} \in c} \vec{x} \tag{14.1}$$

In other words, each component of the centroid vector $\vec{\mu}$ is simply the average of the values for that component in the M points in c.

In hierarchical clustering, assignment is usually 'hard.' In non-hierarchical clustering, both types of assignment are common. Even most soft assignment models assume that an object is assigned to only one cluster. The difference from hard clustering is that there is uncertainty about which cluster is the correct one. There are also true multiple assignment models, so-called *disjunctive clustering* models, in which an object can truly belong to several clusters. For example, there may be a mix of syntactic and semantic categories in word clustering and *book* would fully belong to both the semantic "object" and the syntactic "noun" category. We will not cover disjunctive clustering models here. See (Saund 1994) for an example of a disjunctive clustering model.

DISJUNCTIVE CLUSTERING

Nevertheless, it is worth mentioning at the beginning the limitations that follow from the assumptions of most clustering algorithms. A hard clustering algorithm has to choose one cluster to which to assign every item. This is rather unappealing for many problems in NLP. It is a commonplace that many words have more than one part of speech. For instance *play* can be a noun or a verb, and *fast* can be an adjective or an adverb. And many larger units also show mixed behavior. Nominalized clauses show some verb-like (clausal) behavior and some noun-like (nominalization) behavior. And we suggested in chapter 7 that several senses of a word were often simultaneously activated. Within a hard clustering framework, the best we can do in such cases is to define additional clusters corresponding to words that can be either nouns or verbs, and so on. Soft clustering is therefore somewhat more appropriate for many prob-

Hierarchical clustering:	Non-hierarchical clustering:
▪ Preferable for detailed data analysis	▪ Preferable if efficiency is a consideration or data sets are very large
▪ Provides more information than flat clustering	▪ K-means is the conceptually simplest method and should probably be used first on a new data set because its results are often sufficient
▪ No single best algorithm (each of the algorithms we describe has been found to be optimal for some application)	▪ K-means assumes a simple Euclidean representation space, and so cannot be used for many data sets, for example, nominal data like colors
▪ Less efficient than flat clustering (for n objects, one minimally has to compute an $n \times n$ matrix of similarity coefficients, and then update this matrix as one proceeds)	▪ In such cases, the EM algorithm is the method of choice. It can accommodate definition of clusters and allocation of objects based on complex probabilistic models.

Table 14.1 A summary of the attributes of different clustering algorithms.

lems in NLP, since a soft clustering algorithm can assign an ambiguous word like *play* partly to the cluster of verbs and partly to the cluster of nouns.

The remainder of the chapter looks in turn at various hierarchical and non-hierarchical clustering methods, and some of their applications in NLP. In table 14.1, we briefly characterize some of the features of clustering algorithms for the reader who is just looking for a quick solution to an immediate clustering need.

For a discussion of the pros and cons of different clustering algorithms see Kaufman and Rousseeuw (1990). The main notations that we will use in this chapter are summarized in table 14.2.

14.1 Hierarchical Clustering

The tree of a hierarchical clustering can be produced either bottom-up, by starting with the individual objects and grouping the most similar

Notation	Meaning
$\mathcal{X} = \{x_1, \ldots, x_n\}$	the set of n objects to be clustered
$C = \{c_1, \ldots, c_j, \ldots c_k\}$	the set of clusters (or cluster hypotheses)
$\mathcal{P}(\mathcal{X})$	powerset (set of subsets) of $\mathcal{X}$
$\mathrm{sim}(\cdot, \cdot)$	similarity function
$S(\cdot)$	group average similarity function
m	Dimensionality of vector space $\mathbb{R}^m$
M_j	Number of points in cluster c_j
$\vec{s}(c_j)$	Vector sum of vectors in cluster c_j
N	number of word tokens in training corpus
$w_{i,\ldots,j}$	tokens i through j of the training corpus
$\pi(\cdot)$	function assigning words to clusters
$C(w^1 w^2)$	number of occurrences of string $w^1 w^2$
$C(c_1 c_2)$	number of occurrences of string $w^1 w^2$ s.t. $\pi(w^1) = c_1$, $\pi(w^2) = c_2$
$\vec{\mu}_j$	Centroid for cluster c_j
Σ_j	Covariance matrix for cluster c_j

Table 14.2 Symbols used in the clustering chapter.

ones, or top-down, whereby one starts with all the objects and divides them into groups so as to maximize within-group similarity. Figure 14.2 describes the bottom-up algorithm, also called *agglomerative clustering*. Agglomerative clustering is a greedy algorithm that starts with a separate cluster for each object (3,4). In each step, the two most similar clusters are determined (8), and merged into a new cluster (9). The algorithm terminates when one large cluster containing all objects of S has been formed, which then is the only remaining cluster in C (7).

AGGLOMERATIVE CLUSTERING

Let us flag one possibly confusing issue. We have phrased the clustering algorithm in terms of similarity between clusters, and therefore we join things with *maximum* similarity (8). Sometimes people think in terms of distances between clusters, and then you want to join things that are the *minimum* distance apart. So it is easy to get confused between whether you're taking maximums or minimums. It is straightforward to produce a similarity measure from a distance measure d, for example by $\mathrm{sim}(x, y) = 1/(1 + d(x, y))$.

DIVISIVE CLUSTERING

Figure 14.3 describes top-down hierarchical clustering, also called *divisive clustering* (Jain and Dubes 1988: 57). Like agglomerative clustering

```
1  Given: a set $\mathcal{X} = \{x_1, \dots x_n\}$ of objects
2         a function sim: $\mathcal{P}(\mathcal{X}) \times \mathcal{P}(\mathcal{X}) \to \mathbb{R}$
3  for $i := 1$ to $n$ do
4      $c_i := \{x_i\}$ end
5  $C := \{c_1, \dots, c_n\}$
6  $j := n + 1$
7  while $C > 1$
8          $(c_{n_1}, c_{n_2}) := \arg\max_{(c_u, c_v) \in C \times C} \mathrm{sim}(c_u, c_v)$
9          $c_j = c_{n_1} \cup c_{n_2}$
10         $C := C \backslash \{c_{n_1}, c_{n_2}\} \cup \{c_j\}$
11         $j := j + 1$
```

Figure 14.2 Bottom-up hierarchical clustering.

```
1  Given: a set $\mathcal{X} = \{x_1, \dots x_n\}$ of objects
2         a function coh: $\mathcal{P}(\mathcal{X}) \to \mathbb{R}$
3         a function split: $\mathcal{P}(\mathcal{X}) \to \mathcal{P}(\mathcal{X}) \times \mathcal{P}(\mathcal{X})$
4  $C := \{\mathcal{X}\}$  $(= \{c_1\})$
5  $j := 1$
6  while $\exists c_i \in C$ s.t. $|c_i| > 1$
7          $c_u := \arg\min_{c_v \in C} \mathrm{coh}(c_v)$
8          $(c_{j+1}, c_{j+2}) = \mathrm{split}(c_u)$
9          $C := C \backslash \{c_u\} \cup \{c_{j+1}, c_{j+2}\}$
10         $j := j + 2$
```

Figure 14.3 Top-down hierarchical clustering.

it is a greedy algorithm. Starting from a cluster with all objects (4), each iteration determines which cluster is least coherent (7) and splits this cluster (8). Clusters with similar objects are more coherent than clusters with dissimilar objects. For example, a cluster with several identical members is maximally coherent.

MONOTONIC

Hierarchical clustering only makes sense if the similarity function is *monotonic:*

(14.2) **Monotonicity.**
$$\forall c, c', c'' \subseteq S : \min\big(\mathrm{sim}(c, c'), \mathrm{sim}(c, c'')\big) \geq \mathrm{sim}(c, c' \cup c'')$$

In other words, the operation of merging is guaranteed to not increase similarity. A similarity function that does not obey this condition makes

Function	Definition
single link	similarity of two most similar members
complete link	similarity of two least similar members
group-average	average similarity between members

Table 14.3 Similarity functions used in clustering. Note that for group-average clustering, we average over all pairs, including pairs from the same cluster. For single-link and complete-link clustering, we quantify over the subset of pairs from different clusters.

the hierarchy uninterpretable since dissimilar clusters, which are placed far apart in the tree, can become similar in subsequent merging so that 'closeness' in the tree does not correspond to conceptual similarity anymore.

