16 Density Estimation

We've learned more and more about how to describe and fit functions, but the decision to fit a function can itself be unduly restrictive. Instead of assuming a noise model for the data (such as Gaussianity), why not go ahead and deduce the underlying probability distribution from which the data were drawn? Given the distribution (or *density*) any other quantity of interest, such as a conditional forecast of a new observation, can be derived. Perhaps because this is such an ambitious goal it is surprisingly poorly covered in the literature and in a typical education, but in fact it is not only possible but extremely useful.

This chapter will cover *density estimation* at three levels of generality and complexity. First will be methods based on binning the data that are easy to implement but that can require impractical amounts of data. This is solved by casting density estimation as a problem in functional approximation, but that approach can have trouble representing all of the things that a density can do. The final algorithms are based on clustering, merging the desirable features of the preceeding ones while avoiding the attendant liabilities.

16.1 HISTOGRAMMING, SORTING, AND TREES

Given a set of measurements $\{x_n\}_{n=1}^N$ that were drawn from an unknown distribution $p(x)$, the simplest way to model the density is by histogramming x. If N_i is the number of observations between x_i and x_{i+1}, then the probability $p(x_i)$ to see a new point in the bin between x_i and x_{i+1} is approximately N_i/N (by the Law of Large Numbers).

This idea is simple but not very useful. The first problem is the amount of storage required. Let's say that we're looking at a 15-dimensional data set sampled at a 16-bit resolution. Binning it would require an array of $2^{16\times 15} \approx 10^{72}$ elements, more elements than there are atoms in the universe! But this is an unreasonable approach because most of those bins would be empty. The storage requirement becomes feasible if only occupied bins are allocated. This might appear to be circular reasoning, but in fact can be done quite simply.

The trick is to perform a *lexicographic sort*. The digits of each vector are appended to make one long number, in the preceeding example $15 \times 16 = 240$ bits long. Then the one-dimensional string of numbers is sorted, requiring $\mathcal{O}(N \log N)$ operations. A final pass through the sorted numbers counts the number of times each number appears, giving the occupancy of the multi-dimensional bin uniquely indexed by that number.

Since the resolution is known in advance this can further be reduced to a linear time

algorithm. Sorting takes $\mathcal{O}(N \log N)$ when the primitive operation is a comparison of two elements, but it can be done in linear time at a fixed resolution by sorting on a tree. This is shown schematically in Figure 16.1. Each bit of the appended number is used to decide to branch right or left in the tree. If a node already exists then the count of points passing through it is incremented by one, and if the node does not exist then it is created. Descending through the tree takes a time proportional to the number of bits, and hence the dimension, and the total running time is proportional to the number of points. The storage depends on the amount of space the data set occupies (an idea that will be revisited in Chapter 21 to characterize a time series).

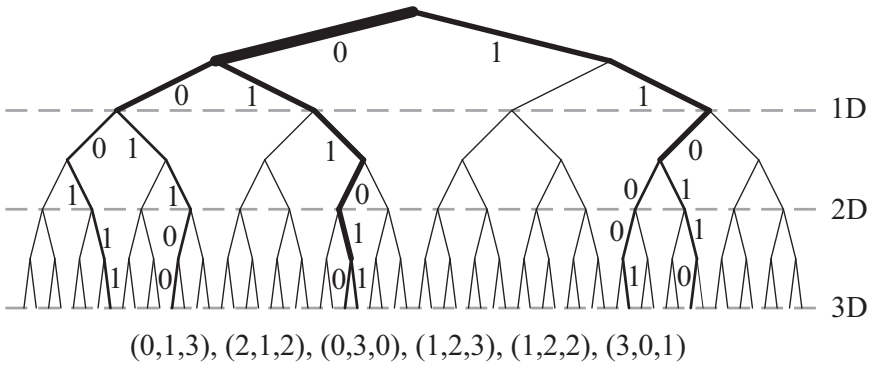

Figure 16.1. Multi-dimensional binning on a binary tree.

Sorting on a tree makes it tractable to accumulate a multi-dimensional histogram but it leaves another problem: the number of points in each bin is variable. Assuming Poisson counting errors, this means that the relative uncertainty of the count of each bin ranges from 100% for a bin with a single point in it to a small fraction for a bin with a large occupancy. Many operations (such as taking a logarithm in calculating an entropy) will magnify the influence of the large errors associated with the low probability bins.

The solution to this problem is to use bins that have a fixed occupancy rather than a fixed size. These can be constructed by using a *k-D tree* [Preparata & Shamos, 1985], shown for 2D in Figure 16.2. A vertical cut is initially chosen at a location x_1^1 that has half of the data on the left and half on the right. Then each of these halves is cut horizontally at y_1^1 and y_2^1 to leave half of the data on top and half on bottom. These four bins are next cut horizontally at x_{1-4}^2, and so forth. In a higher-dimensional space the cuts cycle over the directions of the axes. By construction each bin has the same number of points, but the volume of the bin varies. Since the probability is estimated by the number of points divided by the volume, instead of varying the number of points for a given volume as was done before we can divide the fixed number of points by the varying volumes. This gives a density estimate with constant error per bin.

Building a k-D tree is more work than fixed-mass binning on a tree, particularly if new points need to be added later (which can require moving many of the k-D tree partitions). There is a simple trick that approximates fixed-volume binning while retaining the convenience of a linear-time tree. The idea is to interleave the bits of each coordinate of the point to be sorted rather than appending them, so that each succeeding bit is associated with a different direction. Then the path that the sorted sequence of numbers takes

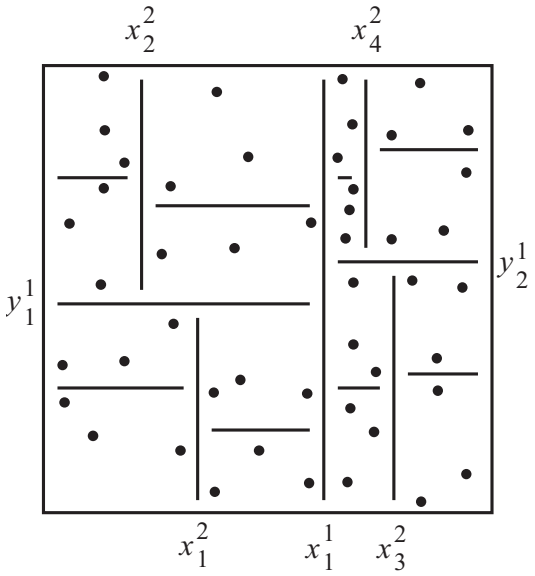

Figure 16.2. k-D tree binning in 2D.

through the space is a fractal curve [Mandelbrot, 1983], shown in 2D in Figure 16.3. This has the property that the average distance between two points in the space is proportional to the linear distance along the curve connecting them. Therefore the difference between the indices of two bins estimates the volume that covers them.

