CHAPTER 9

Markov Chain Monte Carlo Sampling

In Chapter 4 we saw that, for logistic regression, it was not possible to analytically obtain the posterior density over the model parameters. We described how to overcome this problem with three alternatives: the MAP solution (a point estimate of the parameters that maximises the posterior), the Laplace approximation (where the posterior is approximated with a Gaussian density) and sampling from the posterior using the Metropolis–Hastings algorithm.

Unfortunately, this situation is not exceptional. The relatively small number of conjugate prior and likelihood pairs means that, in many applications, we will not be able to get an analytical expression for the posterior.

When compared with point estimates and approximations, sampling methods (such as Metropolis–Hastings) have the benefit of providing samples from the true posterior at the expense of more computational work. As computational power has become less of an issue, sampling methods have increased in popularity across many machine learning applications.

For this reason, we feel that our brief introduction to Metropolis–Hastings in Chapter 4 is insufficient, and sampling methods require a chapter all to themselves. We will start by introducing a common alternative to Metropolis–Hastings: Gibbs sampling. Gibbs sampling is very popular, possibly due to the lack of user-tunable parameters, but is restricted to problems where the conditional distribution of each parameter (conditioned on values for all others) is available. After Gibbs, we will spend a little time looking at the theory behind Markov chain Monte Carlo (MCMC) techniques.

Unfortunately, although these methods are often quite easy to implement, getting them to work properly is quite another matter, and we devote a section to some MCMC practicalities. Finally, we will conclude the chapter with a quick tour of a few of the more advanced sampling algorithms that are currently found within machine learning.

9.1 GIBBS SAMPLING

Our motivation for sampling has been cases where we cannot obtain an analytical expression for the posterior density over parameters for a Bayesian model. The samples that we draw can be used to compute the expectations involved in our ultimate goal – making predictions. For the logistic regression model described in Chapter 4, we could compute something proportional to the posterior – the prior multiplied by the likelihood – but could not normalise it. Metropolis–Hastings was appropriate because we only ever required the ratio of the posterior value for two different parameter values, and in the ratio the normalising constant canceled.

In some cases, we cannot compute the posterior but we can compute the conditional densities of each parameter conditioned on the others. For example, consider a model with M parameters $\boldsymbol{\theta} = [\theta_1, \ldots, \theta_M]^\mathsf{T}$. If we observe some data $\mathbf{X}$, the posterior is given by

$$p(\boldsymbol{\theta}|\mathbf{X}) = \frac{p(\mathbf{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathbf{X})},$$

which we assume that we cannot compute. However, we *can* compute the conditional densities for each parameter conditioned on values for all of the other parameters:

$$\begin{gathered} p(\theta_1|\theta_2, \ldots, \theta_N, \mathbf{X}) \\ p(\theta_2|\theta_1, \theta_3, \ldots, \theta_N, \mathbf{X}) \\ \vdots \\ p(\theta_N|\theta_1, \ldots, \theta_{N-1}, \mathbf{X}). \end{gathered}$$

This might appear to be an unlikely situation – computing all of these things looks harder than computing the posterior itself! But in many cases the conditional distributions are straightforward to compute and sample from even when the posterior is inaccessible. In these situations Gibbs sampling can be used.

Armed with the conditional distributions, **Gibbs sampling** is very straightforward. For notational convenience, we will assume we are sampling from $p(\boldsymbol{\theta})$ and drop the dependence on $\mathbf{X}$. We start with a guess of the values of the parameters, $\boldsymbol{\theta}^0 = [\theta_1^0, \ldots, \theta_M^0]^\mathsf{T}$. When using Gibbs sampling to sample from a posterior over parameters, a sample from the prior would be sensible. We then resample each parameter conditioned on the most recent values of all other parameters. Assuming we go through the parameters in numerical order (we don't have to but it makes it easier to draw), this process is depicted in Figure 9.1. We start by sampling a new value for θ_1 (θ_1^1). The most recent values of all of the other parameters are $\theta_2^0, \ldots, \theta_M^0$ and so we condition on these. Now we move to θ_2. The most recent value of θ_1 is now θ_1^1 and the most recent values of the other parameters are $\theta_3^0, \ldots, \theta_3^M$ so we condition on $\theta_1^1, \theta_3^0, \ldots, \theta_M^0$. We continue until we have resampled all M parameters, at which point the most recent values will be $\boldsymbol{\theta}^1 = [\theta_1^1, \ldots, \theta_M^1]$. $\boldsymbol{\theta}^1$ represents our first sample from $p(\boldsymbol{\theta})$. The process is now repeated as many times as we wish to draw samples. We call each loop through all of the individual parameters a cycle and each cycle produces one sample from $p(\boldsymbol{\theta})$.

We will illustrate Gibbs sampling by demonstrating how it can be used to sample from a two-dimensional Gaussian (MATLAB script: `gibbsgauss.m`). Clearly, we can sample from multivariate Gaussians directly (we did it a lot when dealing with Gaussian processes in Chapter 8) but it helps to illustrate the concept of Gibbs

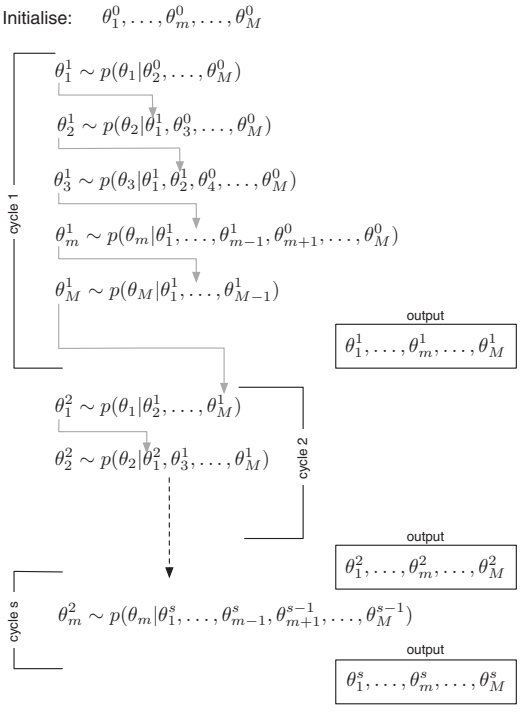


FIGURE 9.1 A schematic of the Gibbs sampling process. Parameters are repeatedly resampled conditioned on the current values of all other parameters.

sampling before we progress to something more realistic where direct sampling is not an option.

Consider the Gaussian shown shown in Figure 9.2. It has mean and covariance given by

$$\boldsymbol{\mu} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 2 \end{bmatrix}. \tag{9.1}$$

Because this is a Gaussian density, we can analytically obtain the conditional densities for x_1 (conditioned on x_2) and x_2 conditioned on x_1 (see Comment 8.2):

$$\begin{aligned} p(x_1|x_2, \boldsymbol{\mu}, \boldsymbol{\Sigma}) &= \mathcal{N}(\mu_1, \sigma_1^2) \qquad (9.2) \\ \mu_1 = 1 + \frac{0.8}{2}(x_2 - 2) &\qquad \sigma_1^2 = 1 - \frac{0.8^2}{2} \\ p(x_2|x_1, \boldsymbol{\mu}, \boldsymbol{\Sigma}) &= \mathcal{N}(\mu_2, \sigma_2^2) \qquad (9.3) \\ \mu_2 = 1 + \frac{0.8}{1}(x_1 - 1) &\qquad \sigma_1^2 = 2 - \frac{0.8^2}{1} \end{aligned}$$

We start with the guesses of our two parameters, θ_1^0 and θ_2^0. We then resample

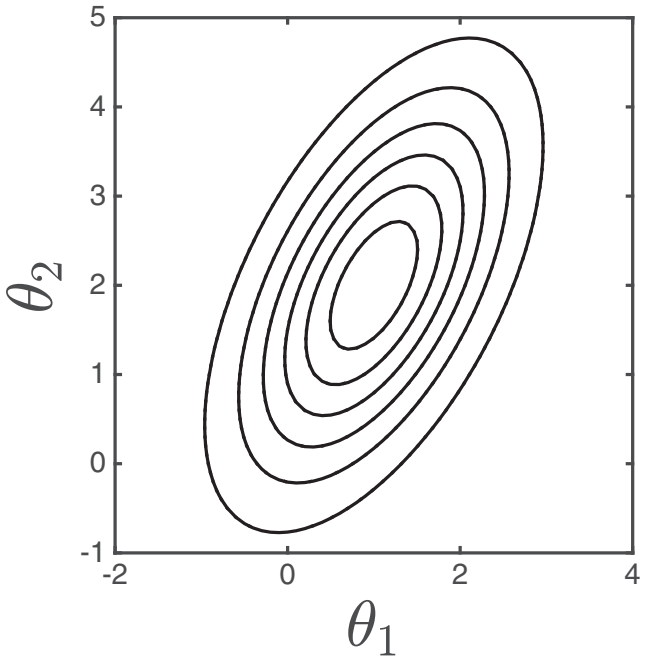
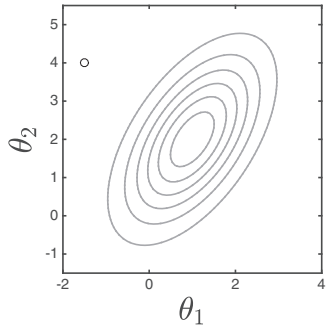


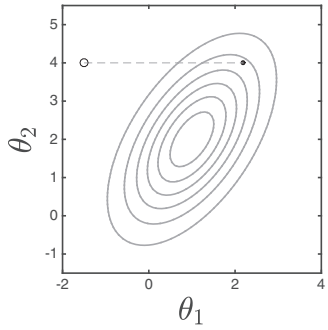
FIGURE 9.2 A 2D Gaussian density.

each parameter in turn, conditioning on the current value of the other parameters. So we first sample a new value for θ_1 (θ_1^1) by conditioning on $\theta_2 = \theta_2^0$. Then we sample θ_2 (θ_2^1) by conditioning on $\theta_1 = x_1^1$. θ_1 is then resampled to give θ_1^2 by conditioning on $\theta_2 = \theta_2^1$, etc. Once we are confident that the effects of the random initialisation have been removed (more on this later), the samples, θ_1^s, θ_2^s are samples from $p(\theta_1, \theta_2|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Note that the order in which the parameters are updated doesn't matter, as long as every sample is conditioned on the most recent values of all other parameters. In practice, we often randomise the order of parameters in each cycle.

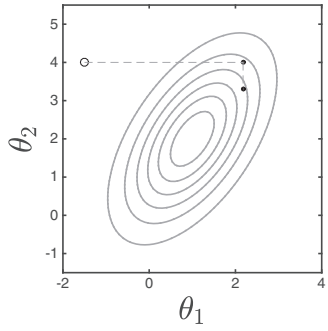
Figure 9.3 shows Gibbs sampling in action for our 2D Gaussian example where the required conditional densities are given in Equations 9.2 and 9.3 (MATLAB script: `gibbs_gaussian.m`). We start with an initial guess, $\boldsymbol{\theta}^0 = [-1.5, 4]^\mathsf{T}$ (shown by the open circle). In the first step, we sample a new value of θ_1 from Equation 9.2, shown by the small solid circle. Based on this new value for θ_1, we sample a new value of θ_2 using Equation 9.3. This is one cycle of the sampler and we have transitioned from our initial value $\boldsymbol{\theta}^0$ to a new value $\boldsymbol{\theta}^1$. In Figure 9.3(d) we see the effect of 2 cycles (4 updates; 2 for each parameter) and in Figures 9.3(e) and 9.3(f) the state after 5 and 100 cycles, respectively. Visually, the samples look like they are coming from the full 2-dimensional Gaussian (shown by the contours). To check more objectively, we can calculate the mean and covariance of the samples, and compare



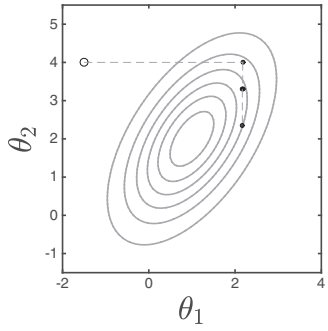
(a) The Gaussian contours and initial value, $\mathbf{x}^0$.



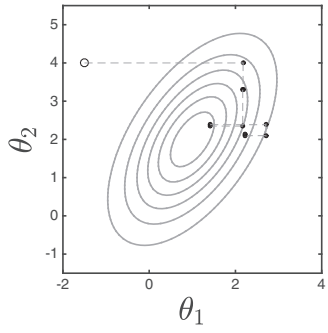
(b) θ_1 is updated by sampling from $p(\theta_1|\theta_2^0, \ldots)$ to give θ_1^1.



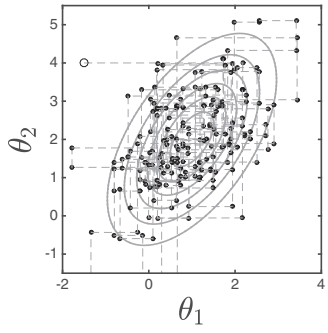
(c) θ_2 is updated by sampling from $p(\theta_2|\theta_1^1, \ldots)$ to give θ_2^1.



(d) The first two full cycles of the sampler.



(e) The first five full cycles.



(f) The first 100 full cycles.

FIGURE 9.3 Gibbs sampling from a 2D Gaussian.

them with the true values. After 100 samples, the sample mean and covariance are

$$\boldsymbol{\mu}^{100} = \begin{bmatrix} 0.93 \\ 2.11 \end{bmatrix} \quad \boldsymbol{\Sigma}^{100} = \begin{bmatrix} 0.92 & 0.74 \\ 0.74 & 2.30 \end{bmatrix}.$$

These are similar to the true values given in Equation 9.1. Are they close enough? It's hard to say, but, if the scheme is working, we ought to get closer to the true values the more samples we take. Below are the values after 1000 samples, which are now much closer to the true values.

$$\boldsymbol{\mu}^{1000} = \begin{bmatrix} 0.95 \\ 1.92 \end{bmatrix} \quad \boldsymbol{\Sigma}^{1000} = \begin{bmatrix} 1.00 & 0.80 \\ 0.80 & 2.05 \end{bmatrix}.$$

As well as samples from the joint distribution, if we only look at the samples generated for θ_1 (or θ_2), we have samples from the marginal density $p(\theta_1)$ (or $p(\theta_2)$). This is a useful property that we will use in the following, more realistic example (see Exercise 9.1).

9.2 EXAMPLE: GIBBS SAMPLING FOR GP CLASSIFICATION

The two-dimensional Gaussian is a nice density to introduce Gibbs sampling, but we could, if we wanted to, sample from the Gaussian directly. We will now look at a more realistic example where we cannot directly sample from the distribution of interest but do have access to all of the conditionals.