Most hierarchical clustering algorithms follow the schemes outlined in figures 14.2 and 14.3. The following sections discuss specific instances of these algorithms.

14.1.1 Single-link and complete-link clustering

Table 14.3 shows three similarity functions that are commonly used in information retrieval (van Rijsbergen 1979: 36ff). Recall that the similarity function determines which clusters are merged in each step in bottom-up clustering. In single-link clustering the similarity between two clusters is the similarity of the two closest objects in the clusters. We search over all pairs of objects that are from the two different clusters and select the pair with the greatest similarity.

LOCAL COHERENCE

Single-link clusterings have clusters with good *local coherence* since the similarity function is locally defined. However, clusters can be elongated or "straggly" as shown in figure 14.6. To see why single-link clustering produces such elongated clusters, observe first that the best moves in figure 14.4 are to merge the two top pairs of points and then the two bottom pairs of points, since the similarities a/b, c/d, e/f, and g/h are the largest for any pair of objects. This gives us the clusters in figure 14.5. The next two steps are to first merge the top two clusters, and then the bottom two clusters, since the pairs b/c and f/g are closer than all others that are not in the same cluster (e.g., closer than b/f and c/g). After doing these two merges we get figure 14.6. We end up with two clusters that

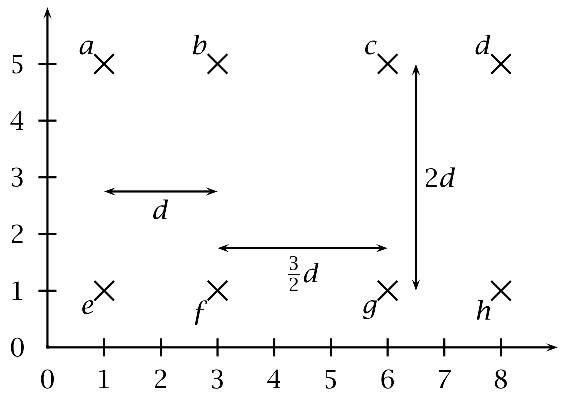

Figure 14.4 A cloud of points in a plane.

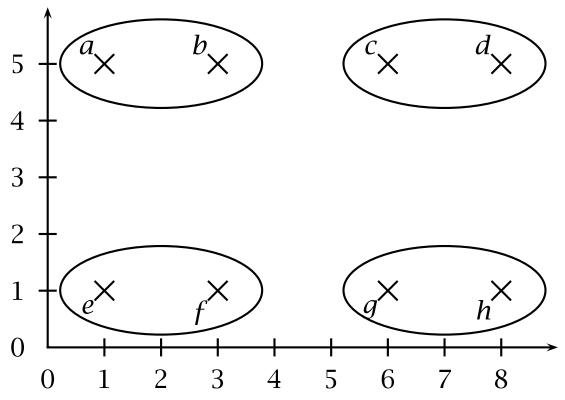

Figure 14.5 Intermediate clustering of the points in figure 14.4.

are locally coherent (meaning that close objects are in the same cluster), but which can be regarded as being of bad global quality. An example of bad global quality is that *a* is much closer to *e* than to *d*, yet *a* and *d* are in the same cluster whereas *a* and *e* are not.

CHAINING EFFECT

The tendency of single-link clustering to produce this type of elongated cluster is sometimes called the *chaining effect* since we follow a chain of large similarities without taking into account the global context.

MINIMUM SPANNING TREE

Single-link clustering is closely related to the *minimum spanning tree* (MST) of a set of points. The MST is the tree that connects all objects with edges that have the largest similarities. That is, of all trees connecting the set of objects the sum of the length of the edges of the MST is mini-

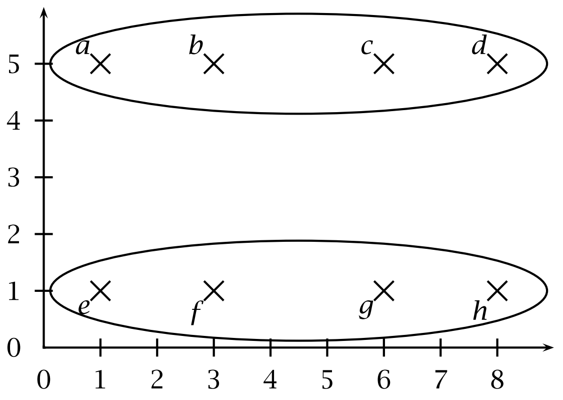

Figure 14.6 Single-link clustering of the points in figure 14.4.

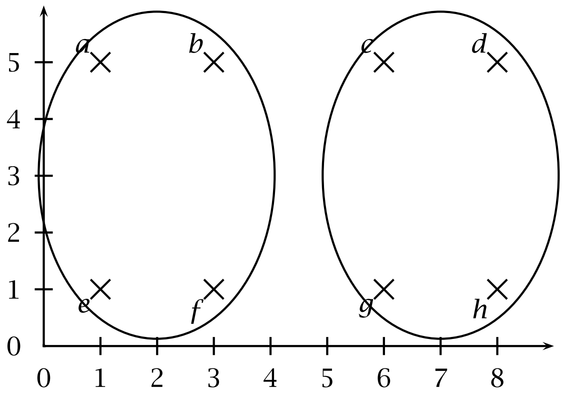

Figure 14.7 Complete-link clustering of the points in figure 14.4.

mal. A single-link hierarchy can be constructed top-down from an MST by removing the longest edge in the MST so that two unconnected components are created, corresponding to two subclusters. The same operation is then recursively applied to these two subclusters (which are also MSTs).

Complete-link clustering has a similarity function that focuses on *global* cluster quality (as opposed to *locally* coherent clusters as in the case of single-link clustering). The similarity of two clusters is the similarity of their two most dissimilar members. Complete-link clustering avoids elongated clusters. For example, in complete-link clustering the two best merges in figure 14.5 are to merge the two left clusters, and then the two right clusters, resulting in the clusters in figure 14.7. Here, the minimally similar pair for the left clusters (a/f or b/e) is "tighter" than the minimally similar pair of the two top clusters (a/d).

So far we have made the assumption that 'tight' clusters are better than 'straggly' clusters. This reflects an intuition that a cluster is a group of objects centered around a central point, and so compact clusters are to be preferred. Such an intuition corresponds to a model like the Gaussian distribution (section 2.1.9), which gives rise to sphere-like clusters. But this is only one possible underlying model of what a good cluster is. It is really a question of our prior knowledge about and model of the data which determines what a good cluster is. For example, the Hawai'ian islands were produced (and are being produced) by a volcanic process which moves along a straight line and creates new volcanoes at more or less regular intervals. Single-link is a very appropriate clustering model here since local coherence is what counts and elongated clusters are what we would expect (say, if we wanted to group several chains of volcanic islands). It is important to remember that the different clustering algorithms that we discuss will generally produce different results which incorporate the somewhat ad hoc biases of the different algorithms. Nevertheless, in most NLP applications, the sphere-shaped clusters of complete-link clustering are preferable to the elongated clusters of single-link clustering.

The disadvantage of complete-link clustering is that it has time complexity $O(n^3)$ since there are n merging steps and each step requires $O(n^2)$ comparisons to find the smallest similarity between any two objects for each cluster pair (where n is the number of objects to be clustered).[1] In contrast, single-link clustering has complexity $O(n^2)$. Once the $n \times n$ similarity matrix for all objects has been computed, it can be updated after each merge in $O(n)$: if clusters c_u and c_v are merged into $c_j = c_u \cup c_v$, then the similarity of the merge with another cluster c_k is simply the maximum of the two individual similarities:

$$\text{sim}(c_j, c_k) = \max(\text{sim}(c_u, c_k), \text{sim}(c_v, c_k))$$

Each of the $n - 1$ merges requires at most n constant-time updates. Both merging and similarity computation thus have complexity $O(n^2)$

1. '$O(n^3)$' is an instance of 'Big Oh' notation for algorithmic complexity. We assume that the reader is familiar with it, or else is willing to skip issues of algorithmic complexity. It is defined in most books on algorithms, including (Cormen et al. 1990). The notation describes just the basic dependence of an algorithm on certain parameters, while ignoring constant factors.

in single-link clustering, which corresponds to an overall complexity of $O(n^2)$.

