7 Clustering

In the parametric approach, we assumed that the sample comes from a known distribution. In cases when such an assumption is untenable, we relax this assumption and use a semiparametric approach that allows a mixture of distributions to be used for estimating the input sample. Clustering methods allow learning the mixture parameters from data. In addition to probabilistic modeling, we discuss vector quantization, spectral clustering, and hierarchical clustering.

7.1 Introduction

In chapters 4 and 5, we discussed the parametric method for density estimation where we assumed that the sample $\mathcal{X}$ is drawn from some parametric family, for example, Gaussian. In parametric classification, this corresponds to assuming a certain density for the class densities $p(\boldsymbol{x}|C_i)$. The advantage of any parametric approach is that given a model, the problem reduces to the estimation of a small number of parameters, which, in the case of density estimation, are the sufficient statistics of the density, for example, the mean and covariance in the case of Gaussian densities.

Though parametric approaches are used quite frequently, assuming a rigid parametric model may be a source of bias in many applications where this assumption does not hold. We thus need more flexible models. In particular, assuming Gaussian density corresponds to assuming that the sample, for example, instances of a class, forms one single group in the d-dimensional space, and as we saw in chapter 5, the center and the shape of this group is given by the mean and the covariance respectively.

In many applications, however, the sample is not one group; there may be several groups. Consider the case of optical character recognition: There are two ways of writing the digit 7; the American writing is '7', whereas the European writing style has a horizontal bar in the middle (to tell it apart from the European '1', which keeps the small stroke on top in handwriting). In such a case, when the sample contains examples from both continents, the class for the digit 7 should be represented as the disjunction of two groups. If each of these groups can be represented by a Gaussian, the class can be represented by a *mixture* of two Gaussians, one for each writing style.

A similar example is in speech recognition where the same word can be uttered in different ways, due to different pronounciation, accent, gender, age, and so forth. Thus when there is not a single, universal prototype, all these different ways should be represented in the density to be statistically correct.

SEMIPARAMETRIC DENSITY ESTIMATION

We call this approach *semiparametric density estimation*, as we still assume a parametric model for each group in the sample. We discuss the *nonparametric* approach in chapter 8, which is used when there is no structure to the data and even a mixture model is not applicable. In this chapter, we focus on density estimation and defer supervised learning to chapter 12.

7.2 Mixture Densities

MIXTURE DENSITY

The *mixture density* is written as

$$p(\boldsymbol{x}) = \sum_{i=1}^{k} p(\boldsymbol{x}|\mathcal{G}_i)P(\mathcal{G}_i) \tag{7.1}$$

MIXTURE COMPONENTS
GROUPS
CLUSTERS
COMPONENT DENSITIES
MIXTURE PROPORTIONS

where $\mathcal{G}_i$ are the *mixture components*. They are also called *group* or *clusters*. $p(\boldsymbol{x}|\mathcal{G}_i)$ are the *component densities* and $P(\mathcal{G}_i)$ are the *mixture proportions*. The number of components, k, is a hyperparameter and should be specified beforehand. Given a sample and k, learning corresponds to estimating the component densities and proportions. When we assume that the component densities obey a parametric model, we need only estimate their parameters. If the component densities are multivariate Gaussian, we have $p(\boldsymbol{x}|\mathcal{G}_i) \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, and $\Phi = \{P(\mathcal{G}_i), \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}_{i=1}^{k}$ are the parameters that should be estimated from the iid sample $\mathcal{X} = \{\boldsymbol{x}^t\}_t$.

Parametric classification is a bona fide mixture model where groups, $\mathcal{G}_i$, correspond to classes, C_i, component densities $p(\boldsymbol{x}|\mathcal{G}_i)$ correspond to class densities $p(\boldsymbol{x}|C_i)$, and $P(\mathcal{G}_i)$ correspond to class priors, $P(C_i)$:

$$p(\boldsymbol{x}) = \sum_{i=1}^{K} p(\boldsymbol{x}|C_i)P(C_i)$$

In this *supervised* case, we know how many groups there are and learning the parameters is trivial because we are given the labels, namely, which instance belongs to which class (component). We remember from chapter 5 that when we are given the sample $\mathcal{X} = \{\boldsymbol{x}^t, \boldsymbol{r}^t\}_{t=1}^N$, where $r_i^t = 1$ if $\boldsymbol{x}^t \in C_i$ and 0 otherwise, the parameters can be calculated using maximum likelihood. When each class is Gaussian distributed, we have a Gaussian mixture, and the parameters are estimated as

$$\begin{aligned} \hat{P}(C_i) &= \frac{\sum_t r_i^t}{N} \\ \boldsymbol{m}_i &= \frac{\sum_t r_i^t \boldsymbol{x}^t}{\sum_t r_i^t} \\ \mathbf{S}_i &= \frac{\sum_t r_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i)(\boldsymbol{x}^t - \boldsymbol{m}_i)^T}{\sum_t r_i^t} \end{aligned} \tag{7.2}$$

The difference in this chapter is that the sample is $\mathcal{X} = \{\boldsymbol{x}^t\}_t$: We have an *unsupervised learning* problem. We are given only $\boldsymbol{x}^t$ and not the labels $\boldsymbol{r}^t$, that is, we do not know which $\boldsymbol{x}^t$ comes from which component. So we should estimate both: First, we should estimate the labels, r_i^t, the component that a given instance belongs to; and, second, once we estimate the labels, we should estimate the parameters of the components given the set of instances belonging to them. We first discuss a simple algorithm, k-means clustering, for this purpose and later on show that it is a special case of the *expectation-maximization* (EM) algorithm.

7.3 *k*-Means Clustering

Let us say we have an image that is stored with 24 bits/pixel and can have up to 16 million colors. Assume we have a color screen with 8 bits/pixel that can display only 256 colors. We want to find the best 256 colors among all 16 million colors such that the image using only the 256 colors in the palette looks as close as possible to the original image. This is *color quantization* where we map from high to lower resolution. In the general

COLOR QUANTIZATION

case, the aim is to map from a continuous space to a discrete space; this process is called *vector quantization*.

VECTOR QUANTIZATION

Of course we can always quantize uniformly, but this wastes the colormap by assigning entries to colors not existing in the image, or would not assign extra entries to colors frequently used in the image. For example, if the image is a seascape, we expect to see many shades of blue and maybe no red. So the distribution of the colormap entries should reflect the original density as close as possible placing many entries in high-density regions, discarding regions where there is no data.

REFERENCE VECTORS

Let us say we have a sample of $\mathcal{X} = \{\boldsymbol{x}^t\}_{t=1}^N$. We have k *reference vectors*, $\boldsymbol{m}_j, j = 1, \ldots, k$. In our example of color quantization, $\boldsymbol{x}^t$ are the image pixel values in 24 bits and $\boldsymbol{m}_j$ are the color map entries also in 24 bits, with $k = 256$.

