This idea works for any subset of attributes and any combination of attribute values, but it may have to be applied recursively. For example, to obtain the count for *humidity* = *high*, *windy* = *false*, and *play* = *no*, we need the count for *windy* = *false* and *play* = *no* and the count for *humidity* = *normal*, *windy* = *false*, and *play* = *no*. We obtain the former by subtracting the count for *windy* = *true* and *play* = *no* (3) from the count for *play* = *no* (5), giving 2, and the latter by subtracting the count for *humidity* = *normal*, *windy* = *true*, and *play* = *no* (1) from the count for *humidity* = *normal* and *play* = *no* (1), giving 0. Thus there must be $2 - 0 = 2$ instances with *humidity* = *high*, *windy* = *false*, and *play* = *no*, which is correct.

AD trees only pay off if the data contains many thousands of instances. It is pretty obvious that they do not help on the weather data. The fact that they yield no benefit on small data sets means that, in practice, it makes little sense to expand the tree all the way down to the leaf nodes. Usually, a cutoff parameter k is employed, and nodes covering fewer than k instances hold a list of pointers to these instances rather than a list of pointers to other nodes. This makes the trees smaller and more efficient to use.

This section has only skimmed the surface of the subject of learning Bayesian networks. We left open questions of missing values, numeric attributes, and hidden attributes. We did not describe how to use Bayesian networks for regression tasks. Some of these topics are discussed later in this chapter. Bayesian networks are a special case of a wider class of statistical models called *graphical models*, which include networks with undirected edges (called *Markov networks*). Graphical models have attracted a lot of attention in the machine learning community and we will discuss them in Section 9.6.

9.3 CLUSTERING AND PROBABILITY DENSITY ESTIMATION

An incremental heuristic clustering approach was described in Section 4.8. While it works reasonably well in some practical situations, it has shortcomings: the arbitrary division by k in the category utility formula that is necessary to prevent overfitting, the need to supply an artificial minimum value for the standard deviation of clusters, and the ad hoc cutoff value to prevent every single instance from becoming a cluster in its own right. On top of this is the uncertainty inherent in incremental algorithms. To what extent is the result dependent on the order of examples? Are the local restructuring operations of merging and splitting really enough to reverse the effect of bad initial decisions caused by unlucky ordering? Does the final result represent even a *local* maximum of category utility? Add to this the problem that one never knows how far the final configuration is to a *global* maximum—and, of course, the standard trick of repeating the clustering procedure several times and choosing the best will destroy the incremental nature of the algorithm. Finally, does not the hierarchical nature of the result really beg the question of which are the *best* clusters? There are so many clusters in Fig. 4.21 that it is hard to separate the wheat from the chaff.

A more principled statistical approach to the clustering problem can overcome some of these shortcomings. From a probabilistic perspective, the goal of clustering is to find the most likely set of clusters given the data (and, inevitably, prior expectations). Because no finite amount of evidence is enough to make a completely firm decision on the matter, instances—even training instances—should not be placed categorically in one cluster or the other: instead they have a certain probability of belonging to each cluster. This helps to eliminate the brittleness that is often associated with schemes that make hard and fast judgments.

The foundation for statistical clustering is a statistical model called a *finite mixture* model. A *mixture* is a set of k probability distributions, representing k clusters, that govern the attribute values for members of that cluster. In other words, each distribution gives the probability that a particular instance would have a certain set of attribute values if it were *known* to be a member of that cluster. Each cluster has a different distribution. Any particular instance "really" belongs to one and only one of the clusters, but it is not known which one. Finally, the clusters are not equally likely: there is some probability distribution that reflects their relative populations.

THE EXPECTATION MAXIMIZATION ALGORITHM FOR A MIXTURE OF GAUSSIANS

One of the simplest finite mixture situations is when there is only one numeric attribute, which has a Gaussian or normal distribution for each cluster—but with different means and variances. The clustering problem is to take a set of instances—in this case each instance is just a number—and a prespecified number of clusters, and work out each cluster's mean and variance and the population distribution between the clusters. The mixture model combines several normal distributions, and its probability density function looks like a mountain range with a peak for each component.