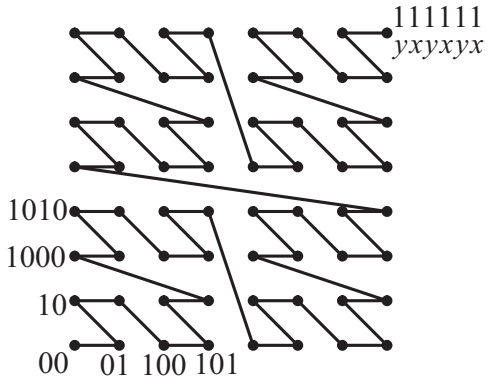

Figure 16.3. Fractal binning in 2D.

Even histogramming with constant error per bin still has a serious problem. Let's say that we're modestly looking at three-dimensional data sampled at an 8-bit resolution. If it is random with a uniform probability distribution and is histogrammed in M bins then the probability of each bin will be roughly $1/M$. The entropy of this distribution is

$$H = -\sum_{i=1}^{M} p_i \log_2 p_i$$

$$
\begin{aligned}
&= \log_2 M \\
\Rightarrow 2^H &= M \quad .
\end{aligned} \tag{16.1}
$$

But we know that the entropy is $H = 8 \times 3 = 24$ bits, therefore there must be $2^{24} \approx 10^7$ occupied bins or the entropy will be underestimated. Since the smallest possible data set would have one point per bin this means that at least 10 million points must be used.

The real problem is that histogramming has no notion of neighborhood. Two adjacent bins could represent different letters in an alphabet just as well as nearby parts of a space, ignoring the strong constraint that neighboring points might be expected to behave similarly. What's needed is some kind of functional approximation that can make predictions about the density in locations where points haven't been observed.

16.2 FITTING DENSITIES

Let's say that we're given a set of data points $\{x_n\}_{n=1}^N$ and want to infer the density $p(x)$ from which they were drawn. If we're willing to add some kind of prior belief, for example that nearby points behave similarly, then we can hope to find a functional representation for $p(x)$. This must of course be normalized to be a valid density, and a possible further constraint is *compact support*: the density is restricted to be nonzero on a bounded set. In this section we'll assume that the density vanishes outside the interval [0,1]; the generalization is straightforward.

One way to relate this problem to what we've already studied is to introduce the *cumulative* distribution

$$
P(x) \equiv \int_0^x p(x)\, dx \quad . \tag{16.2}
$$

If we could find the cumulative distribution then the density follows by differentiation,

$$
p(x) = \frac{dP}{dx} \quad . \tag{16.3}
$$

There is in fact an easy way to find a starting guess for $P(x)$: sort the data set. $P(x_i)$ is defined to be the fraction of points below x_i, therefore if i is the index of where point x_i appears in the sorted set, then

$$
y_i \equiv \frac{i}{N+1} \approx P(x_i) \quad . \tag{16.4}
$$

The denominator is taken to be $N + 1$ instead of N because the normalization constraint is effectively an extra point $P(1) = 1$.

To estimate the error, recognize that $P(x_i)$ is the probability to draw a new point below x_i, and that $NP(x_i)$ is the expected number of such points in a data set of N points. If we assume Gaussian errors then the variance around this mean is $NP(x_i)[1 - P(x_i)]$ [Gershenfeld, 2000]. Since $y_i = i/(N + 1)$, the standard deviation in y_i is the standard deviation in the number of points below it, i, divided by $N + 1$,

$$
\begin{aligned}
\sigma_i &= \frac{1}{N+1} \left\{ NP(x_i)[1 - P(x_i)] \right\}^{1/2} \\
&= \frac{1}{N+1} \left[N \frac{i}{N+1} \left(1 - \frac{i}{N+1} \right) \right]^{1/2} \quad .
\end{aligned} \tag{16.5}
$$

This properly vanishes at the boundaries where we know that $P(0) = 0$ and $P(1) = 1$.

We can now fit a function to the data set $\{x_i, y_i, \sigma_i\}$. A convenient parameterization is

$$\begin{aligned} P(x) &= x + \sum_{m=1}^{M} a_m \sin(m\pi x) \\ \Rightarrow p(x) &= 1 + \sum_{m=1}^{M} a_m m\pi \cos(m\pi x) \quad , \end{aligned} \tag{16.6}$$

which enforces the boundary conditions $P(0) = 0$ and $P(1) = 1$ regardless of the choice of coefficients (this representation also imposes the constraint that $dp/dx = 0$ at $x = 0$ and 1, which may or may not be appropriate).

One way to impose the prior that nearby points behave similarly is to use the integral square curvature of $P(x)$ as a regularizer (Section 14.5), which seeks to minimize the quantity

$$I_0 = \int_0^1 \left(\frac{d^2 P}{dx^2} \right)^2 dx \quad . \tag{16.7}$$

The model mismatch is given by

$$I_1 = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{P(x_i) - y_i}{\sigma_i} \right]^2 \quad . \tag{16.8}$$

The expected value of I_1 is 1; we don't want to minimize it because passing exactly through the data is a very unlikely event given the uncertainty. Therefore we want to impose this constraint by introducing a Lagrange multiplier in a variational sum

$$I = I_0 + \lambda I_1 \quad . \tag{16.9}$$

Solving

$$\frac{\partial I}{\partial a_m} = 0 \tag{16.10}$$

for all m gives the set of coefficients $\{a_m\}$ as a function of λ, and then given the coefficients we can do a one-dimensional search to find the value of λ that results in $I_1 = 1$.