Single-link and complete-link clustering can be graph-theoretically interpreted as finding a maximally connected and maximally complete graph (or clique), respectively, hence the term "complete link" for the latter. See (Jain and Dubes 1988: 64).

14.1.2 Group-average agglomerative clustering

Group-average agglomerative clustering is a compromise between single-link and complete-link clustering. Instead of the greatest similarity between elements of clusters (single-link) or the least similarity (complete link), the criterion for merges is average similarity. We will see presently that average similarity can be computed efficiently in some cases so that the complexity of the algorithm is only $O(n^2)$. The group-average strategy is thus an efficient alternative to complete-link clustering while avoiding the elongated and straggly clusters that occur in single-link clustering.

Some care has to be taken in implementing group-average agglomerative clustering. The complexity of computing average similarity directly is $O(n^2)$. So if the average similarities are computed from scratch each time a new group is formed, that is, in each of the n merging steps, then the algorithm would be $O(n^3)$. However, if the objects are represented as length-normalized vectors in an m-dimensional real-valued space and if

COSINE the similarity measure is the *cosine*, defined as in (14.3):

$$\text{sim}(\vec{v}, \vec{w}) = \frac{\sum_{i=1}^{m} v_i \times w_i}{\sqrt{\sum_{i=1}^{m} v_i \times \sum_{i=1}^{m} w_i}} = \vec{x} \cdot \vec{y} \quad (14.3)$$

then there exists an algorithm that computes the average similarity of a cluster in constant time from the average similarity of its two children. Given the constant-time for an individual merging operation, the overall time complexity is $O(n^2)$.

We write $\mathcal{X}$ for the set of objects to be clustered, each represented by a m-dimensional vector:

$$\mathcal{X} \subseteq \mathbb{R}^m$$

For a cluster $c_j \subseteq \mathcal{X}$, the average similarity S between vectors in c_j is defined as follows. (The factor $|c_j|(|c_j|-1)$ calculates the number of (non-zero) similarities added up in the double summation.)

$$S(c_j) = \frac{1}{|c_j|(|c_j|-1)} \sum_{\vec{x} \in c_j} \sum_{\vec{x} \neq \vec{y} \in c_j} \mathrm{sim}(\vec{x}, \vec{y}) \tag{14.4}$$

Let C be the set of current clusters. In each iteration, we identify the two clusters c_u and c_v which maximize $S(c_u \cup c_v)$. This corresponds to step 8 in figure 14.2. A new, smaller, partition C' is then constructed by merging c_u and c_v (step 10 in figure 14.2):

$$C' = (C - \{c_u, c_v\}) \cup \{c_u \cup c_v\}$$

For cosine as the similarity measure, the inner maximization can be done in linear time (Cutting et al. 1992: 328). One can compute the average similarity between the elements of a candidate pair of clusters in constant time by precomputing for each cluster the sum of its members $\vec{s}(c_j)$.

$$\vec{s}(c_j) = \sum_{\vec{x} \in c_j} \vec{x}$$

The sum vector $\vec{s}(c_j)$ is defined in such a way that: (i) it can be easily updated after a merge (namely by simply summing the $\vec{s}$ of the clusters that are being merged), and (ii) the average similarity of a cluster can be easily computed from them. This is so because the following relationship between $\vec{s}(c_j)$ and $S(c_j)$ holds:

$$\begin{aligned} \vec{s}(c_j) \cdot \vec{s}(c_j) &= \sum_{\vec{x} \in c_j} \vec{x} \cdot \vec{s}(c_j) \\ &= \sum_{\vec{x} \in c_j} \sum_{\vec{y} \in c_j} \vec{x} \cdot \vec{y} \\ &= |c_j|(|c_j|-1)S(c_j) + \sum_{\vec{x} \in c_j} \vec{x} \cdot \vec{x} \\ &= |c_j|(|c_j|-1)S(c_j) + |c_j| \\ \text{Thus,} \quad S(c_j) &= \frac{\vec{s}(c_j) \cdot \vec{s}(c_j) - |c_j|}{|c_j|(|c_j|-1)} \end{aligned} \tag{14.5}$$

Therefore, if $\vec{s}(\cdot)$ is known for two groups c_i and c_j, then the average similarity of their union can be computed in constant time as follows:

$$S(c_i \cup c_j) = \frac{(\vec{s}(c_i) + \vec{s}(c_j)) \cdot (\vec{s}(c_i) + \vec{s}(c_j))}{(|c_i| + |c_j|)(|c_i| + |c_j| - 1)} \tag{14.6}$$

Given this result, this approach to group-average agglomerative clustering has complexity $O(n^2)$, reflecting the fact that initially all pairwise similarities have to be computed. The following step that performs n mergers (each in linear time) has linear complexity, so that overall complexity is quadratic.

This form of group-average agglomerative clustering is efficient enough to deal with a large number of features (corresponding to the dimensions of the vector space) and a large number of objects. Unfortunately, the constant time computation for merging two groups (by making use of the quantities $\vec{s}(c_j)$) depends on the properties of vector spaces. There is no general algorithm for group-average clustering that would be efficient independent of the representation of the objects that are to be clustered.

14.1.3 An application: Improving a language model

LANGUAGE MODEL

Now that we have introduced some of the best known hierarchical clustering algorithms, it is time to look at an example of how clustering can be used for an application. The application is building a better *language model*. Recall that language models are useful in speech recognition and machine translation for choosing among several candidate hypotheses. For example, a speech recognizer may find that *President Kennedy* and *precedent Kennedy* are equally likely to have produced the acoustic observations. However, a language model can tell us what are *a priori* likely phrases of English. Here it tell us that *President Kennedy* is much more likely than *precedent Kennedy*, and so we conclude that *President Kennedy* is probably what was actually said. This reasoning can be formalized by the equation for the noisy channel model, which we introduced in section 2.2.4. It says that we should choose the hypothesis H that maximizes the product of the probability given by the language model, $P(H)$, and the conditional probability of observing the speech signal D (or the foreign language text in machine translation) given the hypothesis, $P(D|H)$.

$$\hat{H} = \arg\max_H P(H|D) = \arg\max_H \frac{P(D|H)P(H)}{P(D)} = \arg\max_H P(D|H)P(H)$$

Clustering can play an important role in improving the language model (the computation of $P(H)$) by way of *generalization*. As we saw in chapter 6, there are many rare events for which we do not have enough training data for accurate probabilistic modeling. If we mediate probabilistic inference through clusters, for which we have more evidence in the training set, then our predictions for rare events are likely to be more accurate. This approach was taken by Brown et al. (1992c). We first describe the formalization of the language model and then the clustering algorithm.

The language model

The language model under discussion is a bigram model that makes a first order Markov assumption that a word depends only on the previous word. The criterion that we optimize is a decrease in *cross entropy* or, equivalently, *perplexity* (section 2.2.8), the amount by which the language model reduces the uncertainty about the next word. Our aim is to find a function π that assigns words to clusters which decreases perplexity compared to a simple word bigram model.