Assume for now that we somehow have the $\boldsymbol{m}_j$ values; we discuss how to learn them shortly. Then in displaying the image, given the pixel, $\boldsymbol{x}^t$, we represent it with the most similar entry, $\boldsymbol{m}_i$ in the color map, satisfying

$$\|\boldsymbol{x}^t - \boldsymbol{m}_i\| = \min_j \|\boldsymbol{x}^t - \boldsymbol{m}_j\|$$

CODEBOOK VECTORS

CODE WORDS

COMPRESSION

That is, instead of the original data value, we use the closest value we have in the alphabet of reference vectors. $\boldsymbol{m}_i$ are also called *codebook vectors* or *code words*, because this is a process of *encoding/decoding* (see figure 7.1): Going from $\boldsymbol{x}^t$ to i is a process of encoding the data using the codebook of $\boldsymbol{m}_i, i = 1, \ldots, k$ and, on the receiving end, generating $\boldsymbol{m}_i$ from i is decoding. Quantization also allows *compression*: For example, instead of using 24 bits to store (or transfer over a communication line) each $\boldsymbol{x}^t$, we can just store/transfer its index i in the colormap using 8 bits to index any one of 256, and we get a compression rate of almost 3; there is also the color map to store/transfer.

RECONSTRUCTION ERROR

Let us see how we can calculate $\boldsymbol{m}_i$: When $\boldsymbol{x}^t$ is represented by $\boldsymbol{m}_i$, there is an error that is proportional to the distance, $\|\boldsymbol{x}^t - \boldsymbol{m}_i\|$. For the new image to look like the original image, we should have these distances as small as possible for all pixels. The total *reconstruction error* is defined as

$$E(\{\boldsymbol{m}_i\}_{i=1}^k|\mathcal{X}) = \sum_t \sum_i b_i^t \|\boldsymbol{x}^t - \boldsymbol{m}_i\|^2 \tag{7.3}$$

where

$$b_i^t = \begin{cases} 1 & \text{if } \|\boldsymbol{x}^t - \boldsymbol{m}_i\| = \min_j \|\boldsymbol{x}^t - \boldsymbol{m}_j\| \\ 0 & \text{otherwise} \end{cases} \tag{7.4}$$

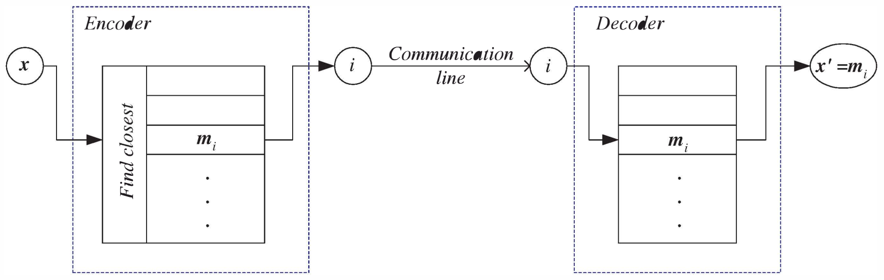

Figure 7.1 Given $\boldsymbol{x}$, the encoder sends the index of the closest code word and the decoder generates the code word with the received index as $\boldsymbol{x}'$. Error is $\|\boldsymbol{x}' - \boldsymbol{x}\|^2$.

The best reference vectors are those that minimize the total reconstruction error. b_i^t also depend on $\boldsymbol{m}_i$, and we cannot solve this optimization problem analytically. We have an iterative procedure named *k-means clustering* for this: First, we start with some $\boldsymbol{m}_i$ initialized randomly. Then at each iteration, we first use equation 7.4 and calculate b_i^t for all $\boldsymbol{x}^t$, which are the *estimated labels*; if b_i^t is 1, we say that $\boldsymbol{x}^t$ belongs to the group of $\boldsymbol{m}_i$. Then, once we have these labels, we minimize equation 7.3. Taking its derivative with respect to $\boldsymbol{m}_i$ and setting it to 0, we get

k-MEANS CLUSTERING

$$\boldsymbol{m}_i = \frac{\sum_t b_i^t \boldsymbol{x}^t}{\sum_t b_i^t} \tag{7.5}$$

The reference vector is set to the mean of all the instances that it represents. Note that this is the same as the formula for the mean in equation 7.2, except that we place the estimated labels b_i^t in place of the labels r_i^t. This is an iterative procedure because once we calculate the new $\boldsymbol{m}_i$, b_i^t change and need to be recalculated, which in turn affect $\boldsymbol{m}_i$. These two steps are repeated until $\boldsymbol{m}_i$ stabilize (see figure 7.2). The pseudocode of the *k*-means algorithm is given in figure 7.3.

One disadvantage is that this is a local search procedure, and the final $\boldsymbol{m}_i$ highly depend on the initial $\boldsymbol{m}_i$. There are various methods for initialization:

- We can simply take randomly selected k instances as the initial $\boldsymbol{m}_i$.
- The mean of all data can be calculated and small random vectors may be added to the mean to get the k initial $\boldsymbol{m}_i$.

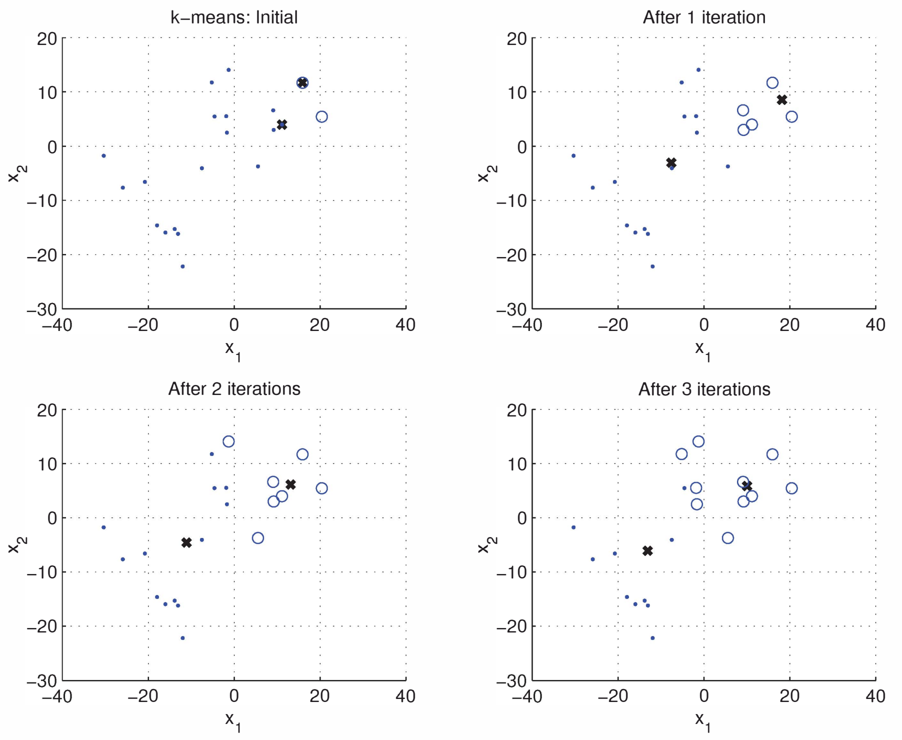

Figure 7.2 Evolution of k-means. Crosses indicate center positions. Data points are marked depending on the closest center.