Fig. 9.5 shows a simple example. There are two clusters A and B, and each has a normal distribution with means and standard deviations μ_A and σ_A for cluster A, and μ_B and σ_B for cluster B. Samples are taken from these distributions, using cluster A with probability p_A and cluster B with probability p_B (where $p_A + p_B = 1$), resulting in a data set like that shown. Now, imagine being given the data set without the classes—just the numbers—and being asked to determine the five parameters that characterize the model: μ_A, σ_A, μ_B, σ_B, and p_A (the parameter p_B can be calculated directly from p_A). That is the finite mixture problem.

If you knew which of the two distributions each instance came from, finding the five parameters would be easy—just estimate the mean and standard deviation for the cluster A samples and the cluster B samples separately, using the formulas

Data

A 51	A 46	B 65
A 43	B 64	A 46
B 62	A 51	A 39
B 64	A 52	B 62
A 45	B 62	B 64
A 42	A 49	A 52
A 46	A 48	B 63
A 45	B 62	B 64
A 45	A 43	A 48
B 62	A 40	B 64
A 47	A 48	A 48
A 52	B 64	A 51
B 64	A 51	A 48
A 51	B 63	B 64
B 65	A 43	A 42
A 48	B 65	A 48
A 49	B 66	A 41

Model

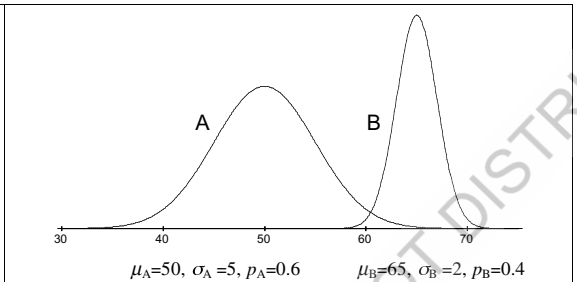

FIGURE 9.5

A two-class mixture model.

$$\mu = \frac{x_1 + x_2 + \cdots + x_n}{n},$$

$$\sigma^2 = \frac{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \cdots + (x_n - \mu)^2}{n - 1}.$$

(The use of $n-1$ rather than n as the denominator in the second formula ensures an unbiased estimate of the variance, rather than the maximum likelihood estimate: it makes little difference in practice if n is used instead.) Here, x_1, x_2,. . ., x_n are the samples from the distribution A or B. To estimate the fifth parameter p_A, just take the proportion of the instances that are in the A cluster.

If you knew the five parameters, finding the (posterior) probabilities that a given instance comes from each distribution would be easy. Given an instance x_i, the probability that it belongs to cluster A is

$$P(A|x_i) = \frac{P(x_i|\mathrm{A}) \cdot P(A)}{P(x_i)} = \frac{N(x_i; \mu_\mathrm{A}, \sigma_\mathrm{A}) p_\mathrm{A}}{P(x_i)},$$

where $N(x; \mu_\mathrm{A}, \sigma_\mathrm{A})$ is the normal or Gaussian distribution function for cluster A, i.e.:

$$N(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

In practice we calculate the numerators for both $P(\mathrm{A}|x_i)$ and $P(\mathrm{B}|x_i)$, then normalize them by dividing by their sum, which is $P(x_i)$. This whole procedure is just the same as the way numeric attributes are treated in the Naïve Bayes learning scheme of Section 4.2. And the caveat explained there applies here too: strictly speaking, $N(x_i; \mu_\mathrm{A}, \sigma_\mathrm{A})$ is not the probability $\mathrm{P}(x|\mathrm{A})$ because the probability of x being any particular real number x_i is zero. Instead, $N(x_i; \mu_\mathrm{A}, \sigma_\mathrm{A})$ is a probability density, which is turned into a probability by the normalization

process used to compute the posterior. Note that the final outcome is not a particular cluster but rather the (posterior) *probability* with which x_i belongs to cluster A or cluster B.