CROSS ENTROPY

PERPLEXITY

We first approximate the cross entropy of the corpus $L = w_1 \dots w_N$ for the cluster assignment function π by making the Markov assumption that a word's occurrence only depends on its predecessor:

$$H(L,\pi) = -\frac{1}{N}\log P(w_{1,\dots,N}) \tag{14.7}$$

$$\approx \frac{-1}{N-1}\log\prod_{i=2}^{N} P(w_i|w_{i-1}) \tag{14.8}$$

$$\approx \frac{-1}{N-1}\sum_{w^1w^2} C(w^1w^2)\log P(w^2|w^1) \tag{14.9}$$

Now we make the basic assumption of cluster-based generalization that the occurrence of a word from cluster c_2 only depends on the cluster c_1 of the preceding word:[2]

$$H(L,\pi) \approx \frac{-1}{N-1}\sum_{w^1w^2} C(w^1w^2)\log P(c_2|c_1)P(w^2|c_2) \tag{14.10}$$

Formula (14.10) can be simplified as follows:

$$H(L,\pi) \approx -\left[\sum_{w^1w^2}\frac{C(w^1w^2)}{N-1}[\log P(w^2|c_2) + \log P(c_2)]\right. \tag{14.11}$$

2. One can observe that this equation is very similar to the probabilistic models used in tagging, which we discuss in chapter 10, except that we induce the word classes from corpus evidence instead of taking them from our linguistic knowledge about parts of speech.

$$+ \sum_{w^1 w^2} \frac{C(w^1 w^2)}{N-1} [\log P(c_2|c_1) - \log P(c_2)] \Bigg]$$

$$= -\left[\sum_{w^2} \frac{\sum_{w^1} C(w^1 w^2)}{N-1} \log P(w^2|c_2) P(c_2) + \sum_{c_1 c_2} \frac{C(c_1 c_2)}{N-1} \log \frac{P(c_2|c_1)}{P(c_2)} \right] \tag{14.12}$$

$$\approx -\left[\sum_{w} P(w) \log P(w) + \sum_{c_1 c_2} P(c_1 c_2) \log \frac{P(c_1 c_2)}{P(c_1) P(c_2)} \right] \tag{14.13}$$

$$= H(w) - I(c_1;\, c_2) \tag{14.14}$$

In (14.13) we rely on the approximations $\frac{\sum_{w^1} C(w^1 w^2)}{N-1} \approx P(w^2)$ and $\frac{C(c_1 c_2)}{N-1} \approx P(c_1 c_2)$, which hold for large n. In addition, $P(w^2|c_2)P(c_2) = P(w^2 c_2) = P(w^2)$ holds since $\pi(w^2) = c_2$.

Equation (14.14) shows that we can minimize the cross entropy by choosing the cluster assignment function π such that the mutual information between adjacent clusters $I(c_1;\, c_2)$ is maximized. Thus we should get the optimal language model by choosing clusters that maximize this mutual information measure.

Clustering

The clustering algorithm is bottom-up with the following merge criterion which maximizes the mutual information between adjacent classes:

$$\text{MI-loss}(c_i, c_j) = \sum_{c_k \in C \setminus \{c_i, c_j\}} I(c_k;\, c_i) + I(c_k;\, c_j) - I(c_k;\, c_i \cup c_j) \tag{14.15}$$

In each step, we select the two clusters whose merge causes the smallest loss in mutual information. In the description of bottom-up clustering in figure 14.2, this would correspond to the following selection criterion for the pair of clusters that is to be merged next:

$$(c_{n_1}, c_{n_2}) := \underset{(c_i, c_j) \in C \times C}{\arg\min}\ \text{MI-loss}(c_i, c_j)$$

The clustering is stopped when a pre-determined number k of clusters has been reached ($k = 1000$ in (Brown et al. 1992c)). Several shortcuts are necessary to make the computation of the MI-loss function and the clustering of a large vocabulary efficient. In addition, the greedy algorithm ("do the merge with the smallest MI-loss") does not guarantee an optimal clustering result. The clusters can be (and were) improved by moving individual words between clusters. The interested reader can look up the specifics of the algorithm in (Brown et al. 1992c).

Here are three of the 1000 clusters found by Brown et al. (1992c):

- plan, letter, request, memo, case, question, charge, statement, draft
- day, year, week, month, quarter, half
- evaluation, assessment, analysis, understanding, opinion, conversation, discussion

We observe that these clusters are characterized by both syntactic and semantic properties, for example, nouns that refer to time periods.

The perplexity for the cluster-based language model was 277 compared to a perplexity of 244 for a word-based model (Brown et al. 1992c: 476), so no direct improvement was achieved by clustering. However, a linear interpolation (see section 6.3.1) between the word-based and the cluster-based model had a perplexity of 236, which is an improvement over the word-based model (Brown et al. 1992c: 476). This example demonstrates the utility of clustering for the purpose of generalization.

We conclude our discussion by pointing out that clustering and cluster-based inference are integrated here. The criterion we optimize on in clustering, the minimization of $H(L, \pi) = H(w) - I(c_1; c_2)$, is at the same time a measure of the quality of the language model, the ultimate goal of the clustering. Other researchers first induce clusters and then use these clusters for generalization in a second, independent step. An integrated approach to clustering and cluster-based inference is preferable because it guarantees that the induced clusters are optimal for the particular type of generalization that we intend to use the clustering for.

14.1.4 Top-down clustering

Hierarchical top down clustering as described in figure 14.3 starts out with one cluster that contains all objects. The algorithm then selects the least coherent cluster in each iteration and splits it. The functions we introduced in table 14.3 for selecting the best pair of clusters to merge in bottom-up clustering can also serve as measures of cluster coherence in top-down clustering. According to the single-link measure, the coherence of a cluster is the smallest similarity in the minimum spanning tree for the cluster; according to the complete-link measure, the coherence is the smallest similarity between any two objects in the cluster; and according to the group-average measure, coherence is the average similarity between objects in the cluster. All three measures can be used to select the least coherent cluster in each iteration of top-down clustering.

Splitting a cluster is also a clustering task, the task of finding two subclusters of the cluster. Any clustering algorithm can be used for the splitting operation, including the bottom-up algorithms described above and non-hierarchical clustering. Perhaps because of this recursive need for a second clustering algorithm, top-down clustering is less often used than bottom-up clustering.

However, there are tasks for which top-down clustering is the more natural choice. An example is the clustering of probability distributions using the *Kullback-Leibler (KL) divergence*. Recall that KL divergence which we introduced in section 2.2.5 is defined as follows:

KULLBACK-LEIBLER DIVERGENCE

(14.16) $$D(\mathrm{p} \parallel \mathrm{q}) = \sum_{x \in X} \mathrm{p}(x) \log \frac{\mathrm{p}(x)}{\mathrm{q}(x)}$$

This "dissimilarity" measure is not defined for $\mathrm{p}(x) > 0$ and $\mathrm{q}(x) = 0$. In cases where individual objects have probability distributions with many zeros, one cannot compute the matrix of similarity coefficients for all objects that is required for bottom-up clustering.

An example of such a constellation is the approach to distributional clustering of nouns proposed by (Pereira et al. 1993). Object nouns are represented as probability distributions over verbs, where $\mathrm{q}_n(v)$ is estimated as the relative frequency that, given the object noun n, the verb v is its predicate. So for example, for the noun *apple* and the verb *eat*, we will have $\mathrm{q}_n(v) = 0.2$ if one fifth of all occurrences of *apple* as an object noun are with the verb *eat*. Any given noun only occurs with a limited number of verbs, so we have the above-mentioned problem with singularities in computing KL divergence here, which prevents us from using bottom-up clustering.

DISTRIBUTIONAL NOUN CLUSTERING

To address this problem, *distributional noun clustering* instead performs top-down clustering. Cluster centroids are computed as (weighted and normalized) sums of the probability distributions of the member nouns. This leads to cluster centroid distributions with few zeros that have a defined KL divergence with all their members. See Pereira et al. (1993) for a complete description of the algorithm.

14.2 Non-Hierarchical Clustering

Non-hierarchical algorithms often start out with a partition based on randomly selected seeds (one seed per cluster), and then refine this initial partition. Most non-hierarchical algorithms employ several passes of *reallocating* objects to the currently best cluster whereas hierarchical algorithms need only one pass. However, reallocation of objects from one cluster to another can improve hierarchical clusterings too. We saw an example in section 14.1.3, where after each merge objects were moved around to improve global mutual information.

REALLOCATING

If the non-hierarchical algorithm has multiple passes, then the question arises when to stop. This can be determined based on a measure of goodness or cluster quality. We have already seen candidates of such a measure, for example, group-average similarity and mutual information between adjacent clusters. Probably the most important stopping criterion is the likelihood of the data given the clustering model which we will introduce below. Whichever measure we choose, we simply continue clustering as long as the measure of goodness improves enough in each iteration. We stop when the curve of improvement flattens or when goodness starts decreasing.