- We can calculate the principal component, divide its range into k equal intervals, partitioning the data into k groups, and then take the means of these groups as the initial centers.

After convergence, all the centers should cover some subset of the data instances and be useful; therefore, it is best to initialize centers where we believe there is data.

There are also algorithms for adding new centers *incrementally* or deleting empty ones. In *leader cluster algorithm*, an instance that is far away from existing centers (defined by a threshold value) causes the creation of a new center at that point (we discuss such a neural network algorithm, ART, in chapter 12). Or, a center that covers a large number of instances ($\sum_t b_i^t / N > \theta$) can be split into two (by adding a small random vector to

LEADER CLUSTER ALGORITHM

```
Initialize $\boldsymbol{m}_i, i = 1, \ldots, k$, for example, to $k$ random $\boldsymbol{x}^t$
Repeat
    For all $\boldsymbol{x}^t \in \mathcal{X}$
        $b_i^t \leftarrow \begin{cases} 1 & \text{if } \|\boldsymbol{x}^t - \boldsymbol{m}_i\| = \min_j \|\boldsymbol{x}^t - \boldsymbol{m}_j\| \\ 0 & \text{otherwise} \end{cases}$
    For all $\boldsymbol{m}_i, i = 1, \ldots, k$
        $\boldsymbol{m}_i \leftarrow \sum_t b_i^t \boldsymbol{x}^t / \sum_t b_i^t$
Until $\boldsymbol{m}_i$ converge
```

Figure 7.3 k-means algorithm.

one of the two copies to make them different). Similarly, a center that covers too few instances can be removed and restarted from some other part of the input space.

k-means algorithm is for clustering, that is, for finding groups in the data, where the groups are represented by their centers, which are the typical representatives of the groups. Vector quantization is one application of clustering, but clustering is also used for *preprocessing* before a later stage of classification or regression. Given $\boldsymbol{x}^t$, when we calculate b_i^t, we do a mapping from the original space to the k-dimensional space, that is, to one of the corners of the k-dimensional hypercube. A regression or discriminant function can then be learned in this new space; we discuss such methods in chapter 12.

7.4 Expectation-Maximization Algorithm

In k-means, we approached clustering as the problem of finding codebook vectors that minimize the total reconstruction error. In this section, our approach is probabilistic and we look for the component density parameters that maximize the likelihood of the sample. Using the mixture model of equation 7.1, the log likelihood given the sample $\mathcal{X} = \{\boldsymbol{x}^t\}_t$ is

$$\begin{aligned} \mathcal{L}(\Phi|\mathcal{X}) &= \log \prod_t p(\boldsymbol{x}^t|\Phi) \\ &= \sum_t \log \sum_{i=1}^{k} p(\boldsymbol{x}^t|\mathcal{G}_i)P(\mathcal{G}_i) \end{aligned} \tag{7.6}$$

where Φ includes the priors $P(\mathcal{G}_i)$ and also the sufficient statistics of the component densities $p(\boldsymbol{x}^t|\mathcal{G}_i)$. Unfortunately, we cannot solve for the parameters analytically and need to resort to iterative optimization.

EXPECTATION-MAXIMIZATION

The *expectation-maximization* algorithm (Dempster, Laird, and Rubin 1977; Redner and Walker 1984) is used in maximum likelihood estimation where the problem involves two sets of random variables of which one, X, is observable and the other, Z, is hidden. The goal of the algorithm is to find the parameter vector Φ that maximizes the likelihood of the observed values of X, $\mathcal{L}(\Phi|\mathcal{X})$. But in cases where this is not feasible, we associate the extra *hidden variables* Z and express the underlying model using both, to maximize the likelihood of the joint distribution of X and Z, the *complete* likelihood $\mathcal{L}_c(\Phi|\mathcal{X},\mathcal{Z})$.

Since the Z values are not observed, we cannot work directly with the complete data likelihood $\mathcal{L}_c$; instead, we work with its expectation, $\mathcal{Q}$, given $\mathcal{X}$ and the current parameter values Φ^l, where l indexes iteration. This is the *expectation* (E) step of the algorithm. Then in the *maximization* (M) step, we look for the new parameter values, Φ^{l+1}, that maximize this. Thus

$$\begin{aligned}
\text{E-step} &: \quad \mathcal{Q}(\Phi|\Phi^l) = E[\mathcal{L}_c(\Phi|\mathcal{X},\mathcal{Z})|\mathcal{X},\Phi^l] \\
\text{M-step} &: \quad \Phi^{l+1} = \arg\max_{\Phi} \mathcal{Q}(\Phi|\Phi^l)
\end{aligned}$$

Dempster, Laird, and Rubin (1977) proved that an increase in $\mathcal{Q}$ implies an increase in the incomplete likelihood

$$\mathcal{L}(\Phi^{l+1}|\mathcal{X}) \geq \mathcal{L}(\Phi^l|\mathcal{X})$$

In the case of mixtures, the hidden variables are the sources of observations, namely, which observation belongs to which component. If these were given, for example, as class labels in a supervised setting, we would know which parameters to adjust to fit that data point. The EM algorithm works as follows: in the E-step we estimate these labels given our current knowledge of components, and in the M-step we update our component knowledge given the labels estimated in the E-step. These two steps are the same as the two steps of k-means; calculation of b_i^t (E-step) and reestimation of $\boldsymbol{m}_i$ (M-step).

We define a vector of *indicator variables* $\boldsymbol{z}^t = \{z_1^t, \ldots, z_k^t\}$ where $z_i^t = 1$ if $\boldsymbol{x}^t$ belongs to cluster $\mathcal{G}_i$, and 0 otherwise. $\boldsymbol{z}$ is a multinomial distribution from k categories with prior probabilities π_i, shorthand for $P(\mathcal{G}_i)$.