The problem is that we know neither of these things: not the distribution that each training instance came from nor the five parameters of the mixture model. So we adopt the procedure used for the *k*-means clustering algorithm and iterate. Start with initial guesses for the five parameters, use them to calculate the cluster probabilities for each instance, use these probabilities to re-estimate the parameters, and repeat. (If you prefer, you can start with guesses for the classes of the instances instead.) This is an instance of the *expectation–maximization* or *EM* algorithm. The first step, calculation of the cluster probabilities (which are the "expected" class values) is the "expectation"; the second, calculation of the distribution parameters, is our "maximization" of the likelihood of the distributions given the data.

A slight adjustment must be made to the parameter estimation equations to account for the fact that it is only cluster probabilities, not the clusters themselves, that are known for each instance. These probabilities just act like weights. If w_i is the probability that instance i belongs to cluster A, the mean and standard deviation for cluster A are

$$\mu_A = \frac{w_1 x_1 + w_2 x_2 + \cdots + w_n x_n}{w_1 + w_2 + \cdots + w_n}$$

$$\sigma_A^2 = \frac{w_1 (x_1 - \mu)^2 + w_2 (x_2 - \mu)^2 + \cdots + w_n (x_n - \mu)^2}{w_1 + w_2 + \cdots + w_n}$$

where now the x_i are *all* the instances, not just those belonging to cluster A. (This differs in a small detail from the estimate for the standard deviation given above: if all weights are equal the denominator is n rather than $n-1$, which uses the maximum likelihood estimator rather than the unbiased estimator.)

Now consider how to terminate the iteration. The *k*-means algorithm stops when the classes of the instances do not change from one iteration to the next—a "fixed point" has been reached. In the EM algorithm things are not quite so easy: the algorithm converges toward a fixed point but never actually gets there. We can see how close it is getting by calculating the overall (marginal) *likelihood* that the data came from this model, given the values for the five parameters. The marginal likelihood is obtained by summing (or marginalizing) over the two components of the Gaussian mixture, i,e,

$$\prod_{i=1}^{n} P(x_i) = \prod_{i=1}^{n} \sum_{c_i} P(x_i | c_i) \cdot P(c_i)$$

$$= \prod_{i=1}^{n} (N(x_i; \mu_A, \sigma_A) p_A + N(x_i; \mu_B, \sigma_B) p_B).$$

This is the product of the marginal probability densities of the individual instances, which are obtained from the sum of the probability density under each

normal distribution $N(x; \mu, \sigma)$, weighted by the appropriate (prior) class probability. The cluster membership variable c is a so-called hidden (or latent) variable; we sum it out to obtain the marginal probability density of an instance.

This overall likelihood is a measure of the "goodness" of the clustering and increases at each iteration of the EM algorithm. The above equation and the expressions $N(x_i; \mu_A, \sigma_A)$ and $N(x_i; \mu_B, \sigma_B)$ are probability densities and not probabilities, so they do not necessarily lie between 0 and 1: nevertheless, the resulting magnitude still reflects the quality of the clustering. In practical implementations the log-likelihood is calculated instead: this is done by summing the logarithms of the individual components, avoiding multiplications. But the overall conclusion still holds: you should iterate until the increase in log-likelihood becomes negligible. For example, a practical implementation might iterate until the difference between successive values of log-likelihood is less than 10^{-10} for 10 successive iterations. Typically, the log-likelihood will increase very sharply over the first few iterations and then converge rather quickly to a point that is virtually stationary.

Although the EM algorithm is guaranteed to converge to a maximum, this is a *local* maximum and may not necessarily be the same as the global maximum. For a better chance of obtaining the global maximum, the whole procedure should be repeated several times, with different initial guesses for the parameter values. The overall log-likelihood figure can be used to compare the different final configurations obtained: just choose the largest of the local maxima.