Plugging in the expansion (16.5) and doing the derivatives and integrals shows that if the matrix **A** is defined by

$$A_{lm} = 2\pi^4 l^2 m^2 \begin{cases} \dfrac{1}{2(l^2 - m^2)\pi} \{(l+m)\sin[(l-m)\pi] \\ \qquad -(l-m)\sin[(l+m)\pi]\} & (l \neq m) \\ \frac{1}{2} - \sin(2m\pi)/(4m\pi) & (l = m) \end{cases} , \tag{16.11}$$

the matrix **B** by

$$B_{lm} = \frac{2}{N} \sum_{i=1}^{N} \sin(l\pi x_i) \sin(m\pi x_i)/\sigma_i^2 \quad , \tag{16.12}$$

and the vector $\vec{c}$ by

$$c_m = \frac{2}{N}\sum_{i=1}^{N}(y_i - x_i)\sin(m\pi x_i)/\sigma_i^2 \quad , \tag{16.13}$$

then the vector of expansion coefficients $\vec{a} = (a_1, \ldots, a_M)$ is given by

$$\vec{a} = \lambda(\mathbf{A} + \lambda\mathbf{B})^{-1} \cdot \vec{c} \quad . \tag{16.14}$$

An example is shown in Figure 16.4. Fifty points were drawn from a Gaussian distribution and the cumulative density was fit using 50 basis functions. Even though there are as many coefficients as data points, the regularization insures that the resulting density does not overfit the data. This figure also demonstrates that the presence of the regularizer biases the estimate of the variance upwards, because the curvature is minimized if the fitted curve starts above the cumulative histogram and ends up below it.

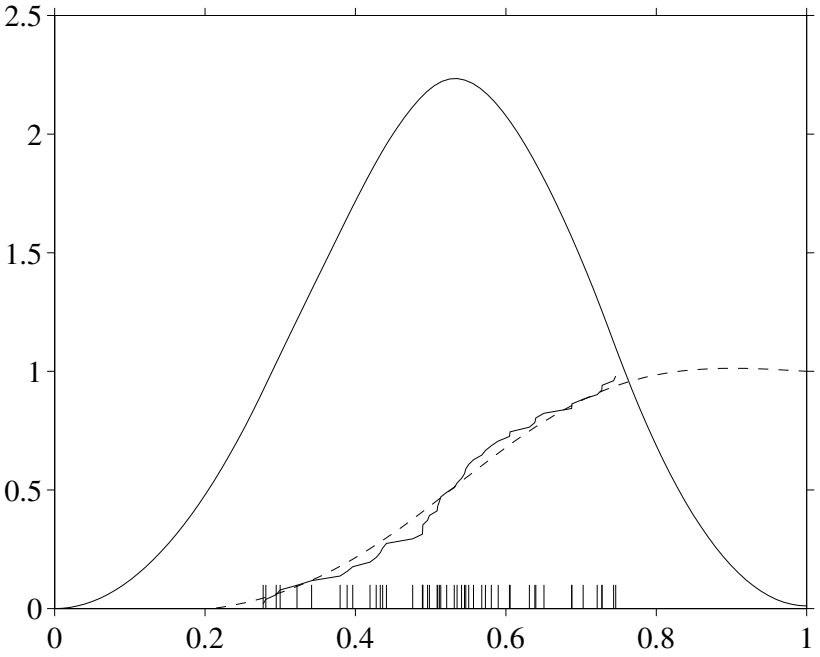

Figure 16.4. Cumulative histogram (solid) and regularized fit (dashed), for the points shown along the axis drawn from a Gaussian distribution. Differentiating gives the resulting density estimate.

16.3 MIXTURE DENSITY ESTIMATION AND EXPECTATION-MAXIMIZATION

Fitting a function lets us generalize our density estimate away from measured points, but it unfortunately does not handle many other kinds of generalization very well. The first problem is that it can be hard to express local beliefs with a global prior. For

example, if a smooth curve has some discontinuities, then a smoothness prior will round out the discontinuities, and a maximum entropy prior will fit the discontinuity but miss the smoothness (Figure 16.5). What's needed is a way to express a statement like "the density is smooth everywhere, except for where it isn't."

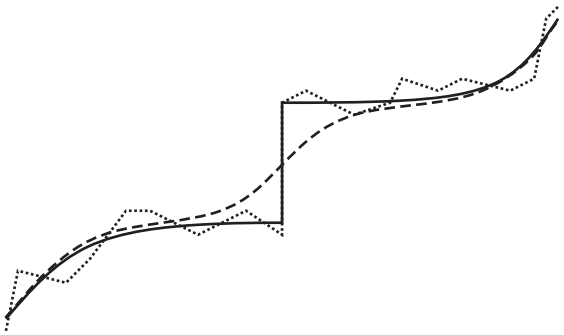

Figure 16.5. The problem with global regularization. A smoothness prior (dashed line) misses the discontinuty in the data (solid line); a maximum entropy prior (dotted) misses the smoothness.

Another problem is with the kinds of functions that need to be represented. Consider points distributed on a low-dimensional surface in a high-dimensional space. Transverse to the surface the distribution is very narrow, something that is hard to expand with trigonometric or polynomial basis functions.

These problems with capturing local behavior suggest that density estimation should be done using local rather than global functions. *Kernel density estimation* [Silverman, 1986] does this by placing some kind of smooth bump, such as a Gaussian, on each data point. An obvious disadvantage of this approach is that the resulting model requires retaining all of the data. A better approach is to find interesting places to put a smaller number of local functions that can model larger neighborhoods. This is done by *mixture models* [McLachlan & Basford, 1988], which are closely connected to the problem of splitting up a data set by *clustering*, and are an example of *unsupervised learning*. Unlike function fitting with a known target, the algorithm must learn for itself where the interesting places in the data set are.