The example we will use is binary Gaussian process (GP) classification using a probit likelihood function. GP classification was introduced in Chapter 8, where we used point and Laplace approximations to overcome the non-conjugacy of the GP prior and the logistic likelihood. Here we will swap the logistic likelihood for the probit likelihood and demonstrate how, using the auxiliary variable trick introduced in Chapter 7, we can perform inference using Gibbs sampling.

Assume that we observe N training examples, each consisting of a one-dimensional observation x_n and a binary target value $t_n \in \{0, 1\}$ (extension to multivariate input data is straightforward). We assume that the observed targets were generated by pushing a real value (f_n) through a squashing function to give a probability and then sampling a value of 1 or 0 according to this probability. Rather than the logistic squashing function used in Chapter 8, we will use the probit function (see Section 7.7)

$$P(t_n = 1|f_n) = \phi(f_n) = \int_{-\infty}^{f_n} \mathcal{N}(y|0, 1)\, dy,$$

which is the probability that a random variable from a standard Gaussian density is below f_n.

Recall from Chapter 8 that performing GP classification has three steps:

1. computing the posterior density over the latent function (f_n),
2. using this posterior to perform a GP regression to predict the function values at test points,
3. pushing the predicted latent function values through the squashing function to produce predictive posteriors.

Placing all of the observations $x_1, \ldots, x_N$ into $\mathbf{X}$, all of the labels t_n into $\mathbf{t}$ and the N latent function values f_n into $\mathbf{f}$, we can begin to think about a sampling scheme. Using the superscript $*$ to denote predictive quantities, here is one possible cycle (that would be repeated S times to generate S samples):

- Sample $\mathbf{f}^s$ from $p(\mathbf{f}|\mathbf{t}, \mathbf{X})$ (sample from the posterior density over the latent function).
- Sample $\mathbf{f}^{*,s}$ from a GP regression based on $\mathbf{f}^s$.
- Sample $\mathbf{t}^{*,s}$ from the probabilities provided by squashing $\mathbf{f}^{*,s}$ through the probit function.

The final predictive probabilities could then be computed by averaging over the S values of $\mathbf{t}^{*,s}$, i.e., for each test point, the probability that it is in class 1 is the proportion of samples its t^* value was equal to 1 and not 0.

The problem with this scheme lies in the first step which requires samples from the posterior over the latent function values. We cannot sample from this density directly because of the non-conjugacy of the GP prior and the probit likelihood. However, using the same scheme that is introduced in Section 7.7, we can introduce a set of auxiliary variables, $\mathbf{y}$, that sit between the latent function and the labels such that we can sample directly from the conditional densities $p(\mathbf{y}|\mathbf{f}, \mathbf{t})$ and $p(\mathbf{f}|\mathbf{y}, \mathbf{X})$ and obtain the samples of $\mathbf{f}$ as required.

To do this, we first rearrange the probit likelihood:

$$\begin{aligned} P(t_n = 1|f_n) &= \int_{-\infty}^{f_n} \mathcal{N}(y|0,1)\, dy \\ &= \int_{-\infty}^{0} \mathcal{N}(y| - f_n, 1)\, dy \\ &= \int_{0}^{\infty} \mathcal{N}(y|f_n, 1)\, dy \\ &= \int_{-\infty}^{\infty} \delta(y > 0)\mathcal{N}(y|f_n, 1)\, dy, \end{aligned} \tag{9.4}$$

where $\delta(y > 0)$ is 1 if $y > 0$ and 0 otherwise. Now, consider the following model:

$$\begin{aligned} P(t_n = 1|y_n) &= \delta(y_n > 0) \\ p(y_n|f_n) &= \mathcal{N}(y_n|f_n, 1) \\ p(t_n = 1, y_n|f_n) &= \delta(y_n > 0)\mathcal{N}(y_n|f_n, 1). \end{aligned} \tag{9.5}$$

The probit likelihood is therefore equivalent to this extended model with the y_n variables integrated out (Equation 9.4 is just Equation 9.5 integrated over all possible values of y).

As with the probabilistic PCA model in Section 7.7, if we keep the auxiliary variables (y_n) in the model, inference becomes more straightforward. In the PCA example, it enabled us to perform Variational Bayes inference, and in this case it will allow us to use Gibbs sampling. The final definition we need is $P(t_n = 0|y_n)$. Because $P(t_n = 0|y_n) + P(t_n = 1|y_n) = 1$, this has to be equal to $\delta(y_n < 0)$.

Our Gibbs sampling procedure becomes

- Sample $\mathbf{y}^s$ from $p(\mathbf{y}|\mathbf{f}^{s-1}, \mathbf{t})$

- Sample $\mathbf{f}^s$ from $p(\mathbf{f}|\mathbf{y}^s, \mathbf{X})$
- Sample $\mathbf{f}^{*,s}$ from a GP regression based on $\mathbf{f}^s$
- Sample $\mathbf{t}^{*,s}$ from the probabilities provided by squashing $\mathbf{f}^{*,s}$ through a probit function,

and compute the final predictive probabilities by averaging over the $\mathbf{t}^{*,s}$. Note that we could make the predictions by sampling auxiliary variables for the test points, $\mathbf{y}^*$, and then assigning the elements of $\mathbf{t}^*$ depending on whether the particular components of y^* are positive or negative. That is, for each test point, we sample from $\mathcal{N}(y^*|f^{*,s}, 1)$ and then assign $t^{*,s} = 1$ if $y^* > 0$ and $t^{*,s} = 0$ otherwise. The two procedures are identical for exactly the reason we can perform the auxiliary variable trick in the first place – as far as t is concerned, the two processes (marginalising y or not) are statistically identical.

The inclusion of the auxiliary variables $\mathbf{y}$ has allowed us to use Gibbs sampling to sample from the joint density $p(\mathbf{f}, \mathbf{y}|\mathbf{X}, \mathbf{t})$ by repeatedly sampling $\mathbf{f}$ conditioned on $\mathbf{y}$ and $\mathbf{y}$ conditioned on $\mathbf{f}$. Recall that, in the last section, we claimed that, if we only look at the samples generated for one of the parameters, these are samples from the marginal density of that parameter. In this example, the marginal density for $\mathbf{f}$ is

$$\int p(\mathbf{f}, \mathbf{y}|\mathbf{t}, \mathbf{X})\ d\mathbf{y} = p(\mathbf{f}|\mathbf{X}, \mathbf{t}),$$

which is exactly the posterior density we wished to sample from. In other words, the inclusion of the additional variables now enables us to generate samples from the original posterior density over $\mathbf{f}$.

9.2.1 Conditional densities for GP classification via Gibbs sampling

The joint posterior density over $\mathbf{f}$ and $\mathbf{y}$ is given by

$$p(\mathbf{f}, \mathbf{y}|\mathbf{t}, \mathbf{X}) = \frac{p(\mathbf{t}|\mathbf{y})p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\mathbf{X})}{p(\mathbf{t}|\mathbf{X})}.$$

We will use Gibbs sampling to sample from this density and therefore need conditional densities for each parameter (or set of parameters) conditioned on the others. Note that we will consider $\mathbf{f}$ as one parameter (we could obtain Gibbs conditionals for all of the individual components conditioned on all others but this isn't necessary) but will consider each element of $\mathbf{y}$ separately (out of necessity). The two densities that we need are therefore $p(\mathbf{f}|\mathbf{y}, \mathbf{X})$ and $p(\mathbf{y}|\mathbf{f}, \mathbf{X})$ which, because we assume that the elements of $\mathbf{y}$ are independent conditioned on $\mathbf{f}$ can be factored into N one-dimensional densities, $p(y_n|t_n, f_n)$.

Consider the identity

$$p(a|b) = \frac{p(a, b)}{p(b)},$$

which comes from the fact that $p(a, b) = p(a|b)p(b)$. Now, the denominator on the right hand side does not involve a, so we can say that $p(a|b) \propto p(a, b)$ – the conditional density is proportional to the joint density. Returning to the joint density of

interest

$$\begin{aligned} p(\mathbf{f}|\mathbf{y},\mathbf{t},\mathbf{X}) \propto p(\mathbf{f},\mathbf{y}|\mathbf{t},\mathbf{X}) &= \frac{p(\mathbf{t}|\mathbf{y})p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\mathbf{X})}{p(\mathbf{t}|\mathbf{X})} \\ &\propto p(\mathbf{t}|\mathbf{y})p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\mathbf{X}) \\ &\propto p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\mathbf{X}), \end{aligned}$$

where at each stage we have removed terms that do not depend on $\mathbf{f}$. This final expression looks familiar – the distribution of interest is proportional to a prior ($p(\mathbf{f}|\mathbf{X})$) multiplied by a likelihood ($p(\mathbf{y}|\mathbf{f})$) – Bayes' rule! In the case of $\mathbf{f}$, the prior is Gaussian (it is a GP prior) and the likelihood is also Gaussian

$$p(y_n|f_n) = \mathcal{N}(f_n, 1), \quad \text{and therefore} \quad p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{f}, \mathbf{I}),$$

so the conditional distribution we are trying to compute must be Gaussian too. To compute the mean and covariance of this Gaussian, we equate the coefficients of $\mathbf{f}$, exactly as we did for $\mathbf{w}$ when computing the posterior distribution over parameters for the model of the Olympic data in Section 3.8:

$$\begin{aligned} \mathcal{N}(\mathbf{f}|\boldsymbol{\mu}_\mathbf{f}, \boldsymbol{\Sigma}_\mathbf{f}) &\propto \mathcal{N}(\mathbf{y}|\mathbf{f}, \mathbf{I})\mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{C}) \\ \exp\left\{-\frac{1}{2}(\mathbf{f}-\boldsymbol{\mu}_\mathbf{f})^\mathsf{T}\boldsymbol{\Sigma}_\mathbf{f}^{-1}(\mathbf{f}-\boldsymbol{\mu}_\mathbf{f})\right\} &\propto \exp\left\{-\frac{1}{2}(\mathbf{y}-\mathbf{f})^\mathsf{T}\mathbf{I}(\mathbf{y}-\mathbf{f}) - \frac{1}{2}\mathbf{f}^\mathsf{T}\mathbf{C}^{-1}\mathbf{f}\right\} \\ \mathbf{f}^\mathsf{T}\boldsymbol{\Sigma}_\mathbf{f}^{-1}\mathbf{f} &= \mathbf{f}^\mathsf{T}\left[\mathbf{C}^{-1}+\mathbf{I}\right]\mathbf{f} \\ \boldsymbol{\Sigma}_\mathbf{f} &= \left[\mathbf{C}^{-1}+\mathbf{I}\right]^{-1} \\ \mathbf{f}^\mathsf{T}\boldsymbol{\Sigma}_\mathbf{f}^{-1}\boldsymbol{\mu}_\mathbf{f} &= \mathbf{f}^\mathsf{T}\mathbf{y} \\ \boldsymbol{\mu}_\mathbf{f} &= \boldsymbol{\Sigma}_\mathbf{f}\mathbf{y}. \end{aligned}$$

Similarly, $p(y_n|f_n, t_n)$ is proportional to the full posterior and we can start by ignoring all terms that do not depend on y_n and f_n. This leaves

$$p(y_n|f_n, t_n) \propto p(t_n|y_n)p(y_n|f_n).$$

If we take an example where $t_n = 1$, this becomes

$$p(y_n|f_n, t_n = 1) \propto P(t_n = 1|y_n)p(y_n|f_n) = \delta(y_n > 0)\mathcal{N}(y_n|f_n, 1),$$

which is a Gaussian with mean f_n truncated to be positive. Sampling from this density is straightforward: simply sample from the untruncated Gaussian and throw away all of the samples until a positive one arrives. For examples where $t_n = 0$, the density is

$$p(y_n|f_n, t_n = 0) \propto \delta(y_n < 0)\mathcal{N}(y_n|f_n, 1),$$

a Gaussian truncated to be negative.

We now have everything we need to generate samples from $p(\mathbf{f}, \mathbf{y}|\mathbf{X}, \mathbf{t})$ and therefore $p(\mathbf{f}|\mathbf{X}, \mathbf{t})$ (by just looking at the $\mathbf{f}$ samples). The procedure is as follows:

1. Initialise $\mathbf{y}$ with some sensible values (i.e. it would make sense for all y_n for which $t_n = 1$ to be positive)
2. Repeat for each desired sample:
   (a) Update $\mathbf{f}$ by sampling from $p(\mathbf{f}|\mathbf{y}, \mathbf{X})$.

(b) Update each y_n by sampling from $p(y_n|f_n, t_n)$.

(c) If making predictions, for a sample of $\mathbf{f}$, perform a GP regression to find $\mathbf{f}^*$.

(d) Sample $\mathbf{y}^*$ from $p(\mathbf{y}^*|\mathbf{f}^*) = \mathcal{N}(\mathbf{f}^*, \mathbf{I})$.

(e) Set $\mathbf{t}^{*,s}$ to one or zero depending on whether the corresponding elements of $\mathbf{y}^*$ are positive or negative.

3. Compute the predictive probabilities by averaging over $\mathbf{t}^{*,s}$.

We will now demonstrate this with an example (MATLAB script: `gibbs_gp_class.m`). Figure 9.4(a) shows some example 1-dimensional classification data. The class of the training examples is depicted by their height on the y axis and the shading of the plot symbol. Using the scheme described above, we can obtain posterior samples of the latent function at these training points. The mean plus and minus one standard deviation of $S = 10,000$ such samples can be seen in Figure 9.4(b), where we have used an RBF covariance function (see Equation 8.2) with $\alpha = 1, \gamma = 10$. For each new sample of $\mathbf{f}$, we can perform a (noise-free) GP regression to produce a sample of the latent function at a set of test points, $\mathbf{f}^*$ (in this case, we have constructed a fine grid of 100 test points in the range $0 < x < 1$). Figure 9.4(c) shows the posterior mean of the predictive function (solid black line), the mean plus and minus one standard deviation (dashed black lines) as well as a randomly chosen subset of 10 of the 10,000 samples (the plot also shows the posterior mean and standard deviation of $\mathbf{f}$, as in Figure 9.4(b)). Finally, for each sample of $\mathbf{f}^*$, we can sample a $\mathbf{y}^*$ and then assign 1 or 0 to the components of $\mathbf{t}^*$ depending on whether the respective components of $\mathbf{y}^*$ are positive or negative. After S samples, we can average these values of t to give a predictive probability. This probability is shown in Figure 9.4(d). We can see that the predictive probabilities seem reasonable – for areas of the input space (x) where we see a high density of training points with $t_n = 0$, the probability is low. Where we see training points with $t_n = 1$, the probability is higher.