EXTENDING THE MIXTURE MODEL

Now that we have seen the Gaussian mixture model for two distributions, let us consider how to extend it to more realistic situations. The basic method is just the same, but because the mathematical notation becomes formidable we will not develop it in full detail.

Changing the algorithm from two-cluster problems to situations with multiple clusters is completely straightforward, so long as the number k of normal distributions is given in advance.

The model can easily be extended from a single numeric attribute per instance to multiple attributes as long as independence between attributes is assumed. The probabilities for each attribute are multiplied together to obtain the joint probability (density) for the instance, just as in the Naïve Bayes method.

When the dataset is known in advance to contain correlated attributes, the independence assumption no longer holds. Instead, two attributes can be modeled jointly by a bivariate normal distribution, in which each has its own mean value but the two standard deviations are replaced by a "covariance matrix" with four numeric parameters. In Appendix A.2 we show the mathematics for the multivariate Gaussian distribution; the special case of a diagonal covariance model leads to a Naïve Bayesian interpretation. Several correlated attributes can be handled using a multivariate distribution. The number of parameters increases with the

square of the number of jointly varying attributes. With n independent attributes, there are $2n$ parameters, a mean and a standard deviation for each. With n covariant attributes, there are $n + n(n + 1)/2$ parameters, a mean for each, and an $n \times n$ covariance matrix that is symmetric and therefore involves $n(n + 1)/2$ different quantities. This escalation in the number of parameters has serious consequences for overfitting, as we will explain later.

To cater for nominal attributes, the normal distribution must be abandoned. Instead, a nominal attribute with v possible values is characterized by v numbers representing the probability of each one. A different set of numbers is needed for every cluster; kv parameters in all. The situation is very similar to the Naïve Bayes method. The two steps of expectation and maximization correspond exactly to operations we have studied before. Expectation—estimating the cluster to which each instance belongs given the distribution parameters—is just like determining the class of an unknown instance. Maximization—estimating the parameters from the classified instances—is just like determining the attribute–value probabilities from the training instances, with the small difference that in the EM algorithm instances are assigned to classes probabilistically rather than categorically. In Section 4.2 we encountered the problem that probability estimates can turn out to be zero, and the same problem occurs here too. Fortunately, the solution is just as simple—use the Laplace estimator.

Naïve Bayes assumes that attributes are independent—i.e., why it is called "naïve." A pair of correlated nominal attributes with v_1 and v_2 possible values, respectively, can be replaced by a single covariant attribute with $v_1 v_2$ possible values. Again, the number of parameters escalates as the number of dependent attributes increases, and this has implications for probability estimates and overfitting.

The presence of both numeric and nominal attributes in the data to be clustered presents no particular problem. Covariant numeric and nominal attributes are more difficult to handle, and we will not describe them here.

Missing values can be accommodated in various different ways. In principle, they should be treated as unknown and the EM process adapted to estimate them as well as the cluster means and variances. A simple way is to replace them by means or modes in a preprocessing step.

With all these enhancements, probabilistic clustering becomes quite sophisticated. The EM algorithm is used throughout to do the basic work. The user must specify the number of clusters to be sought, the type of each attribute (numeric or nominal), which attributes are to modeled as covarying, and what to do about missing values. Moreover, different distributions can be used. Although the normal distribution is usually a good choice for numeric attributes, it is not suitable for attributes (such as weight) that have a predetermined minimum (zero, in the case of weight) but no upper bound; in this case a "log-normal" distribution is more appropriate. Numeric attributes that are bounded above and below can be modeled by a "log-odds" distribution. Attributes that are integer counts rather than real values are best modeled by the "Poisson" distribution. A comprehensive system might allow these distributions to be specified individually for each

attribute. In each case, the distribution involves numeric parameters—probabilities of all possible values for discrete attributes and mean and standard deviation for continuous ones.