In D dimensions a mixture model can be written by factoring the density over multivariate Gaussians

$$\begin{aligned} p(\vec{x}) &= \sum_{m=1}^{M} p(\vec{x}, c_m) \\ &= \sum_{m=1}^{M} p(\vec{x}|c_m)\ p(c_m) \\ &= \sum_{m=1}^{M} \frac{|\mathbf{C}_m^{-1}|^{1/2}}{(2\pi)^{D/2}} e^{-(\vec{x}-\vec{\mu}_m)^T \cdot \mathbf{C}_M^{-1} \cdot (\vec{x}-\vec{\mu}_m)/2}\ p(c_m) \quad , \end{aligned} \tag{16.15}$$

where $|\cdot|^{1/2}$ is the square root of the determinant, and c_m refers to the mth Gaussian with mean $\vec{\mu}_m$ and covariance matrix $\mathbf{C}_m$. The challenge of course is to find these parameters.

If we had a single Gaussian, the mean value $\vec{\mu}$ could be estimated simply by averaging the data,

$$\begin{aligned} \vec{\mu} &= \int_{-\infty}^{\infty} \vec{x}\ p(\vec{x})\ d\vec{x} \\ &\approx \frac{1}{N}\sum_{n=1}^{N} \vec{x}_n \quad . \end{aligned} \tag{16.16}$$

The second line follows because an integral over a density can be approximated by a sum over variables drawn from the density; we don't know the density but by definition it is the one our data set was taken from. This idea can be extended to more Gaussians by recognizing that the mth mean is the integral with respect to the conditional distribution,

$$\begin{aligned} \vec{\mu}_m &= \int \vec{x}\ p(\vec{x}|c_m)\ d\vec{x} \\ &= \int \vec{x}\ \frac{p(c_m|\vec{x})}{p(c_m)} p(\vec{x})\ d\vec{x} \\ &\approx \frac{1}{Np(c_m)} \sum_{n=1}^{N} \vec{x}_n\ p(c_m|\vec{x}_n) \quad . \end{aligned} \tag{16.17}$$

Similarly, the covariance matrix could be found from

$$\mathbf{C}_m \approx \frac{1}{Np(c_m)} \sum_{n=1}^{N} (\vec{x}_n - \vec{\mu}_m)(\vec{x}_n - \vec{\mu}_m)^T\ p(c_m|x_n) \quad , \tag{16.18}$$

and the expansion weights by

$$\begin{aligned} p(c_m) &= \int_{-\infty}^{\infty} p(\vec{x}, c_m)\ d\vec{x} \\ &= \int_{-\infty}^{\infty} p(c_m|\vec{x})\ p(\vec{x})\ d\vec{x} \\ &\approx \frac{1}{N}\sum_{n=1}^{N} p(c_m|\vec{x}_n) \quad . \end{aligned} \tag{16.19}$$

But how do we find the posterior probability $p(c_m|\vec{x})$ used in these sums? By definition it is

$$\begin{aligned} p(c_m|\vec{x}) &= \frac{p(\vec{x}, c_m)}{p(\vec{x})} \\ &= \frac{p(\vec{x}|c_m)\ p(c_m)}{\sum_{m=1}^{M} p(\vec{x}|c_m)\ p(c_m)} \quad , \end{aligned} \tag{16.20}$$

which we can calculate using the definition (equation 16.15). This might appear to be circular reasoning, and it is! The probabilities of the points can be calculated if we know the parameters of the distributions (means, variances, weights), and the parameters can be found if we know the probabilities. Since we start knowing neither, we can start with a random guess for the parameters and go back and forth, iteratively updating the

probabilities and then the parameters. Calculating an expected distribution given parameters, then finding the most likely parameters given a distribution, is called *Expectation-Maximization* (*EM*), and it converges to the maximum likelihood distribution starting from the initial guess [Dempster *et al*., 1977].

To see where this magical property comes from let's take the log-likelihood L of the data set (the log of the product of the probabilities to see each point) and differentiate with respect to the mean of the mth Gaussian:

$$\begin{aligned}
\nabla_{\vec{\mu}_m} L &= \nabla_{\vec{\mu}_m} \log \prod_{n=1}^{N} p(\vec{x}_n) \\
&= \sum_{n=1}^{N} \nabla_{\vec{\mu}_m} \log p(\vec{x}_n) \\
&= \sum_{n=1}^{N} \frac{1}{p(\vec{x}_n)} \nabla_{\vec{\mu}_m} p(\vec{x}_n) \\
&= \sum_{n=1}^{N} \frac{1}{p(\vec{x}_n)} p(\vec{x}_n, c_m) \mathbf{C}_m^{-1} \cdot (\vec{x}_n - \vec{\mu}_m) \\
&= \sum_{n=1}^{N} p(c_m|\vec{x}_n) \mathbf{C}_m^{-1} \cdot \vec{x}_n - \sum_{n=1}^{N} p(c_m|\vec{x}_n) \mathbf{C}_m^{-1} \cdot \vec{\mu}_m \\
&= N p(c_m) \mathbf{C}_m^{-1} \cdot \left[\frac{1}{N p(c_m)} \sum_{n=1}^{N} \vec{x}_n \; p(c_m|\vec{x}_n) - \vec{\mu}_m \right] \quad . \qquad (16.21)
\end{aligned}$$

Writing the change in the mean after one EM iteration as $\delta\vec{\mu}_m$ and recognizing that the term on the left in the bracket is the update rule for the mean, we see that

$$\delta\vec{\mu}_m = \frac{\mathbf{C}_m}{N p(c_m)} \cdot \nabla_{\vec{\mu}_m} L \quad . \qquad (16.22)$$

Now look back at Section 12.4, where we wrote a gradient descent update step in terms of the gradient of a cost function suitably scaled. In one EM update the mean moves in the direction of the gradient of the log-likelihood, scaled by the Gaussian's covariance matrix divided by the weight. The weight is a positive number, and the covariance matrix is *positive definite* (it has positive eigenvalues [Strang, 1986]), therefore the result is to increase the likelihood. The changes in the mean stop at the maximum when the gradient vanishes. Similar equations apply to the variances and weights [Lei & Jordan, 1996].