Then

(7.7) $$P(\mathbf{z}^t) = \prod_{i=1}^{k} \pi_i^{z_i^t}$$

The likelihood of an observation $\mathbf{x}^t$ is equal to its probability specified by the component that generated it:

(7.8) $$p(\mathbf{x}^t|\mathbf{z}^t) = \prod_{i=1}^{k} p_i(\mathbf{x}^t)^{z_i^t}$$

$p_i(\mathbf{x}^t)$ is shorthand for $p(\mathbf{x}^t|\mathcal{G}_i)$. The joint density is

$$p(\mathbf{x}^t, \mathbf{z}^t) = P(\mathbf{z}^t)p(\mathbf{x}^t|\mathbf{z}^t)$$

and the complete data likelihood of the iid sample $\mathcal{X}$ is

$$\begin{aligned}
\mathcal{L}_C(\Phi|\mathcal{X},\mathcal{Z}) &= \log \prod_t p(\mathbf{x}^t, \mathbf{z}^t|\Phi) \\
&= \sum_t \log p(\mathbf{x}^t, \mathbf{z}^t|\Phi) \\
&= \sum_t \log P(\mathbf{z}^t|\Phi) + \log p(\mathbf{x}^t|\mathbf{z}^t, \Phi) \\
&= \sum_t \sum_i z_i^t [\log \pi_i + \log p_i(\mathbf{x}^t|\Phi)]
\end{aligned}$$

E-step: We define

$$\begin{aligned}
\mathcal{Q}(\Phi|\Phi^l) &\equiv E\left[\log P(X, Z)|\mathcal{X}, \Phi^l\right] \\
&= E\left[\mathcal{L}_C(\Phi|\mathcal{X},\mathcal{Z})|\mathcal{X}, \Phi^l)\right] \\
&= \sum_t \sum_i E[z_i^t|\mathcal{X}, \Phi^l][\log \pi_i + \log p_i(\mathbf{x}^t|\Phi^l)]
\end{aligned}$$

where

$$\begin{aligned}
E[z_i^t|\mathcal{X}, \Phi^l] &= E[z_i^t|\mathbf{x}^t, \Phi^l] \quad \mathbf{x}^t \text{ are iid} \\
&= P(z_i^t = 1|\mathbf{x}^t, \Phi^l) \quad z_i^t \text{ is a 0/1 random variable} \\
&= \frac{p(\mathbf{x}^t|z_i^t = 1, \Phi^l)P(z_i^t = 1|\Phi^l)}{p(\mathbf{x}^t|\Phi^l)} \quad \text{Bayes' rule} \\
&= \frac{p_i(\mathbf{x}^t|\Phi^l)\pi_i}{\sum_j p_j(\mathbf{x}^t|\Phi^l)\pi_j} \\
&= \frac{p(\mathbf{x}^t|\mathcal{G}_i, \Phi^l)P(\mathcal{G}_i)}{\sum_j p(\mathbf{x}^t|\mathcal{G}_j, \Phi^l)P(\mathcal{G}_j)} \\
&= P(\mathcal{G}_i|\mathbf{x}^t, \Phi^l) \equiv h_i^t
\end{aligned} \quad (7.9)$$

We see that the expected value of the hidden variable, $E[z_i^t]$, is the posterior probability that $\boldsymbol{x}^t$ is generated by component $\mathcal{G}_i$. Because this is a probability, it is between 0 and 1 and is a "soft" label, as opposed to the 0/1 "hard" label of k-means.

M-step: We maximize $\mathcal{Q}$ to get the next set of parameter values Φ^{l+1}:

$$\Phi^{l+1} = \arg\max_{\Phi} \mathcal{Q}(\Phi|\Phi^l)$$

which is

$$\begin{aligned} \mathcal{Q}(\Phi|\Phi^l) &= \sum_t \sum_i h_i^t [\log \pi_i + \log p_i(\boldsymbol{x}^t|\Phi^l)] \\ &= \sum_t \sum_i h_i^t \log \pi_i + \sum_t \sum_i h_i^t \log p_i(\boldsymbol{x}^t|\Phi^l) \end{aligned} \quad (7.10)$$

The second term is independent of π_i and using the constraint that $\sum_i \pi_i = 1$ as the Lagrangian, we solve for

$$\nabla_{\pi_i} \sum_t \sum_i h_i^t \log \pi_i - \lambda \left(\sum_i \pi_i - 1 \right) = 0$$

and get

$$\pi_i = \frac{\sum_t h_i^t}{N} \quad (7.11)$$

which is analogous to the calculation of priors in equation 7.2.

Similarly, the first term of equation 7.10 is independent of the components and can be dropped while estimating the parameters of the components. We solve for

$$\nabla_{\Phi} \sum_t \sum_i h_i^t \log p_i(\boldsymbol{x}^t|\Phi) = 0 \quad (7.12)$$

If we assume Gaussian components, $\hat{p}_i(\boldsymbol{x}^t|\Phi) \sim \mathcal{N}(\boldsymbol{m}_i, \mathbf{S}_i)$, the M-step is

$$\begin{aligned} \boldsymbol{m}_i^{l+1} &= \frac{\sum_t h_i^t \boldsymbol{x}^t}{\sum_t h_i^t} \\ \mathbf{S}_i^{l+1} &= \frac{\sum_t h_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i^{l+1})(\boldsymbol{x}^t - \boldsymbol{m}_i^{l+1})^T}{\sum_t h_i^t} \end{aligned} \quad (7.13)$$

where, for Gaussian components in the E-step, we calculate

$$h_i^t = \frac{\pi_i |\mathbf{S}_i|^{-1/2} \exp[-(1/2)(\boldsymbol{x}^t - \boldsymbol{m}_i)^T \mathbf{S}_i^{-1} (\boldsymbol{x}^t - \boldsymbol{m}_i)]}{\sum_j \pi_j |\mathbf{S}_j|^{-1/2} \exp[-(1/2)(\boldsymbol{x}^t - \boldsymbol{m}_j)^T \mathbf{S}_j^{-1} (\boldsymbol{x}^t - \boldsymbol{m}_j)]} \quad (7.14)$$

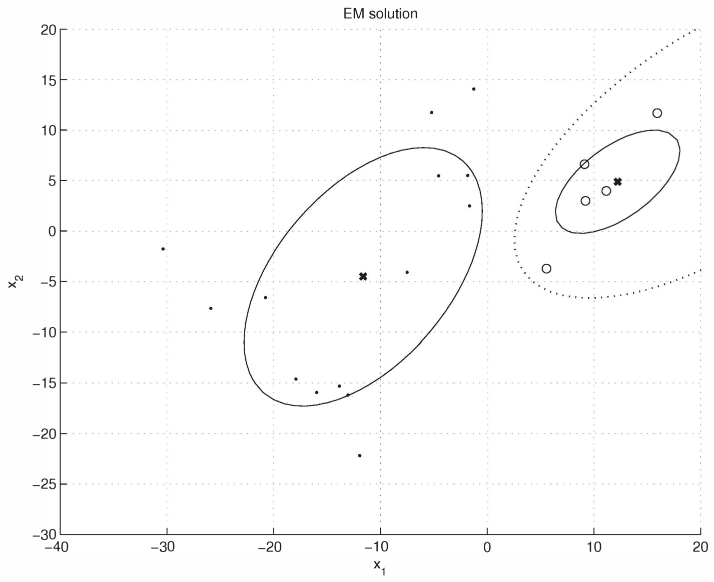

Figure 7.4 Data points and the fitted Gaussians by EM, initialized by one k-means iteration of figure 7.2. Unlike in k-means, as can be seen, EM allows estimating the covariance matrices. The data points labeled by greater h_i, the contours of the estimated Gaussian densities, and the separating curve of $h_i = 0.5$ (dashed line) are shown.