Here, the initial model was not amenable to Gibbs sampling but it became so through the introduction of the auxiliary variables. The addition of extra variables has made posterior inference more straightforward – all of the conditionals are easy to sample from. We could have sampled from the posterior via Metropolis–Hastings and we leave it as an exercise to compare the two (see Exercise 9.2). In Chapter 10 we will see another model where Gibbs sampling is a natural choice for posterior inference.

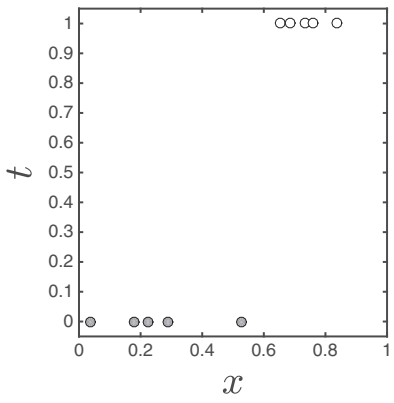
Finally, you may have noticed that, in the Gibbs sampler for GP classification, we updated the whole vector $\mathbf{f}$ by sampling from $p(\mathbf{f}|\mathbf{y}, \mathbf{X})$ rather than updating each variable in turn. We could have done the latter – $p(\mathbf{f}|\mathbf{y}, \mathbf{X})$ is a Gaussian and so all of the conditional densities are available – but there seems little point if we can do it all in one go, and in this case it is also much more efficient (see Section 9.4.2 and Exercise 9.1). In general, when building a Gibbs sampler, we can either update each parameter individually or in blocks (assuming conditionals are available in both cases). In many cases, the choice of blocks is obvious ($\mathbf{f}$ is a natural collection of parameters) and larger blocks tend to lead to greater efficiency.

9.2.2 Summary

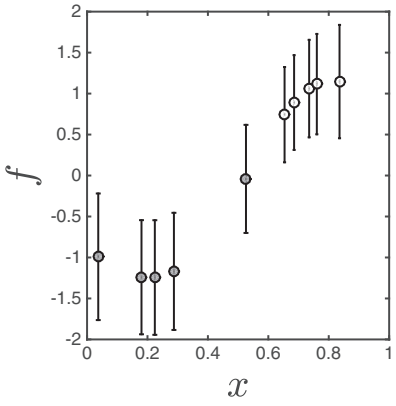
In this section we have introduced Gibbs sampling and demonstrated its use through sampling from a Gaussian as well as a Gaussian process classification model. Whilst

Metropolis–Hastings can, in theory, be used to sample from any posterior for which we can compute the product of prior and likelihood, Gibbs sampling is restricted to those models for which the conditional distribution of each parameter conditioned on all others is available.

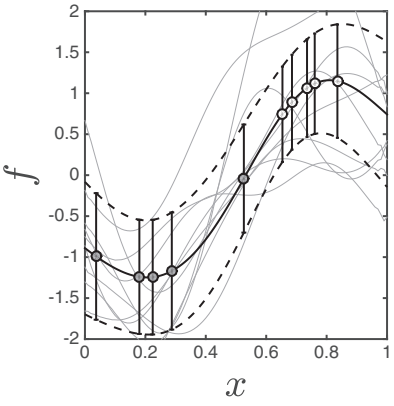
Up to now, we have presented Metropolis–Hastings and Gibbs sampling as recipes for generating samples, without any thought about why they work. This is the subject of the next section. For those of you who are just keen to get started and want some hints on how to use these techniques in the wild, you can skip straight to Section 9.4.



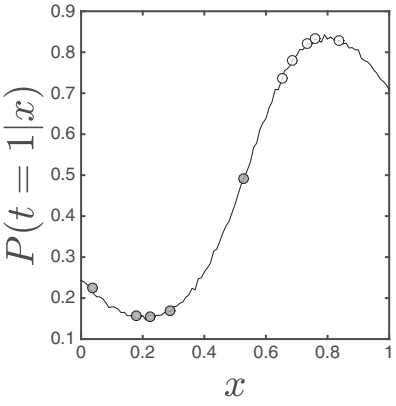
(a) Some example one-dimensional classification data. The y-axis (and plot symbol) denotes the class.



(b) Marginal posterior means and standard deviations over f_n from the Gibbs sampler.



(c) Predictive latent function values at a fine grid of test points. The solid black line gives the posterior mean prediction, the dashed lines ± one standard deviation and the grey lines show 10 example samples.



(d) Predictive probabilities for a fine grid of test points.

FIGURE 9.4 Gibbs sampling for GP classification.

9.3 WHY DOES MCMC WORK?

To understand why Metropolis–Hastings and Gibbs sampling work, we have to take a step back and learn a little about the theory of **Markov chains**. Figure 9.5 shows a system consisting of five boxes and a single token. The *state* of the system is defined as which box the token is in. At every second, the token is moved randomly according to the following rule: with probability equal to 0.5 it moves one square to the left, and with probability equal to 0.5 it moves one square to the right. The end states wrap around so that, if the token is in the far right state and it moves right, it ends up in the far left state. This system satisfies the *Markovian* property – the state of the system (location of the token) at time t depends only on its state at $t-1$. In other words, to decide where to move the token to at time $t+1$, we need only know where it is at time t, not where it was at any previous times.

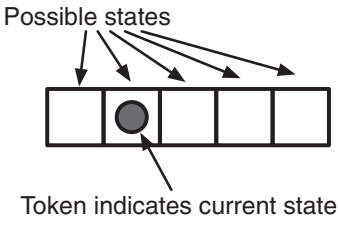


FIGURE 9.5 A simple state-space model consisting of five states (boxes) and one token to indicate the current state.

This state-space model is an example of a Markov chain. The probabilities of moving from one state to another are known as **transition probabilities**. Our simple example has a discrete state space (i.e. there is a set of distinct states) but Markov chains can also be defined over continuous state spaces. Consider the plots in Figure 9.3. Here we have a system moving through a continuous state space where the next position is dependent only on the current position (via the conditional distributions used in Gibbs sampling) – like Metropolis–Hastings, a set of samples generated via Gibbs sampling is an example of a Markov chain (hence the name Markov chain Monte Carlo).

Imagine you left the system in Figure 9.5 running for a long time and then came back: what is the probability that it will be in a particular state? This set of probabilities over states is known as the *stationary distribution* of the chain. For our example it will actually be a uniform distribution over the five states. Now consider Gibbs sampling or Metropolis–Hastings. If we start off a Gibbs or Metropolis–Hastings Markov chain to sample from some density $p(\theta)$, leave it alone for a while and then come back, the probability of the chain being in state θ is $P(\theta)$ – that's the distribution it is sampling over. Gibbs sampling and Metropolis–Hastings work by creating Markov chains with stationary distributions equal to the distribution of interest.

To firm up the relationship between the sampling methods we have seen and Markov chains, let's see how we can use our five-state model to sample from a distribution (MATLAB script: `sampleworld.m`). If we simply let the chain run and log

the states it visits, we are effectively generating samples from a uniform distribution over the five states (that's the stationary distribution of the chain). Can we get it to sample from an arbitrary distribution over the five states? The answer is yes – this is exactly what Metropolis–Hastings allows us to do. We will use it to sample from the following distribution over the five states:

$$1 : 0.1,\ 2 : 0.1,\ 3 : 0.4,\ 4 : 0.2,\ 5 : 0.1 \tag{9.6}$$

Metropolis–Hastings works by first proposing a new state and then deciding whether or not to accept it. For the proposal distribution, we will use the state transition probabilities described above, i.e. the probability of going from state 1 to state 2 is 0.5, as is the probability of going from state 1 to state 5. All other states have probability 0 from state 1. From state 2 we can either go to state 1 (probability 0.5) or state 3 (probability 0.5), etc. We will denote the transition probability of going from state θ_j to state θ_i with $q(\theta_i|\theta_j)$. Recall that the Metropolis–Hastings acceptance probability of a move from θ_j to θ_i is given by the ratio:

$$r = \frac{p(\theta_i)}{p(\theta_j)} \frac{q(\theta_j|\theta_i)}{q(\theta_i|\theta_j)}.$$

As the proposal is symmetric (the transition probability from state i to state j is equal to the transition probability from state j to state i), the second part of the ratio will cancel and so the acceptance probability reduced to the ratio of state probabilities. For example, the acceptance probability of going from state 3 to state 4 is $0.2/0.4 = 0.5$. If the sample is rejected, we stay in the same place. The probabilities obtained by averaging over 2000 posterior samples can be seen in Figure 9.6 (along with the true probabilities) and its clear that the method is indeed sampling from the correct distribution (MATLAB script: `state_chain.m`). This result isn't dependent on our choice of transition probabilities (i.e. our proposal).

So, how do the transition rules used by Gibbs sampling (sampling from conditional distributions) and Metropolis–Hastings (accepting based on the computed acceptance probability) create Markov chains with the distribution of interest as the stationary distribution? To describe this in detail is beyond the scope of this book. However, it can be shown (we won't) that, for a particular set of transition rules to converge to a particular stationary distribution, the rules and the desired stationary distribution must satisfy **detailed balance**. Detailed balance says that being in some state θ_i and moving into some other state θ_j has to be equally likely to being in state θ_j and moving into θ_i. If we use $p(\theta_j|\theta_i)$ to denote the probability of moving from state θ_i into state θ_j, detailed balance says

$$p(\theta_i)p(\theta_j|\theta_i) = p(\theta_j)p(\theta_i|\theta_j). \tag{9.7}$$

Note that it is important to make the distinction between the proposal distribution ($q(\theta_i|\theta_j)$) and the movement distribution ($p(\theta_i|\theta_j)$), particularly in the context of Metropolis–Hastings. Proposing a move from θ_j to θ_i does not necessarily mean it will happen. $p(\theta_i|\theta_j)$ is the probability that we propose to move from θ_j to θ_i *and* that it is accepted.

One way of interpreting detailed balance is that if you start your chain, go away and then come back and observe two consecutive states, you're just as likely

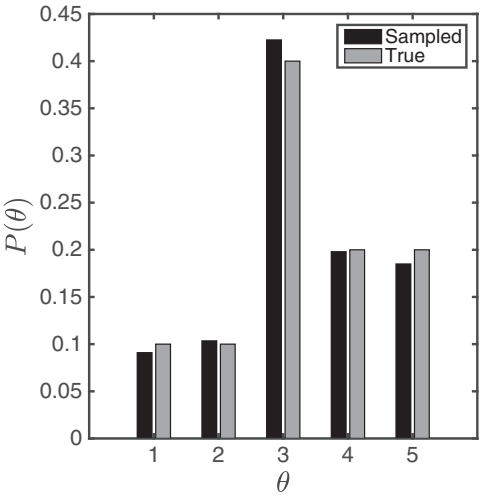


FIGURE 9.6 Samples obtained from the distribution described in Equation 9.6 using Metropolis–Hastings with the state-space model drawn in Figure 9.5. The black bars are the sample approximations and the grey bars the true values.

to observe θ_i, θ_j as you are θ_j, θ_i. Inspecting the samples we obtained when using Metropolis–Hastings with our five-state model, we can see that this is indeed the case. For example, the number of times we saw state 2 followed by state 3 was 100. The number of times we saw state 3 followed by state 2 was 105 – very similar values, despite the fact that we are in state 3 approximately four times as often as we are in state 2. Detailed balance also tells us that if it is possible to be in state θ_j (i.e. $p(\theta_j) > 0$) and possible to move to θ_i ($p(\theta_i|\theta_j) > 0$) then the opposite must also be possible.

For a particular distribution of interest ($p(\theta)$), any set of transition probabilities ($p(\theta_i|\theta_j)$) that satisfies detailed balance will give us a sampler that converges to $p(\theta)$. It is informative to see how the transition schemes defined by Metropolis–Hastings and Gibbs sampling both satisfy detailed balance. In both cases we use $p(\boldsymbol{\theta})$ to denote the distribution of interest (where $\boldsymbol{\theta}$ could be a vector or a scalar), and use $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_j$ to be two different states of $\boldsymbol{\theta}$. We will now demonstrate how the transition probabilities implied by the two methods satisfy Equation 9.7.

Starting with Metropolis–Hastings, we can always label the two states such that $p(\boldsymbol{\theta}_j) \geq p(\boldsymbol{\theta}_i)$ and to start with we will assume that the proposal density ($q(\boldsymbol{\theta}_j|\boldsymbol{\theta}_i)$) is symmetric. The acceptance ratio is computed as:

$$r = \frac{p(\boldsymbol{\theta}_j)}{p(\boldsymbol{\theta}_i)} \frac{q(\boldsymbol{\theta}_i|\boldsymbol{\theta}_j)}{q(\boldsymbol{\theta}_j|\boldsymbol{\theta}_i)},$$

where the second term is 1 (due to the symmetry of the proposal) and the first

term is greater than or equal to 1 (due to our labelling of the points). Therefore the sample is accepted. The left hand side of Equation 9.7 becomes:

$$p(\boldsymbol{\theta}_i)q(\boldsymbol{\theta}_j|\boldsymbol{\theta}_i)$$

For the right hand side, the acceptance ratio is:

$$r = \frac{p(\boldsymbol{\theta}_i)}{p(\boldsymbol{\theta}_j)} \frac{q(\boldsymbol{\theta}_j|\boldsymbol{\theta}_i)}{q(\boldsymbol{\theta}_i|\boldsymbol{\theta}_j}.$$

Again, the second term is 1. However r is now less than 1, so the sample is accepted with probability r. The right hand side of Equation 9.7 is therefore:

$$p(\boldsymbol{\theta}_j)q(\boldsymbol{\theta}_i|\boldsymbol{\theta}_j)\frac{p(\boldsymbol{\theta}_i)}{p(\boldsymbol{\theta}_j)}.$$

For detailed balance, these two sides must be equal. The full detailed balance equation is therefore

$$p(\boldsymbol{\theta}_i)q(\boldsymbol{\theta}_j|\boldsymbol{\theta}_i) = p(\boldsymbol{\theta}_j)q(\boldsymbol{\theta}_i|\boldsymbol{\theta}_j)\frac{p(\boldsymbol{\theta}_i)}{p(\boldsymbol{\theta}_j)},$$

which, given the symmetry of the proposal is clearly the case.