This is as close as data analysis algorithms come to a free lunch. We're maximizing a quantity of interest (the log-likelihood) merely by repeated function evaluations. The secret is in equation (16.20). The numerator measures how strongly one Gaussian predicts a point, and the denominator measures how strongly all the Gaussians predict the point. The ratio gives the fraction of the point to be associated with each Gaussian. Each point has one unit of explanatory power, and it gives it out based on how well it is predicted. As a Gaussian begins to have a high probability at a point, that point effectively disappears from the other Gaussians. The Gaussians therefore start exploring the data set in parallel, "eating" points that they explain well and forcing the other Gaussians towards ones they

don't. The interaction among the Gaussians happens in the denominator of the posterior term, collectively performing a high-dimensional search.

The EM cycle finds the local maximum of the likelihood that can be reached from the starting condition, but that usually is not a significant limitation because there are so many equivalent arrangements that are equally good. The Gaussians can for example be started with random means and variances large enough to "feel" the entire data set. Stopping can be determined by the convergence of their positions, or of the likelihood estimage. And because EM is doing gradient ascent on the log-likelihood, all of the lessons from Chapter 15 apply – a filtered updated combining new and old estimates can be included to control the learning rate, and momentum from the change in prior estimates can be included to climb out of local extrema.

16.4 CLUSTER-WEIGHTED MODELING

As appealing as mixture density estimation is, there is still a final problem. Figure 16.6 shows the result of doing EM with a mixture of Gaussians on random data uniformly distributed over the interval [0, 1]. The individual distributions are shown as dashed lines, and the sum by a solid line. The correct answer is a constant, but because that is hard to represent in this basis we get a very bumpy distribution that depends on the precise details of how the Gaussians overlap. In gaining locality we've lost the ability to model simple functional dependence.

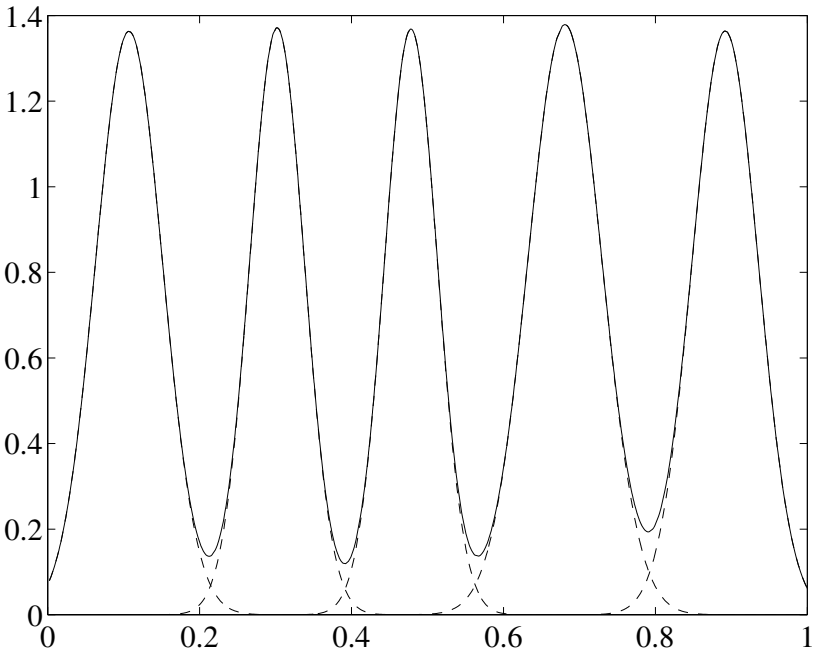

Figure 16.6. Mixture density estimation with Gaussians for data uniformly distributed between 0 and 1.

The problem is that a Gaussian captures only proximity; anything nontrivial must come from the overlap of multiple Gaussians. A better alternative is to base the expansion of a density around models that can locally describe more complex behavior. This powerful idea has re-emerged in many flavors under many names, including *Bayesian networks* [Buntine, 1994], *mixtures of experts* [Jordan & Jacobs, 1994], and *gated experts* [Weigend *et al*., 1995]. The version that we will cover, *cluster-weighted modeling* [Gershenfeld *et al*., 1999], is just general enough to be able to describe many situations, but not so general that it presents difficult architectural decisions.

The goal now is to capture the functional dependence in a system as part of a density estimate. Let's start with a set of N observations $\{y_n, \vec{x}_n\}_{n=1}^N$, where the $\vec{x}_n$ are known inputs and the y_n are measured outputs. y might be the exchange rate between the dollar and the mark, and $\vec{x}$ a group of currency indicators. Or y could be a future value of a signal and $\vec{x}$ a vector of past lagged values. For simplicity we'll take y to be a scalar, but the generalization to vector $\vec{y}$ is straightforward.

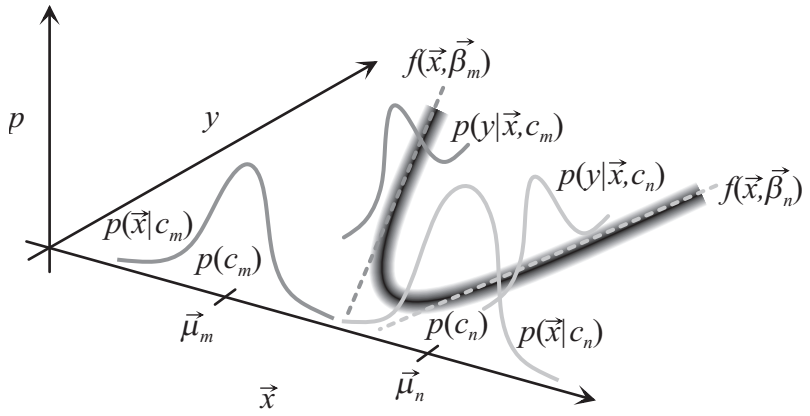

Figure 16.7. The spaces in cluster-weighted modeling.