Again, the similarity between equations 7.13 and 7.2 is not accidental; the estimated soft labels h_i^t replace the actual (unknown) labels r_i^t.

EM is initalized by k-means. After a few iterations of k-means, we get the estimates for the centers $\boldsymbol{m}_i$, and using the instances covered by each center, we estimate the $\mathbf{S}_i$ and $\sum_t b_i^t / N$ give us the π_i. We run EM from that point on, as shown in figure 7.4.

Just as in parametric classification (section 5.5), with small samples and large dimensionality we can regularize by making simplifying assumptions. When $\hat{p}_i(\boldsymbol{x}^t|\Phi) \sim \mathcal{N}(\boldsymbol{m}_i, \mathbf{S})$, the case of a shared covariance matrix, equation 7.12 reduces to

$$\min_{\boldsymbol{m}_i, \mathbf{S}} \sum_t \sum_i h_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i)^T \mathbf{S}^{-1} (\boldsymbol{x}^t - \boldsymbol{m}_i) \tag{7.15}$$

When $\hat{p}_i(\boldsymbol{x}^t|\Phi) \sim \mathcal{N}(\boldsymbol{m}_i, s^2\mathbf{I})$, the case of a shared diagonal matrix, we have

$$\min_{\boldsymbol{m}_i,s} \sum_t \sum_i h_i^t \frac{\|\boldsymbol{x}^t - \boldsymbol{m}_i\|^2}{s^2} \tag{7.16}$$

which is the reconstruction error we defined in k-means clustering (equation 7.3). The difference is that now

$$h_i^t = \frac{\exp\left[-(1/2s^2)\|\boldsymbol{x}^t - \boldsymbol{m}_i\|^2\right]}{\sum_j \exp\left[-(1/2s^2)\|\boldsymbol{x}^t - \boldsymbol{m}_j\|^2\right]} \tag{7.17}$$

is a probability between 0 and 1. b_i^t of k-means clustering makes a hard 0/1 decision, whereas h_i^t is a *soft label* that assigns the input to a cluster with a certain probability. When h_i^t are used instead of b_i^t, an instance contributes to the update of parameters of all components, to each proportional to that probability. This is especially useful if the instance is close to the midpoint between two centers.

We thus see that k-means clustering is a special case of EM applied to Gaussian mixtures where inputs are assumed independent with equal and shared variances, all components have equal priors, and labels are hardened. k-means thus pave the input density with circles, whereas EM in the general case uses ellipses of arbitrary shapes, orientations, and coverage proportions.

7.5 Mixtures of Latent Variable Models

When full covariance matrices are used with Gaussian mixtures, even if there is no singularity, one risks overfitting if the input dimensionality is high and the sample is small. To decrease the number of parameters, assuming a common covariance matrix may not be right since clusters may really have different shapes. Assuming diagonal matrices is even more risky because it removes all correlations.

The alternative is to do dimensionality reduction in the clusters. This decreases the number of parameters while still capturing the correlations. The number of free parameters is controlled through the dimensionality of the reduced space.

When we do factor analysis (section 6.5) in the clusters, we look for *latent* or *hidden variables* or *factors* that generate the data in the clusters (Bishop 1999):

$$p(\boldsymbol{x}^t|\mathcal{G}_i) \sim \mathcal{N}(\boldsymbol{m}_i, \mathbf{V}_i\mathbf{V}_i^T + \boldsymbol{\Psi}_i) \tag{7.18}$$

where $\mathbf{V}_i$ and $\mathbf{\Psi}_i$ are the factor loadings and specific variances of cluster $\mathcal{G}_i$. Rubin and Thayer (1982) give EM equations for factor analysis. It is possible to extend this in mixture models to find *mixtures of factor analyzers* (Ghahramani and Hinton 1997). In the E-step, in equation 7.9, we use equation 7.18, and in the M-step, we solve equation 7.12 for $\mathbf{V}_i$ and $\mathbf{\Psi}_i$ instead of $\mathbf{S}_i$. Similarly, one can also do PCA in groups, which is called *mixtures of probabilistic principal component analyzers* (Tipping and Bishop 1999).

MIXTURES OF FACTOR ANALYZERS

MIXTURES OF PROBABILISTIC PRINCIPAL COMPONENT ANALYZERS

We can of course use EM to learn $\mathbf{S}_i$ and then do FA or PCA separately in each cluster, but doing EM is better because it couples these two steps of clustering and dimensionality reduction and does a soft partitioning. An instance contributes to the calculation of the latent variables of all groups, weighted by h_i^t.

7.6 Supervised Learning after Clustering

Clustering, just as the dimensionality reduction methods discussed in chapter 6, can be used for two purposes. First, it can be used for data exploration, to understand the structure of data. Second, it can be used to map data to a new space where supervised learning is easier.

Dimensionality reduction methods are used to find correlations between variables and thus group variables; clustering methods, on the other hand, are used to find similarities between instances and thus group instances. If such groups are found, these may be named (by application experts) and their attributes be defined. One can choose the group mean as the representative prototype of instances in the group, or the possible range of attributes can be written. This allows a simpler description of the data. For example, if the customers of a company seem to fall in one of k groups, called *segments*, customers being defined in terms of their demographic attributes and transactions with the company, then a better understanding of the customer base will be provided that will allow the company to provide different strategies for different types of customers; this is part of *customer relationship management* (CRM). Likewise, the company will also be able to develop strategies for those customers who do not fall in any large group, and who may require attention—for example, churning customers.

CUSTOMER SEGMENTATION

CUSTOMER RELATIONSHIP MANAGEMENT

Frequently, clustering is also used as a preprocessing stage. Just like the dimensionality reduction methods of chapter 6 allowed us to make a mapping to a new space, after clustering, we also map to a new k-dimensional space where the dimensions are h_i (or b_i at the risk of loss of information). In a supervised setting, we can then learn the discriminant or regression function in this new space. The difference from dimensionality reduction methods like PCA however is that k, the dimensionality of the new space, can be larger than d, the original dimensionality.