The measure of goodness can address another problem: how to determine the right number of clusters. In some cases, we may have some prior knowledge about the right number of clusters (for example, the right number of parts of speech in part-of-speech clustering). If this is not the case, we can cluster the data into n clusters for different values of n. Often the goodness measure improves with n. For example, the more clusters the higher the maximum mutual information that can be attained for a given data set. However, if the data naturally fall into a certain number k of clusters, then one can often observe a substantial increase in goodness in the transition from $k-1$ to k clusters and a small increase in the transition from k to $k+1$. In order to automatically determine the number of clusters, we can look for a k with this property and then settle on the resulting k clusters.

MINIMUM DESCRIPTION LENGTH
AUTOCLASS

A more principled approach to finding an optimal number of clusters is the *Minimum Description Length* (MDL) approach in the *AUTOCLASS* system (Cheeseman et al. 1988). The basic idea is that the measure of goodness captures both how well the objects fit into the clusters (which is what the other measures we have seen do) and how many clusters there are. A high number of clusters will be penalized, leading to a lower goodness value. In the framework of MDL, both the clusters and the objects are specified by code words whose length is measured in bits. The more clusters there are, the fewer bits are necessary to encode the objects. In order to encode an object, we only encode the difference between it and the cluster it belongs to. If there are more clusters, the clusters describe objects better, and we need fewer bits to describe the difference between objects and clusters. However, more clusters obviously take more bits to encode. Since the cost function captures the length of the code for both data and clusters, minimizing this function (which maximizes the goodness of the clustering) will determine both the number of clusters and how to assign objects to clusters.[3]

It may appear that it is an advantage of hierarchical clustering that the number of clusters need not be determined. But the full cluster hierarchy of a set of objects does not define a particular clustering since the tree can be cut in many different ways. For a usable set of clusters in hierarchical clustering one often needs to determine a desirable number of clusters or, alternatively, a value of the similarity measure at which links of the tree are cut. So there is not really a difference between hierarchical and non-hierarchical clustering in this respect. For some non-hierarchical clustering algorithms, an advantage is their speed.

We cover two non-hierarchical clustering algorithms in this section, K-means and the EM algorithm. K-means clustering is probably the simplest clustering algorithm and, despite its limitations, it works sufficiently well in many applications. The EM algorithm is a general template for a family of algorithms. We describe its incarnation as a clustering algorithm first and then relate it to the various instantiations that have been used in Statistical NLP, some of which like the inside-outside algorithm and the forward-backward algorithm are more fully treated in other chapters of this book.

14.2.1 K-means

K-MEANS *K-means* is a hard clustering algorithm that defines clusters by the center of mass of their members. We need a set of initial cluster centers in the beginning. Then we go through several iterations of assigning each object to the cluster whose center is closest. After all objects have been
RECOMPUTATION assigned, we *recompute* the center of each cluster as the centroid or *mean*

3. AUTOCLASS can be downloaded from the internet. See the website.

```
Given: a set $\mathcal{X} = \{\vec{x}_1, \ldots, \vec{x}_n\} \subseteq \mathbb{R}^m$
       a distance measure $d : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$
       a function for computing the mean $\mu : \mathcal{P}(\mathbb{R}) \to \mathbb{R}^m$
Select $k$ initial centers $\vec{f}_1, \ldots, \vec{f}_k$
while stopping criterion is not true do
    for all clusters $c_j$ do
        $c_j = \{\vec{x}_i \mid \forall \vec{f}_l \; d(\vec{x}_i, \vec{f}_j) \leq d(\vec{x}_i, \vec{f}_l)\}$
    end
    for all means $\vec{f}_j$ do
        $\vec{f}_j = \mu(c_j)$
    end
end
```

Figure 14.8 The K-means clustering algorithm.

$\vec{\mu}$ of its members (see figure 14.8), that is $\vec{\mu} = (1/|c_j|) \sum_{\vec{x} \in c_j} \vec{x}$. The distance function is Euclidean distance.

A variant of K-means is to use the L_1 norm instead (section 8.5.2):

$$L_1(\vec{x}, \vec{y}) = \sum_l |x_l - y_l|$$

This norm is less sensitive to outliers. K-means clustering in Euclidean space often creates singleton clusters for outliers. Clustering in L_1 space will pay less attention to outliers so that there is higher likelihood of getting a clustering that partitions objects into clusters of similar size. MEDOIDS The L_1 norm is often used in conjunction with *medoids* as cluster centers. The difference between medoids and centroids is that a medoid is one of the objects in the cluster – a prototypical class member. A centroid, the average of a cluster's members, is in most cases not identical to any of the objects.

The time complexity of K-means is $O(n)$ since both steps of the iteration are $O(n)$ and only a constant number of iterations is computed.

Figure 14.9 shows an example of one iteration of the K-means algorithm. First, objects are assigned to the cluster whose mean is closest. Then the means are recomputed. In this case, any further iterations will not change the clustering since an assignment to the closest center does not change the cluster membership of any object, which in turn means that no center will be changed in the recomputation step. But this is

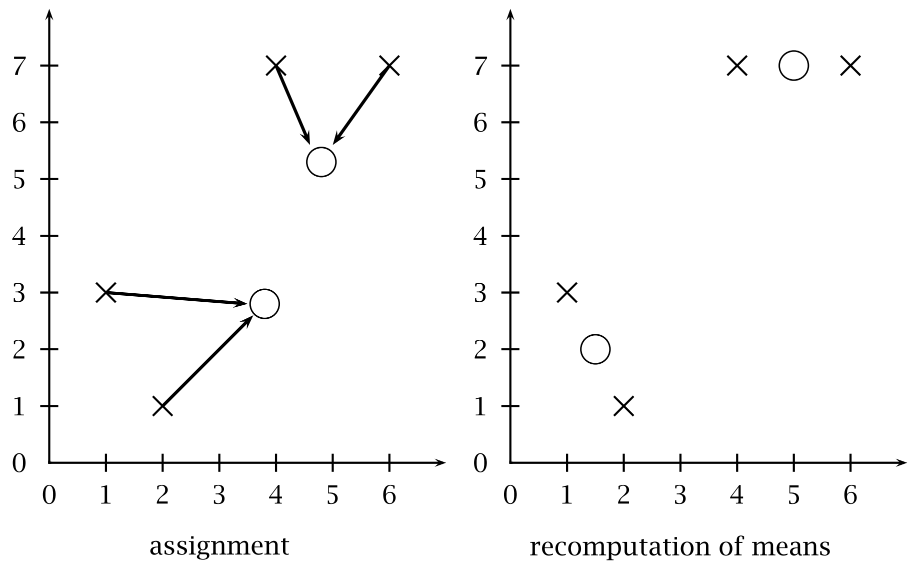

Figure 14.9 One iteration of the K-means algorithm. The first step assigns objects to the closest cluster mean. Cluster means are shown as circles. The second step recomputes cluster means as the center of mass of the set of objects that are members of the cluster.

not the case in general. Usually several iterations are required before the algorithm converges.

One implementation problem that the description in figure 14.8 does not address is how to break ties in cases where there are several centers with the same distance from an object. In such cases, one can either assign objects randomly to one of the candidate clusters (which has the disadvantage that the algorithm may not converge) or perturb objects slightly so that their new positions do not give rise to ties.

Here is an example of how to use K-means clustering. Consider these twenty words from the *New York Times* corpus in chapter 5.

> Barbara, Edward, Gov, Mary, NFL, Reds, Scott, Sox, ballot, finance, inning, payments, polls, profit, quarterback, researchers, science, score, scored, seats

Cluster	Members
1	*ballot* (0.28), *polls* (0.28), *Gov* (0.30), *seats* (0.32)
2	*profit* (0.21), *finance* (0.21), *payments* (0.22)
3	*NFL* (0.36), *Reds* (0.28), *Sox* (0.31), *inning* (0.33), *quarterback* (0.30), *scored* (0.30), *score* (0.33)
4	*researchers* (0.23), *science* (0.23)
5	*Scott* (0.28), *Mary* (0.27), *Barbara* (0.27), *Edward* (0.29)

Table 14.4 An example of K-means clustering. Twenty words represented as vectors of co-occurrence counts were clustered into 5 clusters using K-means. The distance from the cluster centroid is given after each word.