What about the more general case then when the proposal is not symmetric. In this case, we will label the states such that $p(\boldsymbol{\theta}_j)q(\boldsymbol{\theta}_i|\boldsymbol{\theta}_j) \geq p(\boldsymbol{\theta}_i)q(\boldsymbol{\theta}_j|\boldsymbol{\theta}_i)$, which ensures a ratio greater than 1 (and therefore guarantees acceptance) for a move from $\boldsymbol{\theta}_i$ to $\boldsymbol{\theta}_j$. The left hand side of Equation 9.7 is therefore:

$$p(\boldsymbol{\theta}_i)q(\boldsymbol{\theta}_j|\boldsymbol{\theta}_i),$$

as before. For the right hand side, the second term in the acceptance ratio no longer cancels, leaving:

$$p(\boldsymbol{\theta}_j)q(\boldsymbol{\theta}_i|\boldsymbol{\theta}_j)\frac{p(\boldsymbol{\theta}_i)}{p(\boldsymbol{\theta}_j)} \frac{q(\boldsymbol{\theta}_j|\boldsymbol{\theta}_i)}{q(\boldsymbol{\theta}_i|\boldsymbol{\theta}_j}.$$

Cancelling terms leaves:

$$p(\boldsymbol{\theta}_i)q(\boldsymbol{\theta}_j|\boldsymbol{\theta}_i),$$

which is identical to the left hand side, as required. We can conclude that Metropolis–Hastings satisfies detailed balance.

We should take a moment here to clarify the exact naming of this algorithm. Technically, when the proposal is symmetric, we are dealing with the Metropolis algorithm. It becomes the Metropolis–Hastings algorithm when the proposal is asymmetric – the Metropolis acceptance ratio is corrected so that detailed balance is still satisfied when the proposal is asymmetric.

Now let's see how Gibbs sampling satisfies detailed balance. Consider a set of parameters $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_N]^\mathsf{T}$. Consider a second set of parameters $\boldsymbol{\theta}' = [\theta'_1, \theta_2, \ldots, \theta_N]^\mathsf{T}$ that differs only in its value of θ_1. Detailed balance is then

$$p(\boldsymbol{\theta})p(\boldsymbol{\theta}'|\boldsymbol{\theta}) = p(\boldsymbol{\theta}')p(\boldsymbol{\theta}|\boldsymbol{\theta}').$$

Consider a slight variation on the Gibbs sampling method we described above – rather than looping through the individual variables, we select them at random. The left hand side is therefore

$$p(\boldsymbol{\theta})\frac{1}{N}p(\theta'_1|\theta_2, \ldots, \theta_N)$$

where the $\frac{1}{N}$ term comes from the random choice of parameter 1. Doing the same to the right hand side leaves us with the following expression:

$$p(\boldsymbol{\theta})\frac{1}{N}p(\theta_1'|\theta_2,\ldots,\theta_N) = p(\boldsymbol{\theta}')\frac{1}{N}p(\theta_1|\theta_2,\ldots,\theta_N).$$

To show that this is true (and hence that detailed balance holds for Gibbs sampling), we need to expand the conditional distributions. In particular, from the relationship between conditional and joint distributions:

$$p(\theta_1'|\theta_2,\ldots,\theta_N)p(\theta_2,\ldots,\theta_N) = p(\boldsymbol{\theta}')$$

and therefore

$$p(\theta_1'|\theta_2,\ldots,\theta_N) = \frac{p(\boldsymbol{\theta}')}{p(\theta_2,\ldots,\theta_N)}.$$

Doing the same for $p(\theta_1|\theta_2,\ldots,\theta_N)$ and substituting into our detailed balance equation, we have

$$p(\boldsymbol{\theta})\frac{1}{N}\frac{p(\boldsymbol{\theta}')}{p(\theta_2,\ldots,\theta_N)} = p(\boldsymbol{\theta}')\frac{1}{N}\frac{p(\boldsymbol{\theta})}{p(\theta_2,\ldots,\theta_N)}.$$

Detailed balance is therefore satisfied for Gibbs sampling.

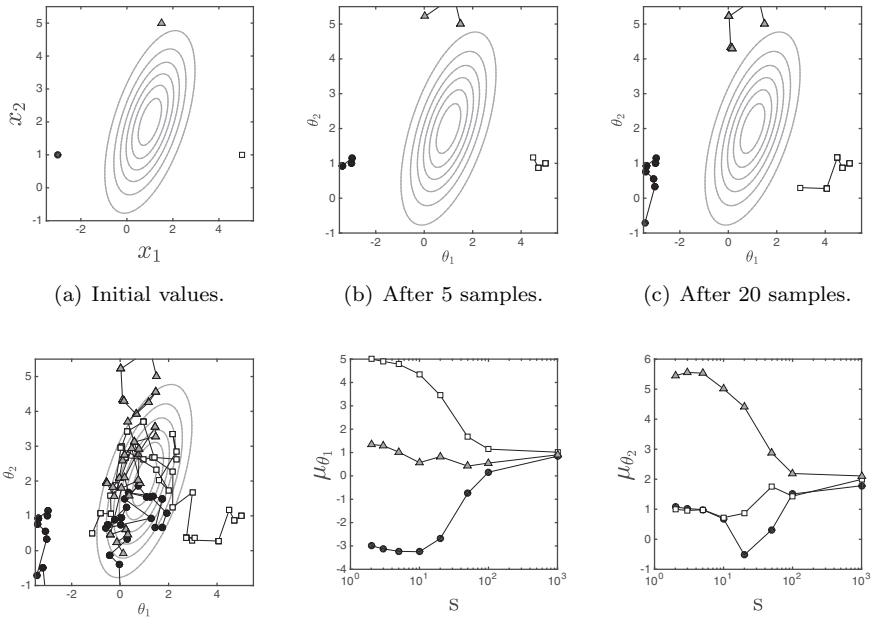
We have seen how Metropolis–Hastings and Gibbs sampling satisfy detailed balance and therefore that chains using the transition probabilities defined by Gibbs sampling and Metropolis–Hastings have the desired distribution as their stationary distribution. What we haven't done is convince you why detailed balance is the right thing to satisfy! That would be beyond the scope of this book, and readers who want to learn more are encouraged to explore the references provided at the end of this chapter.

9.4 SOME SAMPLING PROBLEMS AND SOLUTIONS

In theory, sampling (via either Gibbs sampling or Metropolis–Hastings) appears straightforward. In practice things are not quite so easy. In this section we will look at two of the key practical issues facing sampling methods when used in the wild: assessing convergence and reducing autocorrelation.

9.4.1 Burn-in and convergence

In our description of Gibbs sampling and Metropolis–Hastings, we paid very little attention to the initialisation of the algorithm – i.e., where to start. In theory, it does not matter where we start, but in practice a poor choice of starting position can result in the chain taking a long time to converge to the desired distribution. Some initialisation methods are likely to be more sensible than others. For example, when sampling from a posterior distribution, initialising with a sample from the prior might be more sensible than plucking a value out of thin air. Even with what looks like a sensible choice, our starting point may not be in an area of significant posterior density. Initial samples will therefore be quite unrepresentative of the posterior. We therefore need a method of determining whether or not our chain has converged and is sampling from the intended distribution. As we aren't able to evaluate the true distribution (if we could, we probably wouldn't have decided to sample), this is not straightforward. But, if we change the question slightly, we we can devise a proxy



(a) Initial values. (b) After 5 samples. (c) After 20 samples.

(d) After 50 samples. (e) Estimate of mean of θ_1. (f) Estimate of mean of θ_2.

FIGURE 9.7 Three chains sampling from a two-dimensional Gaussian density. The final plots show the estimates of the Gaussian mean obtained from increasing numbers of samples.

for convergence. In particular, rather than trying to determine if a single chain has converged to the *true* distribution, we try and assess if multiple chains (with different initialisations) have all converged to the *same* distribution. This is illustrated in Figure 9.7 (MATLAB script: `convergence.m`), where we show three Metropolis–Hastings chains (with a symmetric Gaussian proposal density with $\sigma^2 = 0.4$ for both dimensions) sampling from a two-dimensional Gaussian density. To visually assess whether the chains have converged to the same distributions, the final two plots in Figure 9.7 show the estimates of the mean of the Gaussian using increasing numbers of samples (note the logarithmic scale on the x-axis).

In practice, we can use a slightly formal idea of convergence, also taking into account the variance between and within the chains. The procedure that we will describe is introduced in *Bayesian Data Analysis* (see reference at the end of the chapter) and the reader is referred to that book for a more detailed discussion.

Assume that, for each parameter of interest, we have $s = 1 \dots S$ samples from each of $c = 1 \dots C$ chains, where the sth sample in the cth chain is denoted θ_{sc}. In our example, we have $S = 1000$ samples from $C = 3$ chains. We start by computing

the mean (μ_c) and variance (σ_c^2) of the samples within each chain:

$$\mu_c = \frac{1}{S}\sum_{s=1}^{S}\theta_{sc}, \quad \sigma_c^2 = \frac{1}{S-1}\sum_{s=1}^{S}(\theta_{sc}-\mu_c)^2.$$

We then average the mean values to obtain a global mean, $\mu = \frac{1}{C}\sum_{c=1}^{C}\mu_c$, and use these three quantities to compute within and between chain variances, W and B:

$$W = \frac{1}{C}\sum_{c=1}^{C}\sigma_c^2, \quad B = \frac{S}{C-1}\sum_{c=1}^{C}(\mu_c-\mu)^2.$$

Finally, we compute the measure of interest, $\hat{R}$, as

$$\hat{R} = \sqrt{\frac{V}{W}}, \tag{9.8}$$

where V is an estimate of the marginal posterior variance of θ, computed as a weighted sum of B and W:

$$V = \frac{S-1}{S}W + \frac{1}{S}B.$$

$\hat{R}$, converges to 1 as the number of samples approaches infinity. We therefore sample until this quantity becomes sufficiently close to 1 for all parameters. How close *sufficiently close* is depends on the application, but waiting for values below 1.01 is quite common in practice. A value of $\hat{R}$ that seems to stop decreasing at a higher value is possibly indicative of a problem with your sampler. Figure 9.8 shows the evolution of $\hat{R}$ for our example. The solid line corresponds to x_1 and the dashed to x_2. After 2000 samples, the value of $\hat{R}$ for both parameters is well below 1.01.

This gives us an objective procedure for determining whether or not our sampler has converged. Start multiple chains from diverse starting positions (the more chains the better). Monitor $\hat{R}$ for each parameter, and, when it has fallen to a value below some predetermined threshold (say 1.01) for all parameters, consider the chains to have converged. At this point we can start our sampling proper. The convergence of the chains just tells us that they are all sampling from the same distribution (which we assume must be the true one) and so we can confidently start collecting samples to use. Typically, all of the initial samples are discarded, and this initial sampling is known as the burn-in phase (see Exercise 9.3).

9.4.2 Autocorrelation

Once we have decided that our chains have converged, there is still another potential hurdle to overcome – **autocorrelation**. We will often be using the samples to approximate an expectation. For example,

$$\int f(\boldsymbol{\theta})p(\boldsymbol{\theta})\, d\boldsymbol{\theta} \approx \frac{1}{S}\sum_{s=1}^{S} f(\boldsymbol{\theta}^s).$$

For this approximation to work, the S samples $\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^S$ need to be independent samples from $p(\boldsymbol{\theta})$. Unfortunately, the nature of most sampling algorithms means

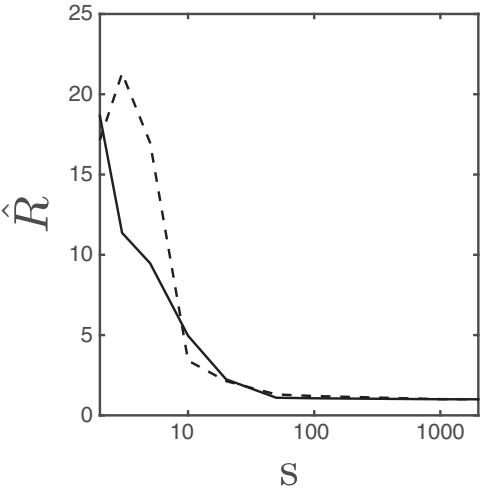


FIGURE 9.8 Evolution of $\hat{R}$ computed from the chains shown in Figure 9.7. The solid line corresponds to x_1 and the dashed to x_2.

that consecutive samples that are produced are very dependent on one another. This is particularly obvious for Metropolis–Hastings, where new samples are proposed by moving a small distance away from the current sample. If the proposed value is accepted, the new sample will be similar to (dependent on) the old one. If the proposed value is rejected, the new sample will be identical to (very dependent upon!) the old one.

There are two ways in which we can overcome this problem. Firstly, design our sampler to produce samples that are as independent as possible and, secondly, remove any remaining dependence by throwing lots of samples away (known as **thinning**). For example, we could keep only every tenth sample, throwing the rest away.

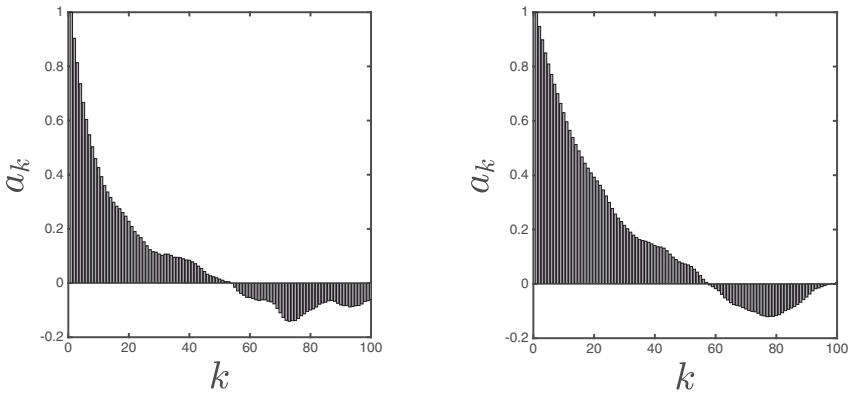
The choice of sampling algorithm will influence the extent to which we can use these two approaches, and, in practice, both will be required. For example, when using Metropolis–Hastings, the choice of proposal distribution will have a significant role in determining the dependence between successive samples, but it's unlikely that you will find a proposal distribution that completely removes the need for thinning. When using Gibbs sampling, it is sometimes possible to update parameters in blocks, or integrate one set of parameters out when sampling another. Both will reduce the dependence between consecutive samples, but only up to a point.

The dependence between samples can be measured by computing the autocorrelation of the samples. Autocorrelation measures the correlation between samples in the same chain separated by some lag. For example, the lag 1 autocorrelation is the correlation of consecutive samples, θ^s and θ^{s+1}. The lag 2 autocorrelation is the correlation between samples two steps apart (e.g. samples θ^1 and θ^3, and θ^2 and θ^4, etc.).