When we use a method like PCA, where the new dimensions are combinations of the original dimensions, to represent any instance in the new space, all dimensions contribute; that is, all z_j are nonzero. In the case of a method like clustering where the new dimensions are defined locally, there are many more new dimensions, b_j, but only one (or if we use h_j, few) of them have a nonzero value. In the former case, where there are few dimensions but all contribute, we have a *distributed representation*; in the latter case, where there are many dimensions but few contribute, we have a *local representation*.

DISTRIBUTED VS. LOCAL REPRESENTATION

One advantage of preceding a supervised learner with unsupervised clustering or dimensionality reduction is that the latter does not need labeled data. Labeling the data is costly. We can use a large amount of unlabeled data for learning the cluster parameters and then use a smaller labeled data to learn the second stage of classification or regression. Unsupervised learning is called "learning what normally happens" (Barrow 1989). When followed by a supervised learner, we first learn what normally happens and then learn what that means. We discuss such methods in chapter 12.

MIXTURE OF MIXTURES

In the case of classification, when each class is a mixture model composed of a number of components, the whole density is a *mixture of mixtures*:

$$\begin{aligned} p(\boldsymbol{x}|C_i) &= \sum_{j=1}^{k_i} p(\boldsymbol{x}|\mathcal{G}_{ij})P(\mathcal{G}_{ij}) \\ p(\boldsymbol{x}) &= \sum_{i=1}^{K} p(\boldsymbol{x}|C_i)P(C_i) \end{aligned}$$

where k_i is the number of components making up $p(\boldsymbol{x}|C_i)$ and $\mathcal{G}_{ij}$ is the component j of class i. Note that different classes may need different number of components. Learning the parameters of components is done separately for each class (probably after some regularization) as we discussed previously. This is better than fitting many components to data from all classes and then labeling them later with classes.

7.7 Spectral Clustering

Instead of clustering in the original space, a possibility is to first map the data to a new space with reduced dimensionality such that similarities are made more apparent and then cluster in there. Any feature selection or extraction method can be used for this purpose, and one such method is the Laplacian eigenmaps of section 6.12, where the aim is to place the data instances in such a way that given pairwise similarities are preserved.

After such a mapping, points that are similar are placed nearby, and this is expected to enhance the performance of clustering—for example, by using k-means. This is the idea behind *spectral clustering* (von Luxburg 2007). There are hence two steps:

SPECTRAL CLUSTERING

1. In the original space, we define a local neighborhood (by either fixing the number of neighbors or a distance threshold), and then for instances that are in the same neighborhood, we define a similarity measure—for example, using the Gaussian kernel—that is inversely proportional to the distance between them. Remember that instances not in the same local neighborhood are assigned a similarity of 0 and hence can be placed anywhere with respect to each other. Given this Laplacian, instances are positioned in the new space using feature embedding.

2. We run k-means clustering with the new data coordinates in this new space.

We remember from section 6.12 that when $\mathbf{B}$ is the matrix of pairwise similarities and $\mathbf{D}$ is the diagonal degree matrix with $d_i = \sum_j B_{ij}$ on the diagonals, the graph Laplacian is defined as

$$\mathbf{L} = \mathbf{D} - \mathbf{B}$$

This is the unnormalized Laplacian. There are two ways to normalize. One is closely related to the random walk (Shi and Malik 2000) and the other constructs a symmetric matrix (Ng, Jordan, and Weiss 2002). They may lead to better performance in clustering:

$$\begin{aligned}\mathbf{L}_{rw} &= \mathbf{I} - \mathbf{D}^{-1}\mathbf{B} \\ \mathbf{L}_{sym} &= \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{B}\mathbf{D}^{-1/2}\end{aligned}$$

It is always a good idea to do dimensionality reduction before clustering using Euclidean distance if there are redundant or correlated features. Using Laplacian eigenmaps makes more sense than multidimensional scaling proper or principal components analysis because those two check for the preservation of pairwise similarities between *all* pairs of instances whereas here with Laplacian eigenmaps, we care about preserving the similarity between neighboring instances only and in a manner that is inversely proportional to the distance between them. This has the effect that instances that are nearby in the original space, probably within the same cluster, will be placed very close in the new space, thus making the work of *k*-means easier, whereas those that are some distance away, probably belonging to different clusters, will be placed far apart. The graph should always be connected; that is, the local neighborhood should be large enough to connect clusters. Remember that the number of eigenvectors with eigenvalue 0 is the number of components and that it should be 1.

Note that though similarities are local, they propagate. Consider three instances, a, b, and c. Let us say a and b lie in the same neighborhood and so do b and c, but not a and c. Still, because a and b will be placed nearby and b and c will be placed nearby, a will lie close to c too, and they will probably be assigned to the same cluster. Consider now a and d that are not in the neighborhood with too many intermediate nodes between them; these two will not be placed nearby and it is very unlikely that they will be assigned to the same cluster.

Depending on which graph Laplacian is used and depending on the neighborhood size or the spread of the Gaussian, different results can be obtained, so one should always try for different parameters (von Luxburg 2009).

7.8 Hierarchical Clustering

We discussed clustering from a probabilistic point of view as fitting a mixture model to the data, or in terms of finding code words minimizing reconstruction error. There are also methods for clustering that use only similarities of instances, without any other requirement on the data; the aim is to find groups such that instances in a group are more similar to each other than instances in different groups. This is the approach taken by *hierarchical clustering*.

HIERARCHICAL CLUSTERING

This needs the use of a similarity, or equivalently a distance, measure defined between instances. Generally Euclidean distance is used, where we have to make sure that all attributes have the same scale. This is a special case of the *Minkowksi distance* with $p = 2$:

$$d_m(\boldsymbol{x}^r, \boldsymbol{x}^s) = \left[\sum_{j=1}^{d} (x_j^r - x_j^s)^p \right]^{1/p}$$

City-block distance is easier to calculate:

$$d_{cb}(\boldsymbol{x}^r, \boldsymbol{x}^s) = \sum_{j=1}^{d} |x_j^r - x_j^s|$$

AGGLOMERATIVE CLUSTERING

DIVISIVE CLUSTERING

An *agglomerative clustering* algorithm starts with N groups, each initially containing one training instance, merging similar groups to form larger groups, until there is a single one. A *divisive clustering* algorithm goes in the other direction, starting with a single group and dividing large groups into smaller groups, until each group contains a single instance.

SINGLE-LINK CLUSTERING

At each iteration of an agglomerative algorithm, we choose the two closest groups to merge. In *single-link clustering*, this distance is defined as the smallest distance between all possible pair of elements of the two groups:

(7.19) $$d(\mathcal{G}_i, \mathcal{G}_j) = \min_{\boldsymbol{x}^r \in \mathcal{G}_i, \boldsymbol{x}^s \in \mathcal{G}_j} d(\boldsymbol{x}^r, \boldsymbol{x}^s)$$

Consider a weighted, completely connected graph with nodes corresponding to instances and edges between nodes with weights equal to the distances between the instances. Then the single-link method corresponds to constructing the minimal spanning tree of this graph.