In this section we have been talking about clustering. But you may be thinking that these enhancements could be applied just as well to the Naïve Bayes algorithm too—and you could be right. A comprehensive probabilistic modeler could accommodate both clustering and classification learning, nominal and numeric attributes with a variety of distributions, various possibilities of covariation, and different ways of dealing with missing values. The user would specify, as part of the domain knowledge, which distributions to use for which attributes.

CLUSTERING USING PRIOR DISTRIBUTIONS

However, there is a snag: overfitting. You might say that if we are not sure which attributes are dependent on each other, why not be on the safe side and specify that *all* the attributes are covariant? The answer is that the more parameters there are, the greater the chance that the resulting structure is overfitted to the training data—and covariance increases the number of parameters dramatically. The problem of overfitting occurs throughout machine learning, and probabilistic clustering is no exception. There are two ways that it can occur: through specifying too large a number of clusters and through specifying distributions with too many parameters.

The extreme case of too many clusters occurs when there is one for every data point: clearly, that will be overfitted to the training data. In fact, in the mixture model, problems will occur whenever any one of the normal distributions becomes so narrow that it is centered on just one data point. Consequently, implementations generally insist that clusters contain at least two different data values.

Whenever there are a large number of parameters, the problem of overfitting arises. If you were unsure of which attributes were covariant, you might try out different possibilities and choose the one that maximized the overall probability of the data given the clustering that was found. Unfortunately, the more parameters there are, the larger the overall data probability will tend to be—not necessarily because of better clustering but because of overfitting. The more parameters there are to play with, the easier it is to find a clustering that seems good.

It would be nice if somehow you could penalize the model for introducing new parameters. One principled way of doing this is to adopt a Bayesian approach in which every parameter has a prior probability distribution whose effect is incorporated into the overall likelihood figure. In a sense, the Laplace estimator that we met in Section 4.2, and whose use we advocated earlier to counter the problem of zero probability estimates for nominal values, is just such a device. Whenever there are few observations, it exacts a penalty because it makes probabilities that are zero, or close to zero, greater, and this will decrease the overall likelihood of the data. In fact, the Laplace estimator is tantamount to using a particular prior distribution for the parameter concerned. Making two nominal attributes covariant will exacerbate the problem of sparse data. Instead of $v_1 + v_2$ parameters, where v_1 and

v_2 are the number of possible values, there are now $v_1 v_2$, greatly increasing the chance of a large number of small observed frequencies.

The same technique can be used to penalize the introduction of large numbers of clusters, just by using a prespecified prior distribution that decays sharply as the number of clusters increases.

AutoClass is a comprehensive Bayesian clustering scheme that uses the finite mixture model with prior distributions on all the parameters. It allows both numeric and nominal attributes, and uses the EM algorithm to estimate the parameters of the probability distributions to best fit the data. Because there is no guarantee that the EM algorithm converges to the global optimum, the procedure is repeated for several different sets of initial values. But that is not all. AutoClass considers different numbers of clusters and can consider different amounts of covariance and different underlying probability distribution types for the numeric attributes. This involves an additional, outer level of search. For example, it initially evaluates the log-likelihood for 2, 3, 5, 7, 10, 15, and 25 clusters: after that, it fits a log-normal distribution to the resulting data and randomly selects from it more values to try. As you might imagine, the overall algorithm is extremely computation intensive. In fact, the actual implementation starts with a prespecified time bound and continues to iterate as long as time allows. Give it longer and the results may be better!

A simpler way of selecting an appropriate model—e.g., to choose the number of clusters—is to compute the likelihood on a separate validation set that has not been used to fit the model. This can be repeated with multiple train-validation splits, just as in the case of classification models—e.g., with *k*-fold cross-validation. In practice, the ability to pick a model in this way is a big advantage of probabilistic clustering approaches compared to heuristic clustering methods.