Autocorrelation is computed separately for each chain, so for notational simplicity we can drop the chain subscript and consider $s = 1 \ldots S$ samples, θ_s for each chain. The autocorrelation (at a lag of k; a_k) is then computed as

$$a_k = \frac{1}{(S-k)\sigma^2} \sum_{s=1}^{S-k} (\theta_s - \mu)(\theta_{s+k} - \mu),$$

where θ_{s+k} is the sample k steps after the sth one, μ is the mean of the S samples, and σ^2 the variance. The summation is over all pairs of samples separated by k steps (the first pair will be θ_1 and θ_{1+k} and the last pair θ_{S-k} and θ_S). In the extreme case of $k = 0$, this expression simplifies to $\sigma^2/\sigma^2 = 1$, which is the maximum value of autocorrelation possible (see Exercise 9.4).

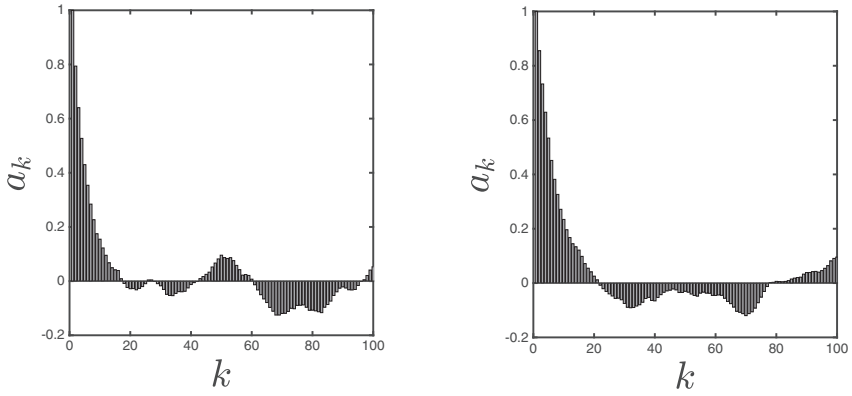


(a) θ_1 (b) θ_2

FIGURE 9.9 Autocorrelation for samples of x_1 and x_2 for one of the three chains shown in Figure 9.7.

Often autocorrelation is plotted as a function of k to provide a useful visual diagnosis of the degree of autocorrelation in the samples. The plots for θ_1 and θ_2 from our Metropolis–Hastings example above are given in Figure 9.9.

These plots show very high autocorrelation – our samples are very dependent on one another, even at quite large lags. An autocorrelation plot for independent samples would have a value of 1 for lag 0 and then values of zero (or very close to zero) elsewhere. If faced with plots like these, one ought to thin the samples considerably to try and leave a set of independent samples. The plots suggest that the autocorrelation reaches zero at a lag of approximately 40. So, we should thin by keeping the first one, discarding the next 40, keeping the 42nd one, discarding the next 40 etc. This comes at a considerable computational cost (we are discarding almost all samples we draw), but thinning at this or higher levels is not uncommon in practice. Fortunately, in this case, we can improve things through other means. These samples were drawn with an isotropic Gaussian proposal density that had a variance of 0.4 for both dimensions. Increasing this value will lead to bigger movements and



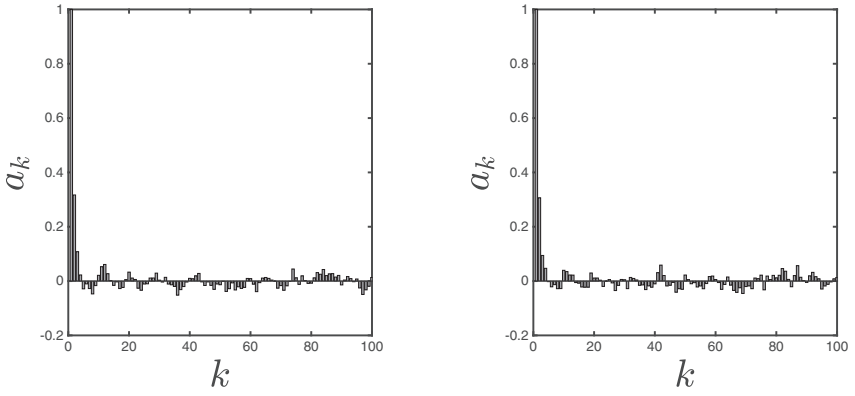
(a) x_1 (b) x_2

FIGURE 9.10 Autocorrelation for samples of x_1 and x_2 for one of the three chains shown in Figure 9.7 with a proposal density with variance increased to 3.

therefore autocorrelation that decays rapidly. The resulting autocorrelation is shown in Figure 9.10 and we can indeed see that the autocorrelation decays more rapidly but still pretty slowly. Thinning these samples would still be required.

In general, when sampling (particularly with Metropolis–Hastings and similar methods), it is useful to compute the **acceptance ratio**, which is defined as the proportion of proposed samples that are accepted. In this example, when the proposal variance was 0.4, the value was around 0.6. When the variance was increased to 3, the value was closer to 0.3. It drops because, as the steps get bigger, there is an increased chance that a bad new value will be proposed. An acceptance rate of 0.3 means that most of the time (about 70%) the proposed sample is rejected and the current sample is used again. This will obviously affect the autocorrelation – if too many samples are rejected, the copying of samples for many steps will give very high autocorrelation. There is therefore a trade-off between high autocorrelation due to small steps (that are regularly accepted) and high correlation due to steps that are too big and often rejected. In practice, varying proposal densities and monitoring autocorrelation is a crucial step in designing an efficient sampler. Optimal acceptance ratios have been theoretically derived for various distributions. See the references at the end of the chapter for more information.

For comparison, we can plot the autocorrelation for a Gibbs sampler for the same density. This is shown in Figure 9.11. The autocorrelation drops much more rapidly, and one could just thin every other sample to obtain samples that were practically independent. Why is this the case? Let's just consider θ_1. In Metropolis–Hastings, a new value of θ_1 is proposed based on the old value. There is therefore a direct dependence between the consecutive values. In Gibbs sampling, the old value of θ_1 is used when sampling a new value of θ_2, which is then used when sampling the new value of θ_1. In this case, the dependence is indirect (mediated by θ_2), and the



(a) θ_1 (b) θ_2

FIGURE 9.11 Autocorrelation for samples of θ_1 and θ_2 for a Gibbs sampler.

dependence therefore depends on the strength of the relationship between θ_1 and θ_2. It seems unsurprising that this weakens the effective dependence between successive values of θ_1 as measured by the autocorrelation.

This will generally be the case – if we have the choice between Metropolis–Hastings and Gibbs for the same set of parameters, we would typically go for Gibbs. However, the situation gets muddier when you consider updating groups of variables at a time. For example, Metropolis–Hastings lets us update as many parameters as we like at once, whereas with Gibbs we are limited to what we can compute the conditional distributions for (we might be able to group things, we might not). When conditional distributions for blocks are unavailable, there will be examples where it is more efficient to do Metropolis–Hastings on the blocks than Gibbs sampling on the individual parameters. We saw an example of a Gibbs block update earlier in this chapter. When we built the GP classification model, we computed a conditional density for the vector $\mathbf{f}$, rather than the individual elements f_n. We leave it as an exercise (Exercise 9.1) to derive the individual updates and assess the (in)efficiency of sampling them all individually.

It is worth remembering that, in many problems, the choice between Gibbs and Metropolis–Hastings (or others) is not one we have. We can use Metropolis–Hastings on any model, as long as we can compute the prior and likelihood. Gibbs sampling can only be used on a small subset of models for which the conditional distributions are available.

9.4.3 Summary

In this section we have described two important measures for diagnosing how well our sampler is operating. $\hat{R}$ can be used to determine if multiple chains are sampling from the same distribution, whilst the autocorrelation can tell us how (in)dependent

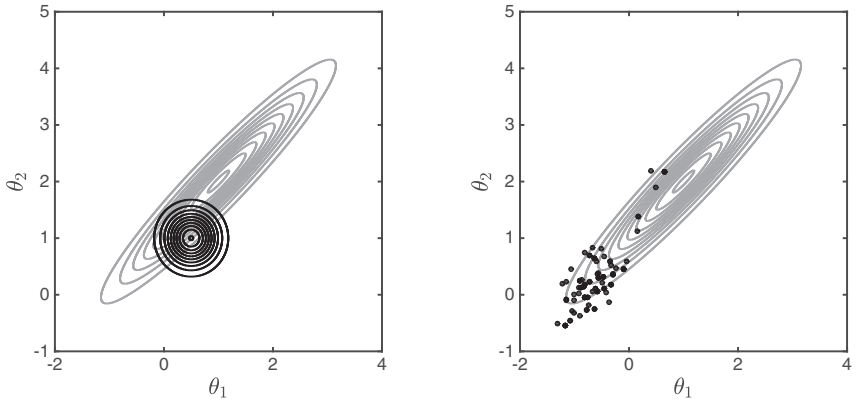
our samples are. Careful use of these measures can help ensure that samplers are working correctly (they will often not be!) and that the samples being produced can legitimately be used to compute expectations, etc.

Other tools are available for assessing convergence and reducing sample dependence. For example, although it is subjective, visual inspection is a useful tool. Re-parameterisation of the model can also be very helpful to, for example, remove parameters that cannot be identified from the data available. It is also possible to marginalise parameters (particularly in Gibbs sampling), and we will see an example of this in Chapter 10.

In the final part of this chapter we will provide a brief introduction to various advanced sampling techniques that are currently popular in machine learning.

9.5 ADVANCED SAMPLING TECHNIQUES

9.5.1 Adaptive proposals and Hamiltonian Monte Carlo



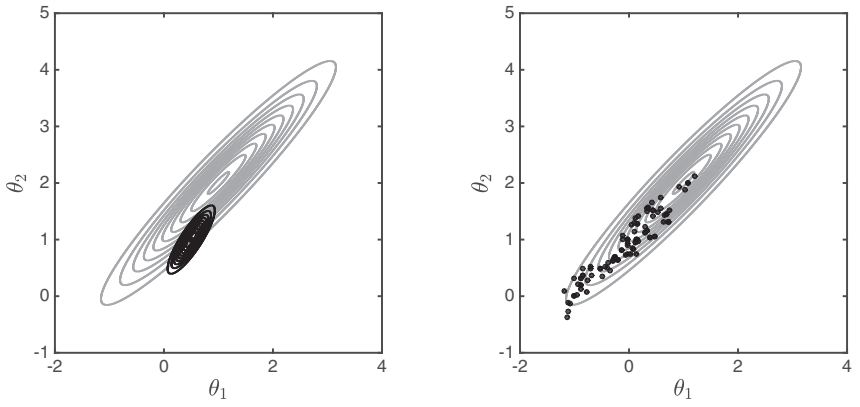
(a) An isotropic proposal (dark contours) being used to sample from a distribution with high covariance.

(b) The first 100 samples using the proposal shown in (a).

FIGURE 9.12 An example of Metropolis–Hastings when the proposal distribution is not well matched to the distribution being sampled from.

The efficiency of Metropolis–Hastings is partly dependent on the relationship between the distribution being sampled from and the proposal distribution. If the proposal distribution is likely to often propose very poor movements, many samples will be rejected. Figure 9.12(a) shows an example where the proposal is poorly matched to the distribution being sampled from (MATLAB script: `badmh.m`). The distribution being sampled from (light contours) exhibits high covariance, whereas the proposal distribution (darker contours) is isotropic. Given the current value of $\boldsymbol{\theta}$ (the centre of the proposal distribution), we can see that the proposal distribution is highly likely to propose a poor new value of $\boldsymbol{\theta}$. Regardless of the position of the

current sample, many samples will be rejected. One way to overcome this would be to reduce the variance of the proposal distribution, resulting in proposals being (on average) closer to the current value. This makes it less likely that proposals will be rejected (shorter moves are less likely to result in large drops in the posterior value) but it will make the autocorrelation between samples higher (resulting in the need for a greater degree of thinning) and mean that the sampler takes much longer to explore the full distribution.



(a) A Gaussian proposal with covariance equal to 0.2 multiplied by the empirical covariance from the samples shown in Figure 9.12(b).

(b) The first 100 samples using the adapted proposal.

FIGURE 9.13 Adapting the proposal based on previous samples.

A second solution is to make the proposal more representative of the true distribution. For example, the distribution in Figure 9.12(a) would benefit from a proposal distribution that favoured moves roughly along the line $\theta_1 = \theta_2$. Without knowledge of the distribution being sampled from, we can add useful covariance structure to the proposal by basing it on samples already drawn. In our example, we can use the covariance matrix of the 100 samples shown in Figure 9.12(b) (multiplied by a small constant) to replace the covariance of the original proposal distribution. This new proposal distribution should produce a more efficient sampler than the original one.

Figure 9.12(b) shows this in action. The proposal density shown has as its covariance the covariance of the samples shown in Figure 9.12(b) multiplied by a small constant (0.2). The first 100 samples using this new proposal are shown in Figure 9.13(b). We can see that the new sampler is able to explore the space more efficiently.

Of course, the more samples on which we base the proposal the better. There is nothing stopping us from continually adapting the proposal as we learn more about the covariance structure of the distribution being sampled from. In this example, the constant we multiplied the empirical covariance by (0.2) was chosen fairly arbitrarily. See the references at the end of this chapter for pointers to where choices of this value are discussed.

An alternative to adapting the proposal covariance in this way is through the use of **Hamiltonian Monte Carlo** (HMC, also known as hybrid Monte Carlo). We will not give a full description of HMC here – see the references at the end of the chapter for suggestions for further reading.

HMC uses information about the gradient of the distribution to inform our proposal. In particular, for each variable being sampled, HMC introduces an additional momentum parameter and samples from the joint distribution of the original and momentum parameters. Calling the momentum parameters $\mathbf{v}$, we sample from

$$p(\boldsymbol{\theta}, \mathbf{v}) = p(\boldsymbol{\theta})p(\mathbf{v}),$$

where $p(\mathbf{v})$ is user defined and will affect the efficiency of the sampler. The proposal step starts by drawing a new value of $\mathbf{v}$ from $p(\mathbf{v})$. This is followed by a user-defined number of steps, in each of which the momentum is updated according to the derivative of the log of the distribution with respect to $\boldsymbol{\theta}$, and $\boldsymbol{\theta}$ is moved based on the momentum. After the steps are complete, the acceptance probability is computed as in standard Metropolis–Hastings but with the ratio of the proposals replaced with the ratio of $p(\mathbf{v})$ after and before the iterative steps. Although we are technically accepting both $\mathbf{x}$ and $\mathbf{v}$, we aren't interested in the values of $\mathbf{v}$, so don't need to store them from one sample to the next.