COMPLETE-LINK CLUSTERING

In *complete-link clustering*, the distance between two groups is taken as the largest distance between all possible pairs:

(7.20) $$d(\mathcal{G}_i, \mathcal{G}_j) = \max_{\boldsymbol{x}^r \in \mathcal{G}_i, \boldsymbol{x}^s \in \mathcal{G}_j} d(\boldsymbol{x}^r, \boldsymbol{x}^s)$$

These are the two most frequently used measures to choose the two closest groups to merge. Other possibilities are the average-link method that uses the average of distances between all pairs and the centroid distance that measures the distance between the centroids (means) of the two groups.

DENDROGRAM

Once an agglomerative method is run, the result is generally drawn as a hierarchical structure known as the *dendrogram*. This is a tree where

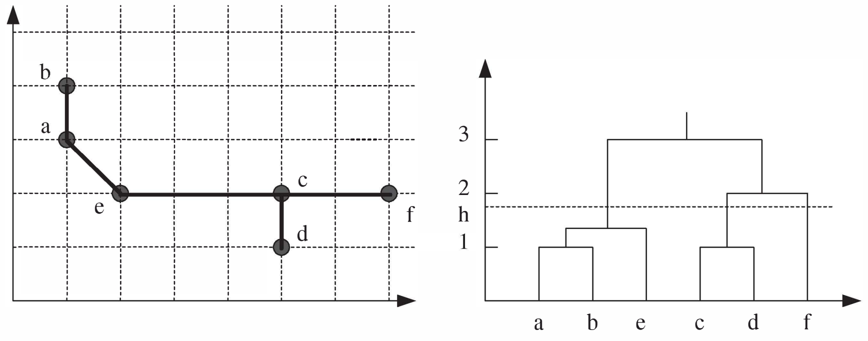

Figure 7.5 A two-dimensional dataset and the dendrogram showing the result of single-link clustering is shown. Note that leaves of the tree are ordered so that no branches cross. The tree is then intersected at a desired value of h to get the clusters.

leaves correspond to instances, which are grouped in the order in which they are merged. An example is given in figure 7.5. The tree can then be intersected at any level to get the wanted number of groups.

Single-link and complete-link methods calculate the distance between groups differently that affect the clusters and the dendrogram. In the single-link method, two instances are grouped together at level h if the distance between them is less than h, or if there is an intermediate sequence of instances between them such that the distance between consecutive instances is less than h. On the other hand, in the complete-link method, all instances in a group have a distance less than h between them. Single-link clusters may be elongated due to this "chaining" effect. (In figure 7.5, what if there were an instance halfway between e and c?) Complete-link clusters tend to be more compact.

7.9 Choosing the Number of Clusters

Like any learning method, clustering also has its knob to adjust complexity; it is k, the number of clusters. Given any k, clustering will always find k centers, whether they really are meaningful groups, or whether they are imposed by the method we use. There are various ways we can use to fine-tune k:

- In some applications such as color quantization, k is defined by the application.
- Plotting the data in two dimensions using PCA may be used in uncovering the structure of data and the number of clusters in the data.
- An incremental approach may also help: Setting a maximum allowed distance is equivalent to setting a maximum allowed reconstruction error per instance.
- In some applications, validation of the groups can be done manually by checking whether clusters actually code meaningful groups of the data. For example, in a data mining application, application experts may do this check. In color quantization, we may inspect the image visually to check its quality (despite the fact that our eyes and brain do not analyze an image pixel by pixel).

Depending on what type of clustering method we use, we can plot the reconstruction error or log likelihood as a function of k and look for the "elbow." After a large enough k, the algorithm will start dividing groups, in which case there will not be a large decrease in the reconstruction error or large increase in the log likelihood. Similarly, in hierarchical clustering, by looking at the differences between levels in the tree, we can decide on a good split.

7.10 Notes

Mixture models are frequently used in statistics. Dedicated textbooks are those by Titterington, Smith, and Makov (1985) and McLachlan and Basford (1988). McLachlan and Krishnan (1997) discuss recent developments in the EM algorithm, how its convergence can be accelerated, and various variants. In signal processing, k-means is called the *Linde-Buzo-Gray* (LBG) algorithm (Gersho and Gray 1992). It is used frequently in both statistics and signal processing in a large variety of applications and has many variants, one of which is *fuzzy k-means*. The *fuzzy membership* of an input to a component is also a number between 0 and 1 (Bezdek and Pal 1995). Alpaydın (1998) compares k-means, fuzzy k-means, and EM on Gaussian mixtures. A comparison of EM and other learning algorithms for the learning of Gaussian mixture models is given by Xu and Jordan (1996). On small data samples, an alternative to simplifying assumptions

FUZZY k-MEANS

is to use a Bayesian approach (Ormoneit and Tresp 1996). Moerland (1999) compares mixtures of Gaussians and mixtures of latent variable models on a set of classification problems, showing the advantage of latent variable models empirically. A book on clustering methods is by Jain and Dubes (1988) and survey articles are by Jain, Murty, and Flynn (1999) and Xu and Wunsch (2005).

One of the advantages of spectral clustering and hierarchical clustering is that we do not need a vectorial representation of the instances, as long as we can define a similarity/distance measure between pairs of instances. The problem of representing an arbitrary data structure—documents, graphs, web pages, and so on—as a vector such that Euclidean distance is meaningful is always a tedious task and leads to artificial representations, such as the bag of words. Being able to use (dis)similarity measures directly defined on the original structure is always a good idea, and we will have the same advantage with kernel functions when we talk about kernel machines in chapter 13.

7.11 Exercises

1. In image compression, k-means can be used as follows: The image is divided into nonoverlapping $c \times c$ windows and these c^2-dimensional vectors make up the sample. For a given k, which is generally a power of two, we do k-means clustering. The reference vectors and the indices for each window is sent over the communication line. At the receiving end, the image is then reconstructed by reading from the table of reference vectors using the indices. Write the computer program that does this for different values of k and c. For each case, calculate the reconstruction error and the compression rate.

2. We can do k-means clustering, partition the instances, and then calculate $\mathbf{S}_i$ separately in each group. Why is this not a good idea?

   SOLUTION: There are basically two reasons: First, k-means does hard partitioning but it is always better to do a soft partitioning (using $h_i^t \in (0,1)$ instead of $b_i^t \in \{0,1\}$) so that instances (in between two clusters) can contribute to the parameters (the covariance matrix in this case) of more than one cluster allowing a smooth transition between clusters.