Given the joint density $p(y, \vec{x})$ we could find any derived quantity of interest, such as a conditional forecast $\langle y|\vec{x}\rangle$. We'll proceed by expanding the density again, but now in terms of explanatory clusters that contain three terms: a weight $p(c_m)$, a domain of influence in the input space $p(\vec{x}|c_m)$, and a dependence in the output space $p(y|\vec{x}, c_m)$:

$$\begin{aligned} p(y,\vec{x}) &= \sum_{m=1}^{M} p(y,\vec{x},c_m) \\ &= \sum_{m=1}^{M} p(y,\vec{x}|c_m)\; p(c_m) \\ &= \sum_{m=1}^{M} p(y|\vec{x},c_m)\; p(\vec{x}|c_m)\; p(c_m) \quad . \end{aligned} \tag{16.23}$$

These terms are shown in Figure 16.7. As before, $p(c_m)$ is a number that measures the fraction of the data set explained by the cluster. The input term could be taken to be a

D-dimensional separable Gaussian,

$$p(\vec{x}|c_m) = \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi\sigma_{m,d}^2}} e^{-(x_d-\mu_{m,d})^2/2\sigma_{m,d}^2} \quad , \tag{16.24}$$

or one using the full covariance matrix,

$$p(\vec{x}|c_m) = \frac{|\mathbf{C}_m^{-1}|^{1/2}}{(2\pi)^{D/2}} e^{-(\vec{x}-\vec{\mu}_m)^T \cdot \mathbf{C}_m^{-1} \cdot (\vec{x}-\vec{\mu}_m)/2} \quad . \tag{16.25}$$

Separable Gaussians require storage and computation linear in the dimension of the space while nonseparable ones require quadratic storage and a matrix inverse at each step. Conversely, using the covariance matrix lets one cluster capture a linear relationship that would require many separable clusters to describe. Therefore covariance clusters are preferred in low-dimensional spaces, but cannot be used in high-dimensional spaces.

The output term will also be taken to be a Gaussian, but with a new twist:

$$p(y|\vec{x}, c_m) = \frac{1}{\sqrt{2\pi\sigma_{m,y}^2}} e^{-[y-f(\vec{x},\vec{\beta}_m)]^2/2\sigma_{m,y}^2} \quad . \tag{16.26}$$

The mean of the Gaussian is now a function f that depends on $\vec{x}$ and a set of parameters $\vec{\beta}_m$. The reason for this can be seen by calculating the conditional forecast

$$\begin{aligned} \langle y|\vec{x}\rangle &= \int y\ p(y|\vec{x})\ dy \\ &= \int y\ \frac{p(y,\vec{x})}{p(\vec{x})}\ dy \\ &= \frac{\sum_{m=1}^{M} \int y\ p(y|\vec{x},c_m)\ dy\ p(\vec{x}|c_m)\ p(c_m)}{\sum_{m=1}^{M} p(\vec{x}|c_m)\ p(c_m)} \\ &= \frac{\sum_{m=1}^{M} f(\vec{x},\vec{\beta}_m)\ p(\vec{x}|c_m)\ p(c_m)}{\sum_{m=1}^{M} p(\vec{x}|c_m)\ p(c_m)} \quad . \end{aligned} \tag{16.27}$$

We see that in the forecast the Gaussians are used to control the interpolation among the local functions, rather than directly serving as the basis for functional approximation. This means that f can be chosen to reflect a prior belief on the local relationship between $\vec{x}$ and y, for example locally linear, and even one cluster is capable of modeling this behavior. Using locally linear models says nothing about the global smoothness because there is no constraint that the clusters need to be near each other, so this architecture can handle the combination of smoothness and discontinuity posed in Figure 16.5.

Equation (16.27) looks like a kind of network model, but it is a derived property of a more general density estimate rather than an assumed form. We can also predict other quantities of interest such as the error in our forecast of y,

$$\begin{aligned} \langle \sigma_y^2|\vec{x}\rangle &= \int (y - \langle y|\vec{x}\rangle)^2\ p(y|\vec{x})\ dy \\ &= \int (y^2 - \langle y|\vec{x}\rangle^2)\ p(y|\vec{x})\ dy \end{aligned}$$

$$= \frac{\sum_{m=1}^{M} [\sigma_{m,y}^2 + f(\vec{x}, \vec{\beta}_m)^2] \; p(\vec{x}|c_m) \; p(c_m)}{\sum_{m=1}^{M} p(\vec{x}|c_m) \; p(c_m)} - \langle y|\vec{x}\rangle^2. \tag{16.28}$$

Note that the use of Gaussian clusters does not assume a Gaussian error model; there can be multiple output clusters associated with one input value to capture multimodal or other kinds of non-Gaussian distributions.

The estimation of the parameters will follow the EM algorithm we used in the last section. From the forward probabilities we can calculate the posteriors

$$\begin{aligned} p(c_m|y, \vec{x}) &= \frac{p(y, \vec{x}, c_m)}{p(y, \vec{x})} \\ &= \frac{p(y|\vec{x}, c_m) \; p(\vec{x}|c_m) \; p(c_m)}{\sum_{m=1}^{M} p(y, \vec{x}, c_m)} \\ &= \frac{p(y|\vec{x}, c_m) \; p(\vec{x}|c_m) \; p(c_m)}{\sum_{m=1}^{M} p(y|\vec{x}, c_m) \; p(\vec{x}|c_m) \; p(c_m)} \end{aligned} \tag{16.29}$$

which are used to update the cluster weights

$$\begin{aligned} p(c_m) &= \int p(y, \vec{x}, c_m) \; dy \; d\vec{x} \\ &= \int p(c_m|y, \vec{x}) \; p(y, \vec{x}) \; dy \; d\vec{x} \\ &\approx \frac{1}{N} \sum_{n=1}^{N} p(c_m|y_n, \vec{x}_n) \quad . \end{aligned} \tag{16.30}$$

A similar calculation is used to find the new means,

$$\begin{aligned} \vec{\mu}_m^{new} &= \int \vec{x} \; p(\vec{x}|c_m) \; d\vec{x} \\ &= \int \vec{x} \; p(y, \vec{x}|c_m) \; dy \; d\vec{x} \\ &= \int \vec{x} \; \frac{p(c_m|y, \vec{x})}{p(c_m)} \; p(y, \vec{x}) \; dy \; d\vec{x} \\ &\approx \frac{1}{N \; p(c_m)} \sum_{n=1}^{N} \vec{x}_n \; p(c_m|y_n, \vec{x}_n) \\ &= \frac{\sum_{n=1}^{N} \vec{x}_n \; p(c_m|y_n, \vec{x}_n)}{\sum_{n=1}^{N} p(c_m|y_n, \vec{x}_n)} \\ &\equiv \langle \vec{x} \rangle_m \end{aligned} \tag{16.31}$$

(where the last line defines the cluster-weighted expectation value). The second line introduces y as a variable that is immediately integrated over, an apparently meaningless step that in fact is essential. Because we are using the stochastic sampling trick to evaluate the integrals and guide the cluster updates, this permits the clusters to respond to both where data are in the input space and how well their models work in the output space. A cluster won't move to explain nearby data if they are better explained by another cluster's model, and if two clusters' models work equally well then they will separate to better explain where the data are.