Figure 9.14 illustrates this proposal procedure. We start by sampling a momentum. The current value of $\boldsymbol{\theta}$ gets moved in this momentum direction. The momentum

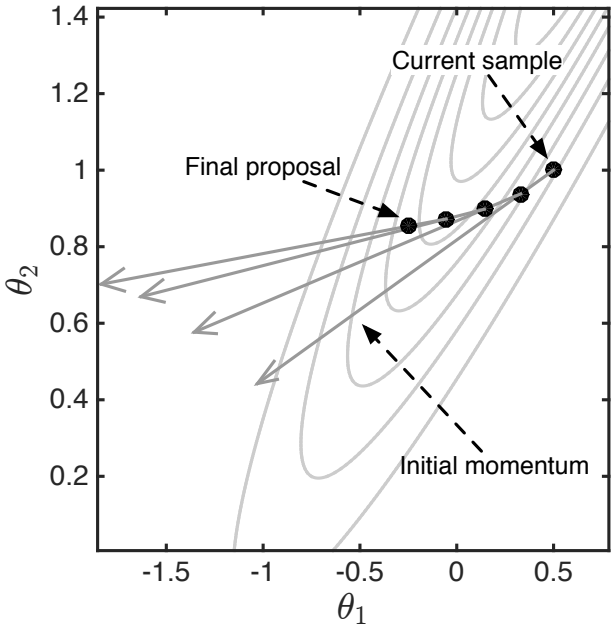


FIGURE 9.14 A single proposal step in the HMC algorithm.

is then updated based on the gradient of $\log p(\boldsymbol{\theta})$, which makes it point slightly more uphill, and $\boldsymbol{\theta}$ is updated again. After several steps, the procedure is stopped and the value of $\boldsymbol{\theta}$ is used as the proposed value. This proposal is still random (it's based on the initial value of $\mathbf{v}$), but it is biased to move up the gradient of $p(\boldsymbol{\theta})$. A proof of how this procedure obeys detailed balance is beyond our scope but, needless to say, it does, and it can make sampling much more efficient. Figure 9.15 shows the first 100 samples obtained from our distribution using HMC (MATLAB script: `badmh.m`). Comparing with the standard Metropolis–Hastings approach (Figure 9.12(b)), we can see that it explores the distribution much more efficiently.

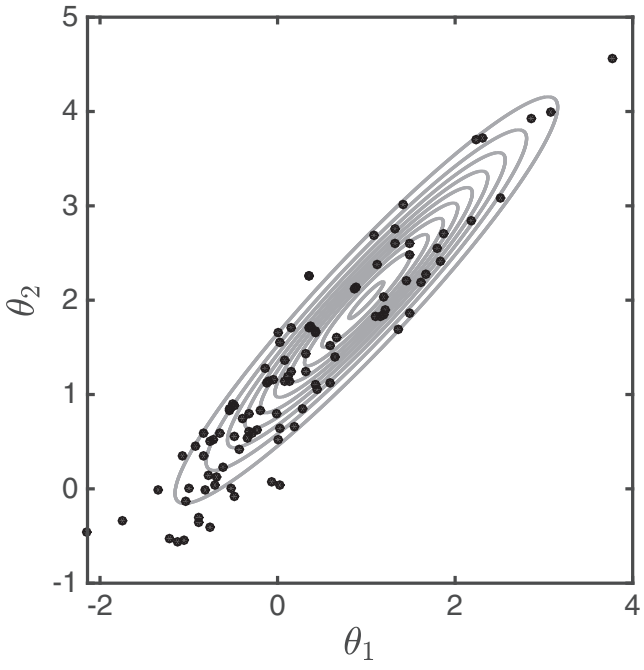


FIGURE 9.15 One hundred samples obtained using HMC.

Because it uses information about the gradient of the distribution, HMC will often be more efficient than standard Metropolis–Hastings. However, we can only use HMC when we do have access to the derivative information, which, for some complex models, may not be the case.

9.5.2 Approximate Bayesian computation

When using Metropolis–Hastings (and related techniques, e.g. HMC) to perform Bayesian inference, we have to be able to compute the likelihood of the observed

data for the current and proposed set of parameters. In all of our applications, we have been able to do this easily. However, there exist applications where computing the likelihood is either impossible or very computationally expensive. **Approximate Bayesian computation** (ABC) has been developed for these situations and is a technique that allows us to sample from the posterior density without having to compute the likelihood at all.

Recall the fairground coin game that we used as our first example of Bayesian inference in Chapter 3. We observed a certain number of heads (n) from a certain number of tosses of a coin (N). We showed how we were able to analytically compute the posterior density over the probability that any individual toss was a head (r) when the likelihod was a Bernoulli distribution and the prior a beta density.

Say we observe $n = 3$ heads in $N = 10$ tosses and we assume that we know nothing about the coin so choose a uniform prior ($\alpha = \beta = 1$). At the fairground you don't have access to the means of computing the posterior exactly, but you do have a way of generating possible coins from this uniform prior which, coincidentally, is exactly the scenario in which you could use ABC!

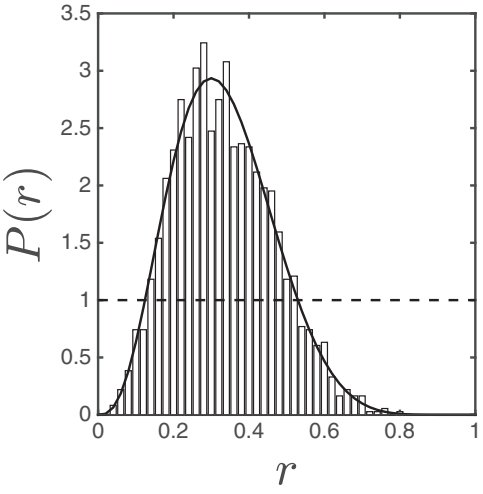
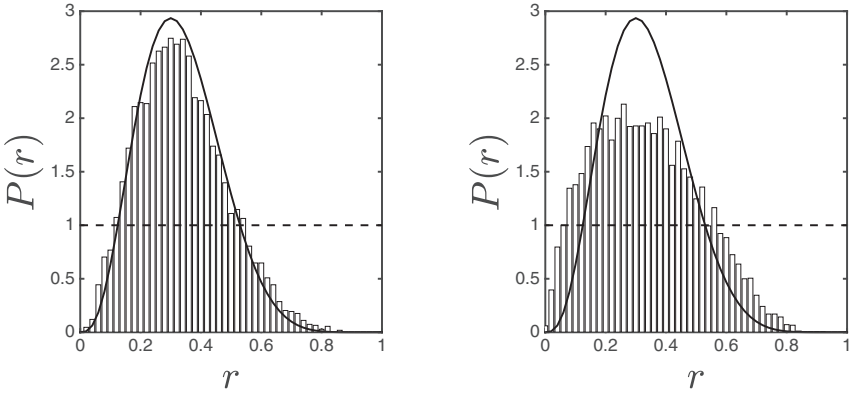


FIGURE 9.16 Example of ABC for the coin game example introduced in Chapter 3.

To use ABC, we generate lots of samples from the prior. For each sample, we generate a dataset according to the generative process defined by the likelihood. Note that, in this and many other scenarios, this doesn't require calculating the likelihood – we simply toss the coin $N = 10$ times, and count n, the number of heads. If the number of heads is equal to the number of heads we observed in reality (three), we accept the prior sample. If not, we throw it away. The magic of ABC is that the accepted samples are samples from the posterior density! We can intuitively see why this is the case. The posterior should give non-zero weight to all values of r that could produce three heads from ten tosses. This is all values except $r = 0$ and

$r = 1$, but samples of r corresponding to areas of high posterior density (in this case, around 0.3) will be more likely to produce three heads from ten tosses and therefore be accepted far more often than those at much higher and lower values.

Figure 9.16 shows the empirical posterior density obtained by sampling (white bars) along with the true posterior (solid black line) and the prior (dashed black line) (MATLAB script: `abc.m`). The empirical distribution was obtained by generating 20,000 samples from the prior, of which 1819 (9%) were accepted. This is pretty inefficient – 90% of the samples we obtain are rejected, and this figure gets worse the more complex we make the problem. For example, if we increase N to 20 and observe $n = 6$ heads, we only accept 5% of the samples and if $N = 50$ and $n = 15$, we only accept 2% of samples. For many real applications the situation is worse, and efficiency is very low.



(a) Maximum error of one.

(b) Maximum error of two.

FIGURE 9.17 ABC sampling for the coin game when we accept samples that generate a number of heads maximally one and two different from the true value (three).

Our set of samples is an exact sample from the posterior – there is no approximation here. So why the 'approximation' in the name? Well, for many likelihoods, even with the true parameters we would never generate a dataset that was identical to the true one. Consider doing Bayesian inference over the posterior density of the mean of a Gaussian for which we have observed one data point. Even if we knew the true mean, we would never sample *exactly* the value we had previously observed.

To overcome this, ABC allows prior samples to be accepted if the dataset they produce is sufficiently similar to the true data. This makes things more efficient (we accept more parameters) but we no longer know that we are sampling from the true posterior density and there is not really any way of knowing how close to the true posterior the distribution we are sampling from is.

In our coin example, we can relax our acceptance criteria by accepting a sample if it generates a number of heads that differs by the true value by a maximum of one or a maximum of two. Plots of the true and empirical posteriors for both cases can be

seen in Figure 9.17. As expected, the efficiency increases (27% and 40% of samples are accepted, respectively) but the empirical posterior becomes quite different from the true one, spreading across more values of r.

When comparing the true and generated datasets, one can compare the data directly or compare summary statistics of the data. This is what we have done above – we haven't compared the exact sequence of heads and tails obtained in the ten tosses, but just the number of heads. In this example, the data (the sequence of heads and tails) and summary statistic (number of heads) will both give the exact posterior (if we don't relax the acceptance criteria). We can get a feeling for why this is the case by looking at what happens when you normally compute the posterior analytically. The posterior is given by

$$p(r|n, N, \alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} r^{\alpha+n-1}(1-r)^{\beta+N-n+1}.$$

which only uses the number of heads (and not their order). So, somehow, all of the information that we require is captured by the number of heads. If we only accepted identical sequences, all prior samples of r would be less likely to be accepted, but this drop in probability would not depend on the particular value of r. So, we would end up with the same distribution, just fewer samples.

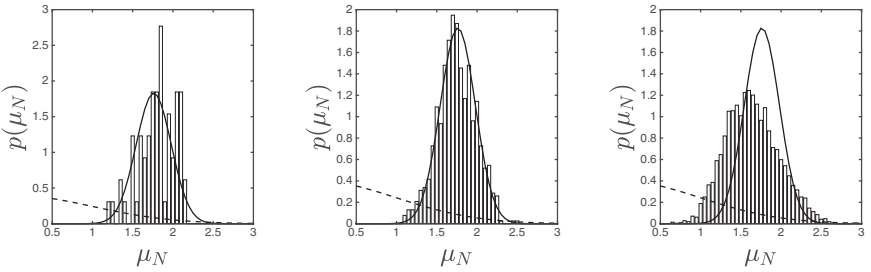
Similarly, if we are interested in inferring the mean of a Gaussian density (with known variance, σ^2) based on observing N values from the Gaussian $(x_1, \ldots, x_N)$ and a Gaussian prior with mean μ_0 and variance σ_0^2, the posterior is Gaussian with mean and variance given by

$$\sigma_N^2 = \left(\frac{1}{\sigma_0^2} + \frac{N}{\sigma^2}\right)^{-1}, \quad \mu_N = \sigma_N^2 \left(\frac{\mu_0}{\sigma_0^2} + \frac{1}{\sigma^2}\sum_n x_n\right),$$

in which the data only appear as a summation. That is all we need to know to compute the posterior. So, if we were willing to wait a long time, we could accept samples that generated datasets with the same sum as the real data and get samples from the true posterior. Statistics such as these that encapsulate all of the information in the data that we need to do inference are examples of *sufficient statistics*.

Finally, we will demonstrate how ABC allows us to get approximate samples from the posterior density of the mean of a Gaussian (with known variance). We will use a scheme where a prior sample is accepted if the absolute difference of the mean of the generated data is less than some tunable parameter ϵ. Figure 9.18 shows empirical and true posteriors for increasing values of ϵ. In all cases, the empirical sample was generated by taking 50,000 samples from the prior (mean zero, unit variance; dashed line on the plots). As ϵ increases, we can clearly see the trade-off between efficiency and accuracy. For $\epsilon = 0.01$, we have the samples most representative of the posterior (they must be, as the acceptance is at its most stringent), but very few of them (only 0.13% of samples) were accepted. At the other extreme ($\epsilon = 0.5$), we have many more accepted samples (although still only 8.5%) but the empirical distribution is quite different from the true posterior. In this example, the middle value ($\epsilon = 0.1$) seems to represent a reasonable trade-off between number of samples (1.5%) and proximity to the true posterior.

Although ABC is approximate, it is increasingly popular in areas where likelihood computation is expensive (or impossible). Improving ABC techniques is an open area of research (see Exercise 9.5).



(a) $\epsilon = 0.01$ (b) $\epsilon = 0.1$ (c) $\epsilon = 0.5$

FIGURE 9.18 Using ABC to sample from the posterior density of the mean of a Gaussian with known variance. The three plots correspond to three values of ϵ, the maximum absolute difference between the means of the data and generated datasets for a sample to be accepted. The solid line is the true posterior and the dashed line the prior.

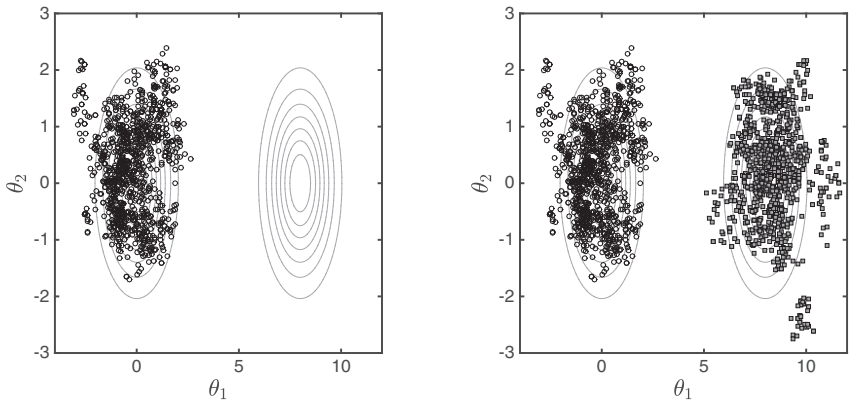
9.5.3 Population MCMC and temperature schedules

In Section 9.5.1 we saw that using Metropolis–Hastings in situations where parameters have high covariance can be challenging unless we can incorporate this information into our proposal (which we did via adapting the proposal covariance and using Hamiltonian Monte Carlo). Another feature of a distribution that can cause problems for samplers is multi-modality. Figure 9.19(a) shows such a distribution and 1000 samples from a Metropolis–Hastings sampler with an isotropic proposal density ($\sigma^2 = 0.05$). We can see that the sampler has explored one of the modes but no samples have been drawn from the other mode. In theory, the sampler will eventually reach all of the modes, but in practice, movement from one mode to another requires a large number of downhill moves which are collectively very unlikely. If we start multiple chains, as in Figure 9.19(b), each chain seems to get stuck in whichever mode it enters first. Fortunately, it is quite easy to detect this kind of problem – the $\hat{R}$ statistic introduced earlier will not converge as the samples from the individual chains will look so different. Unfortunately, it is not so easy to fix!