   Second, k-means proper uses the Euclidean distance and we remember that Euclidean distance implies features that have the same scale and are independent. Using $\mathbf{S}_i$ implies the use of Mahalanobis distance and hence taking care of differences in scale and dependencies.

3. Derive the M-step equations for $\mathbf{S}$ in the case of shared arbitrary covariance

matrix $\mathbf{S}$ (equation 7.15) and s^2 in the case of shared diagonal covariance matrix (equation 7.16).

4. Define a multivariate Bernoulli mixture where inputs are binary and derive the EM equations.

   SOLUTION: When the components are multivariate Bernouilli, we have binary vectors that are d-dimensional. Assuming that the dimensions are independent, we have (see section 5.7)

$$p_i(\boldsymbol{x}^t|\Phi) = \prod_{j=1}^{d} p_{ij}^{x_j^t}(1-p_{ij})^{1-x_j^t}$$

   where $\Phi^l = \{p_{i1}^l, p_{i2}^l, \dots, p_{id}^l\}_{i=1}^k$. The E-step does not change (equation 7.9). In the M-step, for the component parameters $p_{ij}, i = 1, \dots, k, j = 1, \dots, d$, we maximize

$$\begin{aligned}\mathcal{Q}' &= \sum_t \sum_i h_i^t \log p_i(\boldsymbol{x}^t|\phi^l) \\ &= \sum_t \sum_i h_i^t \sum_j x_j^t \log p_{ij}^l + (1-x_j^t)\log(1-p_{ij}^l)\end{aligned}$$

   Taking the derivative with respect to p_{ij} and setting it equal to 0, we get

$$p_{ij}^{l+1} = \frac{\sum_t h_i^t x_j^t}{\sum_t h_j^t}$$

   Note that this is the same as in equation 5.31, except that estimated “soft” labels h_i^t replace the supervised labels r_i^t.

5. In the mixture of mixtures approach for classification, how can we fine-tune k_i, the number of components for class C_i?

EDIT DISTANCE

6. *Edit distance* between two strings—for example, gene sequences—is the number of character operations (insertions, deletions, substitutions) it takes to convert one string into another. List the advantages of doing spectral clustering using the edit distance as opposed to vanilla k-means using Euclidean distance on strings.

7. How can we do hierarchical clustering with binary input vectors—for example, for text clustering using the bag of words representation?

8. What are the similarities and differences between average-link clustering and k-means?

   SOLUTION: They both measure similarity by looking at the average of instances that fall in a cluster. Note, however, that in a hierarchical scheme, there are clusters at different resolutions.

9. In hierarchical clustering, how can we have locally adaptive distances? What are the advantages and disadvantages of this?

10. How can we make *k*-means robust to outliers?

    SOLUTION: An outlier is an instance that is very far from *all* centers. We would not want outliers to affect the solution. One possibility is to not take such instances into account when calculating the parameters—for example, means and covariances. Note that to detect an outlier, we can use the Mahalanobis distance or the likelihood, but we cannot use the posterior. We discuss a nonparametric method for detecting outliers in section 8.7.

11. Having generated a dendrogram, can we “prune” it?

7.12 References

Alpaydın, E. 1998. “Soft Vector Quantization and the EM Algorithm.” *Neural Networks* 11:467–477.

Barrow, H. B. 1989. “Unsupervised Learning.” *Neural Computation* 1:295–311.

Bezdek, J. C., and N. R. Pal. 1995. “Two Soft Relatives of Learning Vector Quantization.” *Neural Networks* 8:729–743.

Bishop, C. M. 1999. “Latent Variable Models.” In *Learning in Graphical Models*, ed. M. I. Jordan, 371–403. Cambridge, MA: MIT Press.

Dempster, A. P., N. M. Laird, and D. B. Rubin. 1977. “Maximum Likelihood from Incomplete Data via the EM Algorithm.” *Journal of Royal Statistical Society* B 39:1–38.

Gersho, A., and R. M. Gray. 1992. *Vector Quantization and Signal Compression*. Boston: Kluwer.

Ghahramani, Z., and G. E. Hinton. 1997. *The EM Algorithm for Mixtures of Factor Analyzers*. Technical Report CRG TR-96-1, Department of Computer Science, University of Toronto (revised Feb. 1997).

Jain, A. K., and R. C. Dubes. 1988. *Algorithms for Clustering Data*. New York: Prentice Hall.

Jain, A. K., M. N. Murty, and P. J. Flynn. 1999. “Data Clustering: A Review.” *ACM Computing Surveys* 31:264–323.

McLachlan, G. J., and K. E. Basford. 1988. *Mixture Models: Inference and Applications to Clustering*. New York: Marcel Dekker.

McLachlan, G. J., and T. Krishnan. 1997. *The EM Algorithm and Extensions*. New York: Wiley.

Moerland, P. 1999. “A Comparison of Mixture Models for Density Estimation.” In *International Conference on Artificial Neural Networks*, ed. D. Willshaw and A. Murray, 25–30. London, UK: IEE Press.

Ng, A., M. I. Jordan, and Y. Weiss. 2002. “On Spectral Clustering: Analysis and an Algorithm.” In *Advances in Neural Information Processing Systems 14*, ed. T. Dietterich, S. Becker, and Z. Ghahramani, 849–856. Cambridge, MA: MIT Press.

Ormoneit, D., and V. Tresp. 1996. “Improved Gaussian Mixture Density Estimates using Bayesian Penalty Terms and Network Averaging.” In *Advances in Neural Information Processing Systems 8*, ed. D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, 542–548. Cambridge, MA: MIT Press.

Redner, R. A., and H. F. Walker. 1984. “Mixture Densities, Maximum Likelihood and the EM Algorithm.” *SIAM Review* 26:195–239.

Rubin, D. B., and D. T. Thayer. 1982. “EM Algorithms for ML Factor Analysis.” *Psychometrika* 47:69–76.

Shi, J., and J. Malik. 2000. “Normalized Cuts and Image Segmentation.” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22:888–905.

Tipping, M. E., and C. M. Bishop. 1999. “Mixtures of Probabilistic Principal Component Analyzers.” *Neural Computation* 11:443–482.

Titterington, D. M., A. F. M. Smith, and E. E. Makov. 1985. *Statistical Analysis of Finite Mixture Distributions*. New York: Wiley.

von Luxburg, U. 2007. “A Tutorial on Spectral Clustering.” *Statistical Computing* 17:395–416.

Xu, L., and M. I. Jordan. 1996. “On Convergence Properties of the EM Algorithm for Gaussian Mixtures.” *Neural Computation* 8:129–151.

Xu, R., and D. Wunsch II. 2005. “Survey of Clustering Algorithms.” *IEEE Transactions on Neural Networks* 16:645–678.