The cluster-weighted expectations are also used to update the variances

$$\sigma_{m,d}^{2,new} = \langle (x_d - \mu_{m,d})^2 \rangle_m \tag{16.32}$$

or covariances

$$[\mathbf{C}_m]_{ij}^{new} = \langle (x_i - \mu_i)(x_j - \mu_j) \rangle_m \quad . \tag{16.33}$$

The model parameters are found by choosing the values that maximize the cluster-weighted log-likelihood:

$$\begin{aligned}
0 &= \frac{\partial}{\partial \vec{\beta}_m} \log \prod_{n=1}^{N} p(y_n, \vec{x}_n) \\
&= \sum_{n=1}^{N} \frac{\partial}{\partial \vec{\beta}_m} \log p(y_n, \vec{x}_n) \\
&= \sum_{n=1}^{N} \frac{1}{p(y_n, \vec{x}_n)} \frac{\partial p(y_n, \vec{x}_n)}{\partial \beta_m} \\
&= \sum_{n=1}^{N} \frac{1}{p(y_n, \vec{x}_n)} p(y_n, \vec{x}_n, c_m) \frac{y_n - f(\vec{x}_n, \vec{\beta}_m)}{\sigma_{m,y}^2} \frac{\partial f(\vec{x}_n, \vec{\beta}_m)}{\partial \vec{\beta}_m} \\
&= \frac{1}{\sigma_{m,y}^2} \sum_{n=1}^{N} p(c_m | y_n, \vec{x}_n)[y_n - f(\vec{x}_n, \vec{\beta}_m)] \frac{\partial f(\vec{x}_n, \vec{\beta}_m)}{\partial \vec{\beta}_m} \\
&= \frac{1}{N p(c_m)} \sum_{n=1}^{N} p(c_m | y_n, \vec{x}_n)[y_n - f(\vec{x}_n, \vec{\beta}_m)] \frac{\partial f(\vec{x}_n, \vec{\beta}_m)}{\partial \vec{\beta}_m} \\
&= \left\langle [y - f(\vec{x}, \vec{\beta})] \frac{\partial f(\vec{x}, \vec{\beta}_m)}{\partial \vec{\beta}_m} \right\rangle_m \quad .
\end{aligned} \tag{16.34}$$

In the last few lines remember that since the left hand side is zero we can multiply or divide the right hand side by a constant.

If we choose a local model that has linear coefficients,

$$f(\vec{x}, \vec{\beta}_m) = \sum_{i=1}^{I} \beta_{m,i} f_i(\vec{x}) \quad , \tag{16.35}$$

then this gives for the jth coefficient

$$\begin{aligned}
0 &= \langle [y - f(\vec{x}, \vec{\beta}_m)] f_j(\vec{x}) \rangle_m \\
&= \underbrace{\langle y f_j(\vec{x}) \rangle_m}_{a_j} - \sum_{i=1}^{I} \beta_{m,i} \underbrace{\langle f_j(\vec{x}) f_i(\vec{x}) \rangle_m}_{B_{ji}} \\
\Rightarrow \vec{\beta}_m &= B^{-1} \cdot \vec{a} \quad .
\end{aligned} \tag{16.36}$$

Finally, once we've found the new model parameters then the new output width is

$$\sigma_{m,y}^{2,new} = \langle [y - f(\vec{x}, \vec{\beta}_m)]^2 \rangle_m \quad . \tag{16.37}$$

It's a good idea to add a small constant to the input and output variance estimates

to prevent a cluster from shrinking down to zero width for data with no functional dependence; this constant reflects the underlying resolution of the data.

Iterating this sequence of evaluting the forward and posterior probabilities at the data points, then maximizing the likelihood of the parameters, finds the most probable set of parameters that can be reached from the starting configuration. The fundamental algorithm parameter is M, the number of clusters. This controls overfitting, which can be determined by cross-validation, and by adaptively pruning small clusters and dividing large ones. The other essential choice to make is f, the form of the local model. This can be based on past practice for a domain, and should be simple enough that the local model cannot overfit. It's even possible to include more than one type of local model and let the clustering find where they are most relevant.

Cluster-weighted modeling then assembles the local models into a global model that can handle nonlinearity (by the nonlinear interpolation among local models), discontinuity (since nothing forces clusters to stay near each other), non-Gaussianity (by the overlap of multiple clusters), nonstationarity (by giving absolute time as an input variable so that clusters can grow or shrink as needed to find the locally stationary time scales), find low-dimensional structure in high-dimensional spaces (models are allocated only where there are data to explain), predict not only errors from the width of the output distribution but also errors that come from forecasting away from where training observations were made (by using the input density estimate), build on experience (by reducing to a familiar global version of the local model in the limit of one cluster), and the resulting model has a transparent architecture (the parameters are easy to interpret, and by definition the clusters find "interesting" places to represent the data) [Gershenfeld *et al*., 1999].

Taken together those specifications are a long way from the simple models we used at the outset of the study of function fitting. Looking beyond cluster-weighted modeling, it's possible to improve on the EM search by doing a full Bayesian integration over prior distributions on all the parameters [Richardson & Green, 1997], but this generality is not needed for the common case of weak priors on the parameters and many equally acceptable solutions. And the model can be further generalized to arbitrary probabilistic networks [Buntine, 1996], a step that is useful if there is some *a priori* insight into the architecture.