Population MCMC is a family of approaches that can help to overcome this problem. In essence, these approaches run multiple chains in parallel, and, as well as standard proposals that allow movement within a chain, more complex proposals that allow positions to be swapped between chains are also allowed. These swaps provide an efficient way for chains to jump out of local modes and provide more efficient sampling.

The individual chains do not all have to be sampling from the same distribution, as long as one of them is sampling from the distribution of interest. A particularly interesting approach for posterior sampling is when we use what is known as a temperature schedule. Here, we run chains in each of N distributions where the nth chain is sampling from:

$$p_n(\boldsymbol{\theta}|\mathbf{X}) \propto p(\mathbf{X}|\boldsymbol{\theta})^{t_n} p(\boldsymbol{\theta}),$$



(a) A Metropolis–Hastings chain stuck in a single mode of a bi-modal density.

(b) Two chains, each stuck in different modes.

FIGURE 9.19 An example of Metropolis–Hastings getting stuck in individual modes of a bi-modal density.

where t_n ranges from 0 to 1. When t_n is equal to 0, $p_n(\boldsymbol{\theta}|\mathbf{X})$ is equal to the prior, and when t_n is equal to 1, the posterior. As t_n increases from 0, the distribution becomes less like the prior and more like the posterior. At each step in the sampler, we can update individual chains according to standard Metropolis–Hastings proposals or swap the states of adjacent chains (i.e. between the nth and the $n+1$th chain). We keep track of the state of all chains, but it is only the samples from the $t_n = 1$ chain that are of interest (samples from the posterior).

The rationale behind such an approach is that samplers in chains with low values of t_n should be able to move more easily around the space than those with higher values of t_n. Allowing swaps between adjacent chains means that this exploration can percolate up to the chains with higher t_n, giving them the opportunity to make larger moves and not get stuck in local optima.

For more detail on population MCMC and associated methods, see the suggested reading at the end of the chapter.

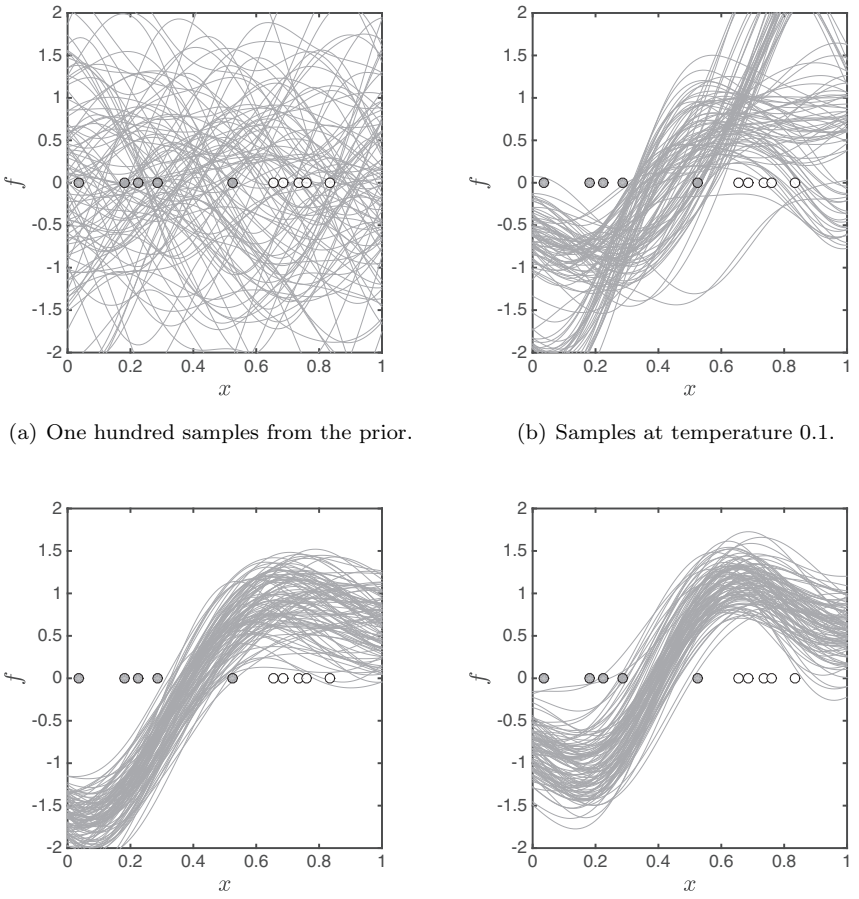
9.5.4 Sequential Monte Carlo

The final method we will look at is **sequential Monte Carlo** (SMC), often also called *particle filtering*. SMC techniques have been used widely to perform real-time inference for distributions that change over time (for example, tracking the movement of an object).

In its simplest setting, SMC represents the distribution over parameters at time t via a set of samples. These samples are then modified when we see the new observation. The simplest form of modification is to weight each of the current samples by the likelihood of the new observation and then create a new population by choosing current samples with probability proportional to their likelihood and applying some

random modification to them. Crucially, samples are chosen from the current population with replacement, meaning that the same current sample could be chosen more than once. The modification to the parameters is often something simple, like moving them according to a simple Gaussian density (much like a Metropolis–Hastings proposal).

This simple process has been used in many applications due to its efficiency and speed, which are both particularly important in real-time tracking applications. More recently, it has also been shown to be useful for sampling for things that do not change over time. In these cases, time can be replaced with a temperature schedule (as described above). Starting with a large sample of the prior, we gradually increase the power to which the likelihood is raised, updating our sample at each step. Finally, we have a sample that can be interpreted as a sample from the posterior.



(a) One hundred samples from the prior.

(b) Samples at temperature 0.1.

(c) Samples at temperature 0.5.

(d) Samples at temperature 1.

FIGURE 9.20 GP classification with SMC.

Figure 9.20 shows this in action to sample from the posterior over latent functions for a one-dimensional GP classifier (MATLAB script: `gp_smc.m`) using the probit likelihood we used earlier in this chapter. We start with $N = 10$ observed data points in two classes (those in class 0 are denoted by grey circles, those in class 1 by white circles) and 100 sampled latent functions from the prior (Figure 9.20(a)), denoted by $\mathbf{f}^1, \ldots, \mathbf{f}^S$. Note that, in the plot, rather than plotting the samples directly (they are only defined at the values of x in the training set), we have plotted the functions given by doing a GP regression from the prior samples onto a fine grid over x. Assuming a temperature schedule that increases by 0.1 at each stage, we give each of these samples a weight of

$$w_s = \left[\prod_{n=1}^{N} P(t_n = 1|f_n^s)^{t_n} (1 - P(t_n = 1|f_n^s))^{1-t_n} \right]^{0.1} \mathcal{N}(\mathbf{f}^s|\mathbf{C})$$

We then normalise this weight and use the normalised weight to resample a new population of 100 samples. For each new sample, we move it slightly away from the original (say $\mathbf{f}^s$) by sampling from a Gaussian with mean $\mathbf{f}^s$ and a covariance matrix equal to $0.05\mathbf{C}$. This new population of samples can be seen in Figure 9.20(b), where we can see that already the samples look reasonable – having lower values on the left (class 0) than on the right (class 1). This process is repeated with the likelihood raised to increasing temperature values for each temperature up to 1 $(0.2, 0.3, \ldots, 1)$, at which point the samples have been weighted and resampled based on their posterior values. The population of samples at temperatures of 0.5 and 1.0 can be seen in Figures 9.20(c) and 9.20(d), respectively. The samples from the highest temperature look like what we might expect from the posterior – they are heavily concentrated around functions that are low where we see examples in class 0 and high where we see examples in class 1. In this example, we can compare them with the samples from the true posterior as shown in Figure 9.4(c) and they look pretty similar.

Using SMC in this way, we cannot really quantify how close to the true posterior our samples are. For example, when we regenerate samples, we pick one of the previous ones and perturb it a little. Choice of this perturbation will affect our final samples. However, the simplicity of the process has led to it being used in various applications.

9.6 CHAPTER SUMMARY

In this chapter we have built upon the brief introduction to Metropolis–Hastings that we provided in Chapter 4. As well as providing more detail on Metropolis–Hastings, we have introduced Gibbs sampling and briefly introduced various other sampling approaches. All of these methods are widely used for generating posterior samples that can be used to make predictions across many machine learning applications, and a knowledge of MCMC techniques is increasingly important for those working in machine learning.

Although theoretically quite straightforward, MCMC techniques are often tricky to use, particularly when sampling from complex distributions that exhibit high correlation between variables, or multiple modes. Being able to assess whether chains have converged and assessing the autocorrelation of the samples being produced are vital to producing useful samples. Although we have covered these areas, we strongly

recommend practising building and evaluating sampling methods – it is the only way to really get a feel for these powerful approaches.

There is much ongoing research in this area, and many techniques that we have not described. However, we hope that from this introduction you are in a position to explore this area further and are able to make use of MCMC techniques. The suggested reading below provides a lot more detail in the areas we have covered, and more.

9.7 EXERCISES

9.1 For some one-dimensional input data, construct a GP RBF covariance matrix. Use Gibbs sampling to sample functions from this GP prior (samples from a GP prior are just samples from a multivariate Gaussian). Compare the samples with values obtained by sampling directly from the Gaussian. In particular, compute the autocorrelation of the function at one of the inputs across the two methods.

9.2 Use Metropolis–Hastings instead of Gibbs sampling to sample from the GP classification model. Note that you will no longer need to perform the auxiliary variable trick. Compare the predictions with those obtained from the Gibbs sampling approach.

9.3 Using the provided code for performing Gibbs sampling for binary GP classification, compute the $\hat{R}$ value for one of the training latent function values. How many samples need to be drawn before $\hat{R}$ is lower than 1.01?

9.4 For the same GP model as in Exercise 9.3, compute the autocorrelation of one of the training latent function values. How much thinning would be required to obtain independent samples?

9.5 Generate a regression dataset by sampling a function from a GP prior and then adding some noise. Use ABC to sample from the posterior. Each sample should be generated by first sampling from the GP prior and then adding noise. Use the Euclidean distance between the generated data and the original data to determine acceptance (experiment with different levels of tolerance). How efficient is your sampler? How do the samples compare with those from the true posterior (see Section 8.2.3)?

9.8 FURTHER READING

[1] Arnaud Doucet, Nando de Freitas, and Neil Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, first edition, 2001.

An excellent introduction to many sequential Monte Carlo algorithms.

[2] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, third edition, 2014.

This is an excellent resource for practical Bayesian inference. In particular, it provides a solid introduction to several sampling techniques as well as procedures for determining if Metropolis–Hastings and other sampling algorithms have converged. It also includes discussion of suitable acceptance rates when sampling from different distributions.

[3] Walter Wilks, Sylvia Richardson, and David Spegelhalter. *Markov Chain Monte Carlo in Practice*. Springer, first edition, 1996.

An excellent and practical introduction to many facets of MCMC approaches.

CHAPTER 10

Advanced Mixture Modelling

In Chapter 6 we introduced mixture models as a statistical method for clustering. Mixture models allow us to model complex distributions as combinations of simpler ones. This is particularly useful for clustering, as data that we might wish to cluster is likely to have multiple modes and it is natural to imagine each mode (cluster) being modelled by a separate distribution. To introduce inference in mixture models, we used the EM algorithm, which produces point estimates of the parameters of each component, and the component membership probabilities of each data point.

Mixture models are used across machine learning and so we have decided to devote a whole chapter to describe things that we didn't have space for in the clustering chapter. We will start by showing how we can use Gibbs sampling as a nice alternative to the EM algorithm. As well as giving samples from the posterior distribution over clusterings (rather than point estimates), the Gibbs sampler allows us to create an infinite mixture model, freeing us from having to specify the precise number of components in the data.

The infinite mixture model is closely related to a stochastic process that has gained a lot of traction in machine learning in recent years – the Dirichlet process. We will provide a high-level introduction to Dirichlet processes and their extension to Hierarchical Dirichlet processes.

Finally, topic models have become popular within machine learning, particularly for text modelling. These models relax the assumption that each data point was generated by one of the components, and allow data points to be modelled by multiple components. We will briefly discuss the most popular topic model – Latent Dirichlet Allocation.

10.1 A GIBBS SAMPLER FOR MIXTURE MODELS

In Chapter 9 we introduced Gibbs sampling, a Markov Chain Monte Carlo algorithm for sampling from joint distributions for which conditional distributions over all parameters (conditioned on the others) are available. With suitable prior distribution choices, mixture models fall into this category and Gibbs sampling is a popular alternative to the EM algorithm we described in Section 6.3.3. To illustrate this, we will construct a Gibbs sampler for a two-dimensional Gaussian Mixture.

Figure 10.1 shows a dataset that appears to consist of three clusters. Data in each cluster looks like it could have been derived from a two-dimensional Gaussian density, and we will assume in this example that each Gaussian has an identity

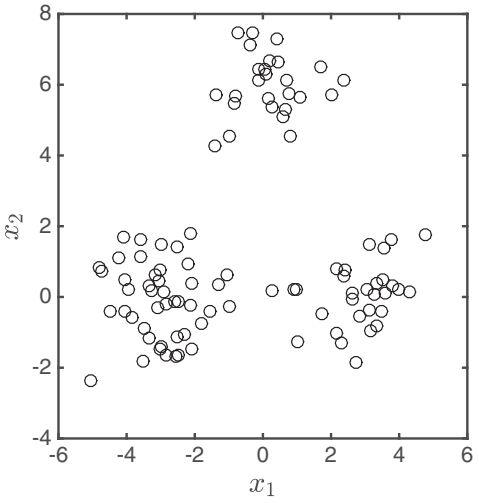


FIGURE 10.1 A dataset consisting of three Gaussian clusters.

covariance matrix (extending the model to include unknown covariance matrices is possible, but we have omitted it here, as the extra algebra might be a bit off-putting). We would therefore like to perform inference over the means of the Gaussians as well as infer which Gaussian each particular data point came from. We will do this with Gibbs sampling.