Table 14.4 shows the result of clustering these words using K-means with $k = 5$. We used the data representation from chapter 8 that is also the basis of table 8.8 on page 302. The first four clusters correspond to the topics 'government,' 'finance,' 'sports,' and 'research,' respectively. The last cluster contains names. The benefit of clustering is obvious here. The clustered display of the words makes it easier to understand what types of words occur in the sample and what their relationships are.

Initial cluster centers for K-means are usually picked at random. It depends on the structure of the set of objects to be clustered whether the choice of initial centers is important or not. Many sets are well-behaved and most initializations will result in clusterings of about the same quality.

For ill-behaved sets, one can compute good cluster centers by first running a hierarchical clustering algorithm on a subset of the objects. This is the basic idea of the *Buckshot* algorithm. Buckshot first applies group-average agglomerative clustering (GAAC) to a random sample of the data that has size square root of the complete set. GAAC has quadratic time complexity, but since $(\sqrt{n})^2 = n$, applying GAAC to this sample results in overall linear complexity of the algorithm. The K-means reassignment step is also linear, so that the overall complexity is $O(n)$.

BUCKSHOT

14.2.2 The EM algorithm

One way to introduce the EM algorithm is as a 'soft' version of K-means clustering. Figure 14.10 shows an example. As before, we start with a set of random cluster centers, c_1 and c_2. In K-means clustering we would

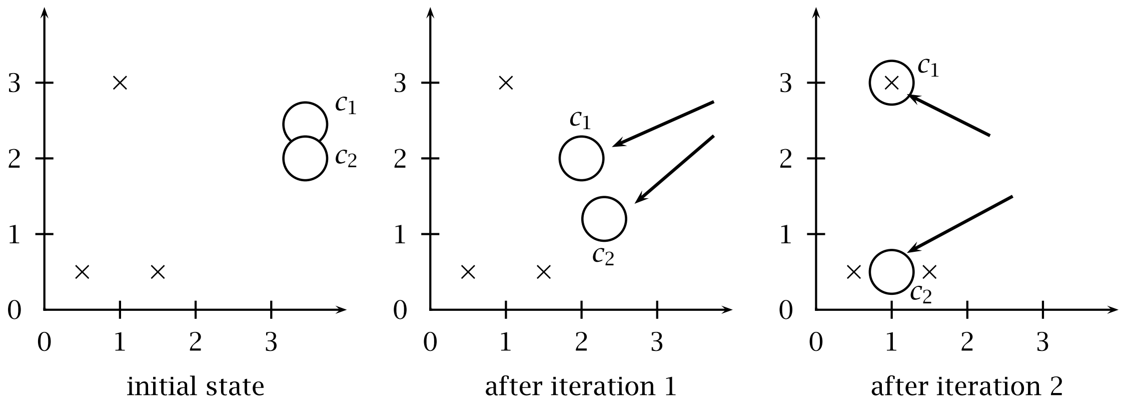

Figure 14.10 An example of using the EM algorithm for soft clustering.

arrive at the final centers shown on the right side in one iteration. The EM algorithm instead does a soft assignment, which, for example, makes the lower right point mostly a member of c_2, but also partly a member of c_1. As a result, both cluster centers move towards the centroid of all three objects in the first iteration. Only after the second iteration do we reach the stable final state.

An alternative way of thinking of the EM algorithm is as a way of estimating the values of the hidden parameters of a model. We have seen some data X, and can estimate $P(X|\mathrm{p}(\Theta))$, the probability of the data according to some model p with parameters Θ. But how do we find the model which maximizes the likelihood of the data? This point will be a maximum in the parameter space, and therefore we know that the probability surface will be flat there. So for each model parameter θ_i, we want to set $\frac{\partial}{\partial \theta_i} \log P(\ldots) = 0$ and solve for the θ_i. Unfortunately this (in general) gives a non-linear set of equations for which no analytical methods of solution are known. But we can hope to find the maximum using the EM algorithm.

In this section, we will first introduce the EM algorithm for the estimation of Gaussian mixtures, the soft clustering algorithm that figure 14.10 is an example of. Then we will describe the EM algorithm in its most general form and relate the general form to specific instances like the inside-outside algorithm and the forward-backward algorithm.

EM for Gaussian mixtures

In applying EM to clustering, we view clustering as estimating a mixture of probability distributions. The idea is that the observed data are generated by several underlying causes. Each cause contributes independently to the generation process, but we only see the final mixture – without information about which cause contributed what. We formalize this notion by representing the data as a pair. There is the *observable* data $X = \{\vec{x}_i\}$, where each $\vec{x}_i = (x_{i1}, \ldots, x_{im})^T$ is simply the vector that corresponds to the i^{th} data point. And then there is the *unobservable* data $\mathcal{Z} = \{\vec{z}_i\}$, where within each $\vec{z}_i = z_{i1}, \ldots, z_{ik}$, the component z_{ij} is 1 if object i is a member of cluster j (that is, it is assumed to be generated by that underlying cause) and 0 otherwise.

OBSERVABLE

UNOBSERVABLE

We can cluster with the EM algorithm if we know the type of distribution of the individual clusters (or causes). When estimating a Gaussian mixture, we make the assumption that each cluster is a Gaussian. The EM algorithm then determines the most likely estimates for the parameters of the distributions (in our case, the mean and variance of each Gaussian), and the prior probability (or relative prominence or weight) of the individual causes. So in sum, we are supposing that the data to be clustered consists of n m-dimensional objects $\mathcal{X} = \{\vec{x}_1 \ldots \vec{x}_n\} \subseteq \mathbb{R}^m$ generated by k Gaussians $\mathrm{n}_1 \ldots \mathrm{n}_k$.

Once the mixture has been estimated we can view the result as a clustering by interpreting each cause as a cluster. For each object $\vec{x}_i$, we can compute the probability $P(\omega_j|\vec{x}_i)$ that cluster j generated i. An object can belong to several clusters, with varying degrees of confidence.

Multivariate normal distributions. The (multivariate) m-dimensional *Gaussian* family is parameterized by a mean or center $\vec{\mu}_j$ and an $m \times m$ invertible positive definite symmetric matrix, the *covariance matrix* Σ_j. The probability density function for a Gaussian is given by:

GAUSSIAN

COVARIANCE MATRIX

$$\mathrm{n}_j(\vec{x};\ \vec{m}_j, \Sigma_j) = \frac{1}{\sqrt{(2\pi)^m |\Sigma_j|}} \exp\left[-\frac{1}{2}(\vec{x} - \vec{\mu}_j)^T \Sigma_j^{-1} (\vec{x} - \vec{\mu}_j)\right] \tag{14.17}$$

Since we are assuming that the data is generated by k Gaussians, we wish to find the maximum likelihood model of the form:

$$\sum_{j=1}^{k} \pi_j \mathrm{n}(\vec{x};\ \vec{\mu}_j, \Sigma_j) \tag{14.18}$$

Main		$P(w_i \mid c_j) = \mathrm{n}_j(\vec{x}_i;\ \mu_j \Sigma_j)$				
cluster	Word	1	2	3	4	5
1	*ballot*	0.63	0.12	0.04	0.09	0.11
1	*polls*	0.58	0.11	0.06	0.10	0.14
1	*Gov*	0.58	0.12	0.03	0.10	0.17
1	*seats*	0.55	0.14	0.08	0.08	0.15
2	*profit*	0.11	0.59	0.02	0.14	0.15
2	*finance*	0.15	0.55	0.01	0.13	0.16
2	*payments*	0.12	0.66	0.01	0.09	0.11
3	*NFL*	0.13	0.05	0.58	0.09	0.16
3	*Reds*	0.05	0.01	0.86	0.02	0.06
3	*Sox*	0.05	0.01	0.86	0.02	0.06
3	*inning*	0.03	0.01	0.93	0.01	0.02
3	*quarterback*	0.06	0.02	0.82	0.03	0.07
3	*score*	0.12	0.04	0.65	0.06	0.13
3	*scored*	0.08	0.03	0.79	0.03	0.07
4	*researchers*	0.08	0.12	0.02	0.68	0.10
4	*science*	0.12	0.12	0.03	0.54	0.19
5	*Scott*	0.12	0.12	0.11	0.11	0.54
5	*Mary*	0.10	0.10	0.05	0.15	0.59
5	*Barbara*	0.15	0.11	0.04	0.12	0.57
5	*Edward*	0.16	0.18	0.02	0.12	0.51

Table 14.5 An example of a Gaussian mixture. The five cluster centroids from table 14.4 are the means $\vec{\mu}_j$ of the five clusters. A uniform diagonal covariance matrix $\Sigma = 0.05 \cdot I$ and uniform priors $\pi_j = 0.2$ were used. The posterior probabilities $P(w_i|c_j)$ can be interpreted as cluster membership probabilities.