Rather than showing just the most likely clustering to the user, it may be best to present all of them, weighted by probability. Recently, fully Bayesian techniques for *hierarchical* clustering have been developed that produce as output a probability distribution over possible hierarchical structures representing a dataset. Fig. 9.6 is a visualization, known as a *DensiTree*, that shows the set of all trees for a particular dataset in a triangular shape. The tree is best described in terms of its "clades," a biological term from the Greek *klados*, meaning *branch*, for a group of the same species that includes all ancestors. Here there are five clearly distinguishable clades. The first and fourth correspond to a single leaf, while the fifth has two leaves that are so distinct they might be considered clades in their own right. The second and third clades each have five leaves, and there is large uncertainty in their topology. Such visualizations make it easy for people to grasp the possible hierarchical clusterings of their data, at least in terms of the big picture.

CLUSTERING WITH CORRELATED ATTRIBUTES

Many clustering methods make the assumption of independence among the attributes. An exception is AutoClass, which does allow the user to specify in advance

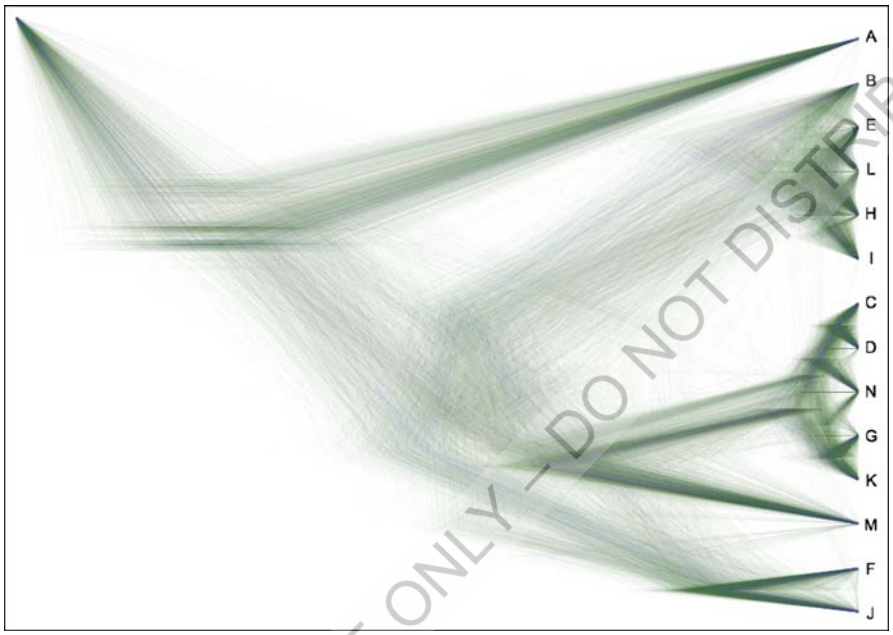

FIGURE 9.6

DensiTree showing possible hierarchical clusterings of a given data set.

that two or more attributes are dependent and should be modeled with a joint probability distribution. (There are restrictions, however: nominal attributes may vary jointly, as may numeric attributes, but not both together. Moreover, missing values for jointly varying attributes are not catered for.) It may be advantageous to preprocess a data set to make the attributes more independent, using statistical techniques such as the independent component transform described in Section 8.3. Note that joint variation that is specific to particular classes will not be removed by such techniques; they only remove overall joint variation that runs across all classes.

If all attributes are continuous, more advanced clustering methods can help capture joint variation on a per-cluster basis, without having the number of parameters explode when there are many dimensions. As discussed above, if each covariance matrix in a Gaussian mixture model is "full" we need to estimate $n(n+1)/2$ parameters per mixture component. However, as we will see in Section 9.6, principal component analysis can be formulated as a probabilistic model, yielding probabilistic principal component analysis (PPCA), and approaches known as *mixtures of principal component analyzers* or *mixtures of factor analyzers* provide ways of using a much smaller number of parameters to represent large covariance matrices. In fact, the problem of estimating $n(n+1)/2$ parameters in a *full* covariance matrix can be transformed into the problem of estimating as few as $n \times d$ parameters in a *factorized covariance matrix*, where d