An important special case of cluster-weighted modeling is observables that are restricted to a discrete set of values such as events, patterns, or conditions. Let's now let $\{s_n, \vec{x}_n\}_{n=1}^N$ be the training set of pairs of input points $\vec{x}$ and discrete output states s. Once again we'll factor the joint density over clusters

$$p(s, \vec{x}) = \sum_{m=1}^{M} p(s|\vec{x}, c_m)\, p(\vec{x}|c_m)\, p(c_m) \quad , \tag{16.38}$$

but now the output term is simply a histogram of the probability to see each state for each cluster (with no explicit $\vec{x}$ dependence). The histogram is found by generalizing the simple binning we used at the beginning of the chapter to the cluster-weighted expectation of a delta function that equals one on points with the correct state and zero otherwise:

$$p(s|\vec{x}, c_m) = \frac{1}{N p(c_m)} \sum_{n=1}^{N} \delta_{s_n=s} p(c_m|s_n, \vec{x}_n)$$

$$= \frac{1}{Np(c_m)} \sum_{n|s_n=s} p(c_m|s_n, \vec{x}_n) \quad . \tag{16.39}$$

Everything else is the same as the real-valued output case. An example is shown in Figure 16.8, for two states s (labelled "1" and "2"). The clusters all started equally likely to predict either state, but after convergence they have specialized in one of the data types as well as dividing the space.

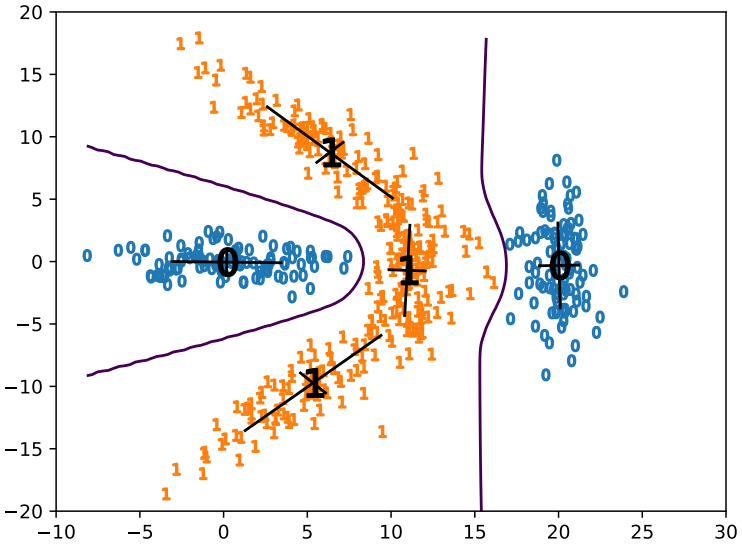

Figure 16.8. Cluster-weighted modeling with discrete output states. The small numbers are the events associated with each point, the cluster covariances are shown along with the most likely event associated with each cluster, and the lines are the decision boundaries.

This model can predict the probability to see each state given a new point by

$$p(s|\vec{x}) = \frac{p(s, \vec{x})}{\sum_s p(s, \vec{x})} \quad . \tag{16.40}$$

For applications where computational complexity or speed is a concern there are easy approximations that can be made. It is possible to use just the cluster variances instead of the full covariance matrix, and to restrict the output models to one state per cluster rather than keeping the complete histogram. If the goal is simply to predict the most likely state rather than the distribution of outcomes, then evaluating the density can be approximated by comparing the square distances to the clusters divided by the variances. Finally, keeping only the cluster means reduces to a linear-time search to find the nearest cluster, $\min_m |\vec{x} - \vec{\mu}_m|$, which defines a *Voronoi tesselation* of the space by the clusters [Preparata & Shamos, 1985].

The final approximation of simply looking up the state of the cluster nearest a new point might appear to not need such a general clustering to start with. A much simpler

algorithm would be to iteratively assign points to the nearest cluster and then move the cluster to the mean of the points associated with it. This is called the *k-means* algorithm [MacQueen, 1967; Hartigan, 1975]. It doesn't work very well, though, because in the iteration a cluster cannot be influenced by a large number of points that are just over the boundary with another cluster. By embedding the cluster states in Gaussian kernels as we have done, this is a *soft clustering* unlike the *hard clustering* done by k-means. The result is usually a faster search and a better model.

Predicting states is part of the domain of *pattern recognition*. State prediction can also be used to approximate a set of inputs by a common output; *vector quantization* applies this to a feature vector describing a signal in order to compress it for transmission and storage [Nasrabadi & King, 1988; Gersho & Gray, 1992]). Alternatively, it can be used to retrieve information associated with related inputs stored in a *content-addressable memory* [Kohonen, 1989; Bechtel & Abrahamsen, 1991]. We'll pick up those problems in Chapter 18.

16.5 SELECTED REFERENCES

[Duda & Hart, 1973] Duda, Richard O., & Hart, Peter E. (1973). *Pattern Classification and Scene Analysis*. New York, NY: Wiley.

[Therrien, 1989] Therrien, Charles W. (1989). *Decision, Estimation, and Classification: An Introduction to Pattern Recognition and Related Topics*. New York, NY: Wiley.

[Fukunaga, 1990] Fukunaga, Keinosuke (1990). *Introduction to Statistical Pattern Recognition*. 2nd edn. Boston, MA: Academic Press.

These are classic pattern recognition texts.

[Silverman, 1986] Silverman, B.W. (1986). *Density Estimation for Statistics and Data Analysis*. New York, NY: Chapman and Hall.

Traditional density estimation.

16.6 PROBLEMS

(15.1) Generate a data set by sampling 100 points from $y = \tanh(x)$ between $x = -5$ to 5, adding random noise with a magnitude of 0.25 to y. Fit it with a cluster-weighted model using a local linear function in the clusters, $y = a + bx$. Plot the data and the resulting forecast $\langle y|x\rangle$, uncertainty $\langle \sigma_y^2|x\rangle$, and cluster probabilities $p(\vec{x}|c_m)p(c_m)$ as the number of clusters is increased (starting with one), and animate their evolution during the EM iterations.

(15.2) Generate a 2D distribution of data labeled with discrete states as shown in Figure 16.8, fit it with a cluster-weighted model, and plot the decision boundaries.