In this model, we need to assume a prior or weight π_j for each Gaussian, so that the integral of the combined Gaussians over the whole space is 1.

Table 14.5 gives an example of a Gaussian mixture, using the centroids from the K-means clustering in table 14.4 as cluster centroids $\vec{\mu}_j$ (this is a common way of initializing EM for Gaussian mixtures). For each word, the cluster from table 14.4 is still the dominating cluster. For example, *ballot* has a higher membership probability in cluster 1 (its cluster from the K-means clustering) than in other clusters. But each word also has some non-zero membership in all other clusters. This is useful for assessing the strength of association between a word and a topic. Comparing two members of the 'sports' cluster, *inning* and *score*, we can see that *inning* is strongly associated with 'sports' ($p = 0.93$) whereas *score* has some affinity with other clusters as well (e.g., $p = 0.12$ with the 'government' cluster). This is a good example of the utility of soft clustering.

We now develop the EM algorithm for estimating the parameters of a Gaussian mixture. Let us write $\theta_j = (\vec{\mu}_j, \Sigma_j, \pi_j)$. Then, for the parameters of the model, we end up with $\Theta = (\theta_1, \ldots, \theta_k)^T$. The log likelihood of the data X given the parameters Θ is:

$$l(X|\Theta) = \log \prod_{i=1}^{n} P(\vec{x}_i) = \log \prod_{i=1}^{n} \sum_{j=1}^{k} \pi_j \mathrm{n}_j(\vec{x}_i;\ \vec{\mu}_j, \Sigma_j) = \sum_{i=1}^{n} \log \sum_{j=1}^{k} \pi_j \mathrm{n}_j(\vec{x}_i;\ \vec{\mu}_j, \Sigma_j) \tag{14.19}$$

The set of parameters Θ with the maximum likelihood gives us the best model of the data (assuming that it was generated by a mixture of k Gaussians). So our goal is to find parameters Θ that maximize the log likelihood given in the equation above. Here, we have to fiddle with all the parameters so as to try to make the likelihood of each data point a maximum, while still observing various constraints on the values of the parameters (so that the area under the pdf remains 1, for instance). This is a nasty problem in constrained optimization. We cannot calculate the maximum directly since it involves the log of a sum. Instead, we approximate the solution by iteration using the EM algorithm.

The EM algorithm is an iterative solution to the following circular statements:

Estimate: If we knew the value of Θ we could compute the expected values of the hidden structure of the model.

Maximize: If we knew the expected values of the hidden structure of the model, then we could compute the maximum likelihood value of Θ.

EXPECTATION STEP
MAXIMIZATION STEP

We break the circularity by beginning with a guess for Θ and iterating back and forth between an *expectation step* and a *maximization step*, hence the name EM algorithm. In the expectation step, we compute expected values for the hidden variables z_{ij} which can be interpreted as cluster membership probabilities. Given the current parameters, we compute how likely it is that an object belongs to any of the clusters. The maximization step computes the most likely parameters of the model given the cluster membership probabilities. This procedure improves our estimates of the parameters so that the parameters of a cluster better reflect the properties of objects with high probability of membership in it.

A key property of the EM algorithm is monotonicity: With each iteration of E and M steps, the likelihood of the model given the data increases. This guarantees that each iteration produces model parameters that are more likely given the data we see. However, while the algorithm will eventually move to a local maximum, it will often not find the globally best solution. This is an important difference from least-squares methods like SVD (covered in chapter 15), which are guaranteed to find the global optimum.

In what follows we describe the EM algorithm for estimating a Gaussian mixture. We follow here the discussion in (Dempster et al. 1977), (Mitchell 1997: ch. 6), and (Ghahramani 1994). See also (Duda and Hart 1973: 193).

To begin, we **initialize** all the parameters. Here, it would be appropriate to initialize the covariance matrices Σ_j of each Gaussian as an identity matrix, each of the weights π_j as $\frac{1}{k}$, and the k means $\vec{\mu}_j$ are each chosen to be a random perturbation away from a data point randomly selected from $\mathcal{X}$.

The **E-step** is the computation of parameters h_{ij}. h_{ij} is the expectation of the hidden variable z_{ij} which is 1 if n_j generated $\vec{x}_i$ and 0 otherwise.

$$h_{ij} = E(z_{ij}|\vec{x}_i;\, \Theta) = \frac{P(\vec{x}_i|\mathrm{n}_j;\, \Theta)}{\sum_{l=1}^{k} P(\vec{x}_i|\mathrm{n}_l;\, \Theta)} \tag{14.20}$$

The **M-step** is to recompute the parameters Θ (mean, variance, and prior for each Gaussian) as maximum likelihood estimates given expected values h_{ij}:

$$\vec{\mu}_j' = \frac{\sum_{i=1}^{n} h_{ij}\vec{x}_i}{\sum_{i=1}^{n} h_{ij}} \tag{14.21}$$

$$\Sigma_j' = \frac{\sum_{i=1}^{n} h_{ij}(\vec{x}_i - \vec{\mu}_j')(\vec{x}_i - \vec{\mu}_j')^T}{\sum_{i=1}^{n} h_{ij}} \tag{14.22}$$

These are the maximum-likelihood estimates for the mean and variance of a Gaussian (Duda and Hart 1973: 23).

The weights of the Gaussians are recomputed as:

$$\pi_j' = \frac{\sum_{i=1}^{n} h_{ij}}{\sum_{j=1}^{k}\sum_{i=1}^{n} h_{ij}} = \frac{\sum_{i=1}^{n} h_{ij}}{n} \tag{14.23}$$

Once means, variances, and priors have been recomputed, we **repeat**, by performing the next iteration of the E and M steps. We keep iterating as long as the log likelihood keeps improving significantly at each step.

EM and its applications in Statistical NLP

This description of the EM algorithm has been for clustering and Gaussian mixtures, but the reader should be able to recognize other applications of EM as instantiations of the same general scheme, for example, the forward-backward algorithm and the inside-outside algorithm, which we covered in earlier chapters. Here is the most general formulation of EM (Dempster et al. 1977: 6).

Define a function Q as follows:

$$Q(\Theta|\Theta^k) = E(l(X,Z)|\Theta)|X,\Theta^k)$$

Here, the z are the hidden variables (the vectors $z_{i1},\ldots,z_{iM}$ above), the x are the observed data (the vectors $\vec{x}_i$ above). Θ are the parameters of the model, the mean, variances and priors in the case of Gaussian mixtures. $l((x,z)|\Theta)$ is (the log of) the joint probability distribution of observable and unobservable data given the parameters Θ.

Then the E and M step take the following form:

- E-step: Compute $Q(\Theta|\Theta^k)$.
- M-step: Choose Θ^{k+1} to be a value of Θ that maximizes $Q(\Theta|\Theta^k)$.

For the Gaussian mixture case, computing Q corresponds to computing the h_{ij}, the expected values of the hidden variable z. The M step chooses the parameters Θ that maximize $Q(\Theta|\Theta^k)$.

This general formulation of EM is not literally an algorithm. We are not told how to compute the M-step in general. (There are cases where it cannot be computed.) However, for a large class of problems there exist such algorithms, for example, for all distributions of the exponential family, of which the Gaussian distribution is an example. The remainder of this section briefly discusses how certain other algorithms covered in this book are instances of the EM algorithm.