can be chosen to be small. The idea is to decompose the covariance matrix $\mathbf{M}$ into the form $\mathbf{M} = (\mathbf{W}\mathbf{W}^{\mathrm{T}} + \mathbf{D})$, where $\mathbf{W}$ is typically a long and skinny matrix of size $n \times d$, with as many rows as there are dimensions n of the input, and as many columns d as there are dimensions in the reduced space. Standard PCA corresponds to setting $\mathbf{D} = 0$; PPCA corresponds to using the form $\mathbf{D} = \sigma^2\mathbf{I}$, where σ^2 is a scalar parameter and $\mathbf{I}$ is the identity matrix; and factor analysis corresponds to using a diagonal matrix for $\mathbf{D}$. The mixture model versions give each mixture component this type of factorization.

KERNEL DENSITY ESTIMATION

Mixture models can provide compact representations of probability distributions but do not necessarily fit the data well. In Chapter 4, Algorithms: the basic methods, we mentioned that when the form of a probability distribution is unknown, an approach known as *kernel density estimation* can be used to approximate the underlying distribution more accurately. This estimates the underlying true probability distribution $p(\mathbf{x})$ of data $\mathbf{x}_1$, $\mathbf{x}_2$,..., $\mathbf{x}_n$ using a kernel density *estimator*, which can be written in the following general form

$$\hat{p}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} K_\sigma(\mathbf{x}, \mathbf{x}_i) = \frac{1}{n\sigma}\sum_{i=1}^{n} K\left[\frac{\mathbf{x} - \mathbf{x}_i}{\sigma}\right],$$

where $K()$ is a nonnegative kernel function that integrates to one. Here, we use the notation $\hat{p}(\mathbf{x})$ to emphasize that this is an estimation of the true (unknown) distribution $p(\mathbf{x})$. The parameter $\sigma > 0$ is the *bandwidth* of the kernel, and serves as a form of smoothing parameter for the approximation. When the kernel function is defined using σ as a subscript it is known as a "scaled" kernel function and is given by $K_\sigma(\mathbf{x}) = 1/\sigma\, K(\mathbf{x}/\sigma)$. Estimating densities using kernels is also known as *Parzen window density estimation*.

Popular kernel functions include the Gaussian, box, triangle, and Epanechnikov kernels. The Gaussian kernel is popular because of its simple and attractive mathematical form. The box kernel implements a windowing function, whereas the triangle kernel implements a smoother, but still conceptually simple, window. The Epanechnikov kernel can be shown to be optimal under a mean squared error metric. The bandwidth parameter affects the smoothness of the estimator and the quality of the estimate. There are several methods for coming up with an appropriate bandwidth, ranging from heuristics motivated by theoretical results on known distributions to empirical choices based on validation sets and cross-validation techniques. Many software packages offer the choice between simple heuristic default values, bandwidth selection through cross-validation methods, and the use of plug-in estimators derived from further analytical analysis.

Kernel density estimation is closely related to k-nearest neighbor density estimation, and it can be shown that both techniques converge to the true distribution $p(\mathbf{x})$ as the amount of data grows towards infinity. This result, combined with the

fact that they are easy to implement, makes kernel density estimators attractive methods in many situations.

Consider, e.g., the practical problem of finding outliers in data, given only positive or only negative examples (or perhaps with just a handful of examples of the other class). One effective approach is to do the best possible job of modeling the probability distribution of the data for the plentiful class using a kernel density estimator, and considering new data to which the model assigns low probability as outliers.

COMPARING PARAMETRIC, SEMIPARAMETRIC AND NONPARAMETRIC DENSITY MODELS FOR CLASSIFICATION

One might view a mixture model as intermediate between two extreme ways of modeling distributions by estimating probability densities. One extreme is a single simple parametric form such as the Gaussian distribution. It is easy to estimate the relevant parameters. However, data often arises from a far more complex distribution. Mixture models use two or more Gaussians to approximate the distribution. In the limit, at the other extreme, one Gaussian is used for each data point. This is kernel density estimation with a Gaussian kernel function.

Fig. 9.7 shows a visual example of this spectrum of models. A density estimate for each class of a 3-class classification problem has been created using three different techniques. Fig. 9.7A uses a single Gaussian distribution for each class, an approach that is often referred to as a "parametric" technique. Fig. 9.7B uses a Gaussian mixture model with two components per class, a "semiparametric" technique in which the number of Gaussians can be determined using a variety of methods. Fig. 9.7C uses a kernel density estimate with a Gaussian kernel on each example, a "nonparametric" method. Here the model complexity grows in proportion to the volume of data.

All three approaches define density models for each class, so Bayes' rule can be used to compute the posterior probability over all classes for any given input.

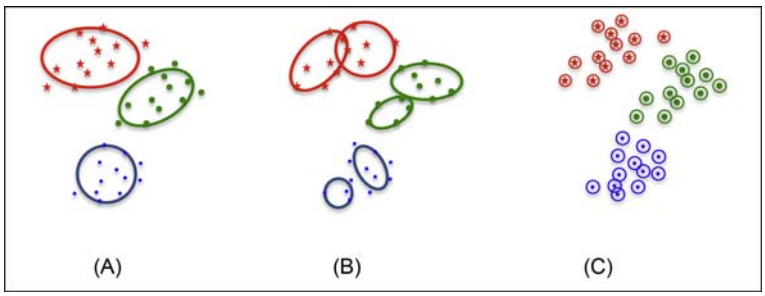

FIGURE 9.7

Probability contours for three types of model, all based on Gaussians.

In this way, density estimators can easily be transformed into classifiers. For simple parametric models, this is quick and easy. Kernel density estimators are guaranteed to converge to the true underlying distribution as the amount of data increases, which means that classifiers constructed from them have attractive properties. They share the computational disadvantages of nearest-neighbor classification, but, just as for nearest-neighbor classification, fast data structures exist that can make them applicable to large datasets.

Mixture models, the intermediate option, give control of model complexity without it growing with the amount of data. For this reason this approach has been standard practice for initial modeling in fields such as in speech recognition, which deal in large datasets. It allows speech recognizers to be created by first clustering data into groups, but in such a way that more complex models of temporal relationships can be added later using a hidden Markov model. (We will consider sequential and temporal probabilistic models, such as hidden Markov models, in Section 9.6.)

9.4 HIDDEN VARIABLE MODELS

We now move on to advanced learning algorithms that can infer new attributes in addition to the ones present in the data—so-called hidden (or latent) variables whose values cannot be observed. As noted in Section 9.3, a quantity called the *marginal likelihood* can be obtained by summing (or integrating) these variables out of the model. It is important not to confuse random variables with observations or hard assignments of random variables. We write $p(x_i = \tilde{x}_i) = p(\tilde{x}_i)$ to denote the probability of the random variable x_i associated with instance i taking the value represented by the observation $\tilde{x}_i$. We use h_i to represent a hidden discrete random variable, and z_i to denote a hidden continuous one. Then, given a model with observations given by $\tilde{x}_i$, the marginal likelihood is

$$L(\theta; \tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n) = \prod_{i=1}^{n} p(\tilde{x}_i; \theta) = \prod_{i=1}^{n} \int_{z_i} \sum_{h_i} p(\tilde{x}_i, z_i, h_i; \theta) dz_i,$$

where the sum is taken over all possible discrete values of h_i and the integral is taken over the entire domain of z_i. The end result of all this integrating and summing is a single number—a scalar quantity—that gives the marginal likelihood for any value of the parameters.

Maximum likelihood based learning for hidden variable models can sometimes be done using marginal likelihood, just as when no hidden variables are present, but the additional variables usually affect the parameterization used to define the model. In fact, these additional variables are often very important: *they are used to represent precisely the things we wish to mine from our data*, be it clusters, topics in a text mining problem, or the factors that underlie variation in the data. By treating parameters as random variables and using functions that are easy to manipulate, marginal likelihoods can also be used to define sophisticated