Baum-Welch reestimation. In the Baum-Welch or forward-backward algorithm (see section 9.3.3), the E step computes (i) for each state i, the expected number of transitions from i in the observed data; (ii) for each pair (i, j) of states, the expected number of transitions from state i to state j. The unobservable data here are the unobservable state transitions. In the E step, we compute expected values for these unobservables given the current parameters of the model.

The M step computes new maximum likelihood estimates of the parameters given the expected values for the unobservables. For Baum-Welch, these parameters are the initial state probabilities π_i, the state transition probabilities a_{ij} and the symbol emission probabilities b_{ijk}.

Inside-outside algorithm. The unobservable data in this algorithm (see section 11.3.4) are whether a particular rule $N^j \rightarrow \zeta$ is used to generate a particular subsequence w_{pq} of words or not. The E step computes expectations over these data, corresponding to the expected number of times that a particular rule will be used. We use the symbols $u_i(p, q, j, r, s)$ and $v_i(p, q, j)$ for these expectations in section 11.3.4. (The difference between the u_i and the v_i is that the u_i are for rules that produce nonterminals and the v_i are for rules that produce preterminals. The subscript i refers to sentence i in the training set.)

The M step then computes maximum-likelihood estimates of the parameters based on the f_i and g_i. The parameters here are the rule probabilities and maximum-likelihood simply consists of summing and renormalizing the f_i or g_i for a particular nonterminal.

Unsupervised word sense disambiguation. The unsupervised word sense disambiguation algorithm in section 7.4 is a clustering algorithm very similar to EM estimation of Gaussian mixtures, except that the probability model is different. The E step again computes expectations of hidden binary variables z_{ij} that record cluster memberships, but the probabilities are computed based on the Bayesian independence model described in that section, not a Gaussian mixture. The probability that a cluster (or sense, which is what we interpret each cluster as) generates a particular word is then recomputed in the M step as a maximum-likelihood estimate given the expectation values.

K-means. K-means can be interpreted as a special case of estimating a Gaussian mixture with EM. To see this, assume that we only recompute the mean of each Gaussian in each iteration of the algorithm. Priors and variances are fixed. If we fix the variances to be very small, then the shape of the Gaussian will be that of a sharp peak that falls off steeply from the center. As a result, if we look at the probability $P(\mathrm{n}_j|\vec{x}_i)$ of the 'best' cluster j and the probability $P(\mathrm{n}_{j'}|\vec{x}_i)$ of the next best cluster j', then the first will be much larger than the second.

This means that based on the posterior probabilities computed in the E step, each object will be a member of one cluster with probability very close to 1.0. In other words, we have an assignment that is a hard assignment for the purpose of recomputing the means since the contributions of objects with a very small membership probability will have a negligible influence on the computation of the means.

So an EM estimation of a Gaussian mixture with fixed small variances is very similar to K-means. However, there is a difference for ties. Even in the case of small variances, tied objects will have equal probabilities of membership for two clusters. In contrast, K-means makes a hard choice for ties.

Summary. The EM algorithm is very useful, and is currently very popular, but it is sensible to also be aware of its deficiencies. On the down-side, the algorithm is very sensitive to the initialization of the parameters, and unless parameters are initialized well, the algorithm usually gets stuck in one of the many local maxima that exist in the space. One possibility that is sometimes used is to use the results of another clustering algorithm to initialize the parameters for the EM algorithm. For instance, the K-means algorithm is an effective way of finding initial estimates for the cluster centers for EM of Gaussian mixtures. The rate of convergence of the EM algorithm can also be very slow. While reestimation via the EM algorithm is guaranteed to improve (or at least to not have a detrimental effect on) the likelihood of the data according to the model, it is also important to remember that it isn't guaranteed to improve other things that aren't actually in the model, such as the ability of a system to assign part of speech tags according to some external set of rules. Here there is a mismatch between what the EM algorithm is maximizing and the objective function on which the performance of the system is being evaluated, and, not surprisingly, in such circumstances the EM algorithm might have deleterious effects. Finally, it is perhaps worth pointing out that the EM algorithm is only really called for when there isn't a more straightforward way of solving the constrained optimization problem at hand. In simple cases where there is an algebraic solution or the solution can be found by a simple iterative equation solver such as Newton's method, then one may as well do that.

14.3 Further Reading

General introductions to clustering are (Kaufman and Rousseeuw 1990) and (Jain and Dubes 1988). Overviews of work on clustering in information retrieval can be found in (van Rijsbergen 1979), (Rasmussen 1992) and (Willett 1988).

MINIMUM SPANNING TREE

Algorithms for constructing *minimum spanning trees* can be found in (Cormen et al. 1990: ch. 24). For the general case, these algorithms run in $O(n \log n)$ where n is the number of nodes in the graph.

PRINCIPAL COMPONENT ANALYSIS

To the extent that clustering is used for data analysis and comprehension, it is closely related to visualization techniques that project a high-dimensional space onto (usually) two or three dimensions. Three commonly used techniques are *principal component analysis* (PCA) (see (Biber et al. 1998) for its application to corpora), Multi-Dimensional Scaling (MDS) (Kruskal 1964a,b) and Kohonen maps or Self-Organizing Maps (SOM) (Kohonen 1997). These spatial representations of a multi-dimensional space are alternatives to the dendrogram in figure 14.1.

Clustering can also be viewed as a form of category induction in cognitive modeling. Many researchers have attempted to induce syntactic categories by clustering corpus-derived word representations (Brill et al. 1990; Finch 1993). We used the clustering method proposed by Schütze (1995) for figure 14.1. Waterman (1995) attempts to find clusters of semantically related words based on syntactic evidence.

An early influential paper in which object nouns are clustered on the basis of verbs in a way similar to (Pereira et al. 1993) is (Hindle 1990). Li and Abe (1998) develop a symmetric model of verb-object pair clustering in which clusters generate both nouns and verbs. A pure verb clustering approach is adopted by Basili et al. (1996). An example of adjective clustering can be found in (Hatzivassiloglou and McKeown 1993).

The efficiency of clustering algorithms is becoming more important as text collections and NLP data sets increase in size. The Buckshot algorithm was proposed by Cutting et al. (1992). Even more efficient constant-time algorithms (based on precomputation of a cluster hierarchy) are described by Cutting et al. (1993) and Silverstein and Pedersen (1997).

14.4 Exercises

Exercise 14.1 [⋆⋆]

Retrieve the word space data from the website and do a single-link, complete-link, and group-average clustering of a subset.

Exercise 14.2 [⋆]

Construct an example data set for which the K-means algorithm takes more than one iteration to converge.

Exercise 14.3 [⋆]

Create a data set with 10 points in a plane where each point is a distance of 1 or less from the origin. Place an eleventh point (the outlier) in turn at distance (a) 2, (b) 4, (c) 8, and (d) 16 from the origin. For each of these cases run K-means clustering with two clusters (i) in Euclidean space and (ii) in L_1 space. (Pick the initial two centers from the 10 points near the origin.) Do the two distance measures give different results? What are the implications for data sets with outliers?

Exercise 14.4 [⋆⋆]

Since the EM algorithm only finds a local minimum, different starting conditions will lead to different clusterings. Run the EM algorithm 10 times with different initial seeds on the 1000 most frequent words from the website and analyze the differences. Compute the percentage of pairs of words that are in the same cluster in all 10 clusterings, in 9 clusterings, etc.

Exercise 14.5 [⋆]

Discuss the trade-off between time and the quality of clustering that needs to be made when choosing one of the three agglomerative algorithms or K-means.

Exercise 14.6 [⋆]

The model proposed by Brown et al. (1992c) is optimal in that it finds clusters that improve the evaluation measure directly. But it actually did not improve the measure in their experiment, or only after linear interpolation. What are possible reasons?

Exercise 14.7 [⋆]

Show that K-means converges if there are no ties. Compute as a goodness measure of the clustering the sum squared error that is incurred when each object is replaced by its cluster's center. Then show that this goodness measure decreases (or stays the same) in both the reassignment and the recomputation steps.