The mixture model likelihood for the nth observation ($\mathbf{x}_n$) is given by

$$p(\mathbf{x}_n|\boldsymbol{\pi}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K) = \sum_{k=1}^{K} \pi_k p(\mathbf{x}_n|\boldsymbol{\mu}_k),$$

where K is the number of components, $\boldsymbol{\pi} = [\pi_1, \ldots, \pi_K]^\mathsf{T}$ is the vector of prior component probabilities and, in this case, $p(\mathbf{x}_n|\boldsymbol{\mu}_k) = \mathcal{N}(\boldsymbol{\mu}_k, \mathbf{I})$.

When defining a mixture model, we are making the implicit assumption that each data point originated from one (and only one) of the K components (we will look at how this can be relaxed in Section 10.5). Part of our aim is to perform inference over which component each data point came from. To this end, we define a set of $N \times K$ binary parameters z_{nk}, where $z_{nk} = 1$ if observation n belongs to the kth component and 0 otherwise. Formally, if $z_{nk} = 1$, then $z_{nj} = 0$ for all $j \neq k$. We will collect all of these binary variables into an $N \times K$ matrix $\mathbf{Z}$ and we will use $\mathbf{z}_n$ to denote the $K \times 1$ binary membership vector for the nth data point.

If we know that $z_{nk} = 1$, our likelihood becomes

$$p(\mathbf{x}_n, \mathbf{z}_n|\boldsymbol{\pi}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K) = \pi_k p(\mathbf{x}_n|\boldsymbol{\mu}_k),$$

i.e. the likelihood of $\mathbf{x}_n$ for the component that produced it, multiplied by the prior probability of that component producing a data point (π_k). In general, it's

notationally easier to write this as

$$p(\mathbf{x}_n, \mathbf{z}_n | \boldsymbol{\pi}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K) = \prod_{k=1}^{K} [\pi_k p(\mathbf{x}_n | \boldsymbol{\mu}_k)]^{z_{nk}},$$

which is identical – raising the likelihood of each component to the power z_{nk} has the effect of *switching on* the component to which this data point is assigned.

In reality we don't know $\mathbf{Z}$ – this is one of the things we are trying to learn. We therefore have three sets of unknown parameters: the means, $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$, the component priors, $\boldsymbol{\pi}$, and the component memberships $\mathbf{Z}$. Gibbs sampling will give us samples from the joint posterior distribution of these three sets of variables. The final definitions we need are the various prior densities. We will assume a Gaussian prior over each of the mean vectors with mean $\boldsymbol{\mu}_0$ and covariance $\boldsymbol{\Sigma}_0$:

$$p(\boldsymbol{\mu}_k | \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0).$$

We will also assume that, under the prior, the different mean vectors are independent, allowing us to compute the joint prior density by taking a product over the K individual priors:

$$p(\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K | \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) = \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{\mu}_k | \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0).$$

What is the prior for z_{nk}? If we think of our model generatively, we sample from the kth component with probability π_k. Therefore, isn't the prior probability that $z_{nk} = 1$ simply π_k? Not quite – we need to be a little more specific. If we set $z_{nk} = 1$, we also have to set $z_{nj} = 0$ for all other components. It is therefore the whole vector $\mathbf{z}_n$ that we are setting, of which only one element can be 1. $\mathbf{z}_n$ is therefore one draw from a multinomial distribution with parameters $\boldsymbol{\pi}$ (see Section 2.3.3). The multinomial distribution is defined as

$$P(\mathbf{z}_n | \boldsymbol{\pi}) = \frac{\left(\sum_k z_{nk}\right)!}{\prod_{k=1}^{K} z_{nk}!} \prod_{k=1}^{K} \pi_k^{z_{nk}}.$$

Because we are only making one draw from the multinomial, both terms in the fraction evaluate to 1 ($\sum_k z_{nk} = 1$, and $z_{nk}! = 1$ for all k). The distribution therefore reduces to

$$P(\mathbf{z}_n | \boldsymbol{\pi}) = \prod_{k=1}^{K} \pi_k^{z_{nk}},$$

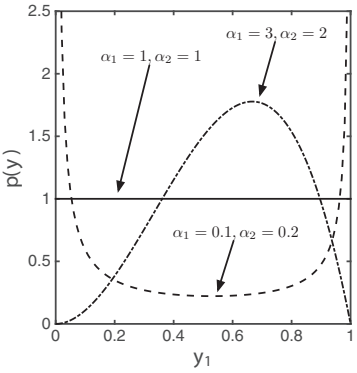
where only one term in the product is switched on (the one corresponding to the chosen component).

Comment 10.1 – The Dirichlet Distribution: A Dirichlet distribution in M dimensions is defined over all vectors of length M where each value is between 0 and 1 and the sum of the values equals 1. In particular, for vectors $\mathbf{y} = [y_1, \ldots, y_M]^\intercal$, the Dirichlet is defined over all vectors that satisfy:$0 \leq y_m \leq 1, \quad \sum_m y_m = 1$. The pdf is given by

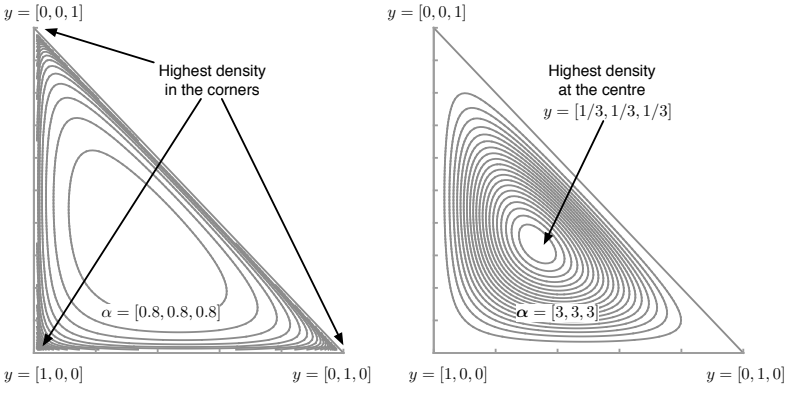
$$p(\mathbf{y}|\boldsymbol{\alpha}) = Dir(\alpha_1, \ldots, \alpha_M) = \frac{\Gamma\left(\sum_m \alpha_m\right)}{\prod_m \Gamma(\alpha_m)} \prod_{m=1}^{M} y_m^{\alpha_m - 1},$$

where $\Gamma(\cdot)$ is the gamma function that we have seen before (see, e.g. Section 2.5.2). As the vector has to sum to 1, the distribution is therefore really defined over an $M-1$ dimensional subspace of the M dimensions. This is because the Mth dimension is completely determined by the other $M-1$ ($y_M = 1 - \sum_{m=1}^{M-1} y_m$).

For $M = 2$, we therefore have a one-dimensional subspace (a line) on which the Dirichlet defines a probability density over y_1 and $y_2 = 1 - y_1$. Three examples, with different parameter values, are shown on the right. On the x axis is the value for y_1. The value for y_2 is not shown (it has to be $1 - y_1$).



For $M = 3$, the valid points lie on the two-dimensional triangular surface with corners at $[1,0,0]$, $[0,1,0]$ and $[0,0,1]$. By varying the parameters, the distribution can be varied between something where the highest density is in the corners (producing sparse probability vectors with a small number of non-zero values) and something where it is in the centre. Examples with $\boldsymbol{\alpha} = [0.8, 0.8, 0.8]$ (left plot; high density in the corners) and $\boldsymbol{\alpha} = [3,3,3]$ (right plot; high density in the centre) can be seen below.



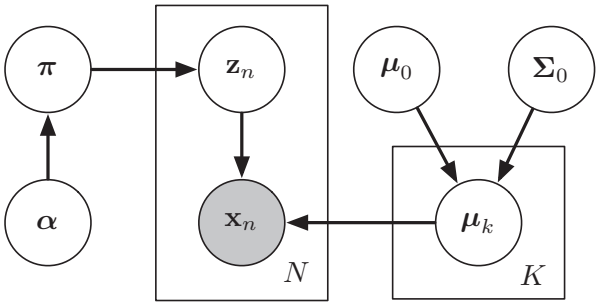


FIGURE 10.2 A plates diagram of the mixture model.

Finally, we need a prior for $\boldsymbol{\pi}$. As this is a vector of probabilities, all of its values have to be between 0 and 1 and the sum of the values must be equal to 1. A suitable prior distribution defined over vectors with these properties is the **Dirichlet** distribution (see Comment 10.1 and Equation 5.7). The pdf of the Dirichlet is

$$p(\boldsymbol{\pi}|\boldsymbol{\alpha}) = \frac{\Gamma\left(\sum_{k=1}^{K}\alpha_k\right)}{\prod_{k=1}^{K}\Gamma(\alpha_k)}\prod_{k=1}^{K}\pi_k^{\alpha_k-1}.$$

This completes our model definition and a plates diagram of the model can be seen in Figure 10.2.

The joint posterior density over the three sets of unknown parameters is:

$$p(\mathbf{Z},\boldsymbol{\pi},\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_K|\mathbf{X},\alpha,\boldsymbol{\mu}_0,\boldsymbol{\Sigma}_0) \propto \left[\prod_{n=1}^{N}\prod_{k=1}^{K}[\pi_k p(\mathbf{x}_n|\boldsymbol{\mu}_k)]^{z_{nk}}\right] \times \left[\prod_{k=1}^{K}p(\boldsymbol{\mu}_k|\boldsymbol{\mu}_0,\boldsymbol{\Sigma}_0)\right]p(\boldsymbol{\pi}|\boldsymbol{\alpha}). \quad (10.1)$$

We saw in Chapter 9 that the conditional distributions for each parameter are proportional to this joint posterior. For each parameter we can therefore ignore any terms that don't involve it on the right hand side and use the terms that remain to work out the form of the conditional distribution. Starting with $\boldsymbol{\pi}$, if we only keep terms involving $\boldsymbol{\pi}$ on the right hand side, we are left with

$$p(\boldsymbol{\pi}|\ldots) \propto p(\boldsymbol{\pi}|\boldsymbol{\alpha})\prod_{n=1}^{N}\prod_{k=1}^{K}\pi_k^{z_{nk}}.$$

Expanding the Dirichlet prior (and ignoring the constant, as it doesn't involve $\boldsymbol{\pi}$),

$$p(\boldsymbol{\pi}|\ldots) \propto \left[\prod_{k=1}^{K}\pi_k^{\alpha_k-1}\right]\left[\prod_{n=1}^{N}\prod_{k=1}^{K}\pi_k^{z_{nk}}\right].$$

Because z_{nk} is binary, the second term can be rewritten as

$$\prod_{n=1}^{N}\prod_{k=1}^{K}\pi_k^{z_{nk}} = \prod_{k=1}^{K}\pi_k^{\sum_n z_{nk}},$$

and the whole thing can be expressed with one product over K:

$$p(\boldsymbol{\pi}|\ldots) \propto \prod_{k=1}^{K} \pi_k^{\alpha_k - 1 + \sum_n z_{nk}}.$$

So, the unnormalised conditional density for $\boldsymbol{\pi}$ has the form of a product over the components of $\boldsymbol{\pi}$, each raised to the power of $\alpha_k^* - 1$ where

$$\alpha_k^* = \alpha_k + \sum_{n=1}^{N} z_{nk}.$$

This is exactly the form of a Dirichlet distribution (see Comment 10.1) and so the conditional density of $\boldsymbol{\pi}$ has to be a Dirichlet. The fact that this distribution is a Dirichlet also tells us that the Dirichlet prior and the multinomial likelihood form a conjugate pair – a feature that has led to so-called Dirichlet-multinomial models being widely used, particularly in text modelling applications. Our Dirichlet is given as

$$p(\boldsymbol{\pi}|\ldots) = Dir(\alpha_1 + \sum_n z_{n1}, \ldots, \alpha_K + \sum_n z_{nK}). \tag{10.2}$$

Next we will derive the conditional distribution for $\boldsymbol{\mu}_k$. Taking the terms involving $\boldsymbol{\mu}_k$ from the right hand side of Equation 10.1, we are left with

$$p(\boldsymbol{\mu}_k|\ldots) \propto p(\boldsymbol{\mu}_k|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \prod_{n=1}^{N} p(\mathbf{x}_n|\boldsymbol{\mu}_k)^{z_{nk}}. \tag{10.3}$$

The two terms on the right hand side could be interpreted as a Gaussian prior and a Gaussian likelihood and therefore we know that the conditional distribution will also be Gaussian. We leave it as an exercise (see Exercise 10.1) to show that the conditional distribution is

$$p(\boldsymbol{\mu}_k|\ldots) = \mathcal{N}(\mathbf{a}, \mathbf{B}), \tag{10.4}$$

where

$$\mathbf{B} = \left[\boldsymbol{\Sigma}_0^{-1} + \left(\sum_n z_{nk}\right)\mathbf{I}\right]^{-1}, \quad \mathbf{a} = \mathbf{B}\left[\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0 + \sum_n z_{nk}\mathbf{x}_n\right].$$

Finally, we need the conditional distribution for $\mathbf{z}_n$. Isolating terms involving $\mathbf{z}_n$ in the right hand side of Equation 10.1 we are left with

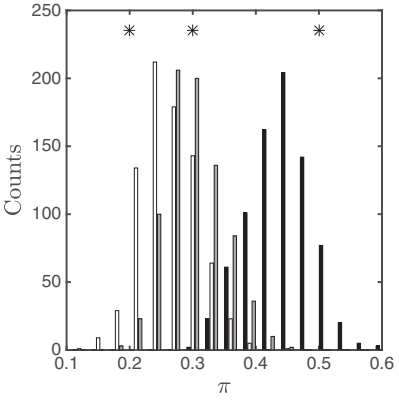
$$P(\mathbf{z}_n|\ldots) \propto \prod_{k=1}^{K} \left[\pi_k p(\mathbf{x}_n|\boldsymbol{\mu}_k)\right]^{z_{nk}}.$$

Consider the vector $\mathbf{z}_n$ that has its 1 in the kth position (all other entries must be 0). We will write the probability of this vector as $P(z_{nk} = 1|\ldots)$, with the implicit assumption that $z_{nj} = 0$ for all $j \neq k$. The unnormalised probability is

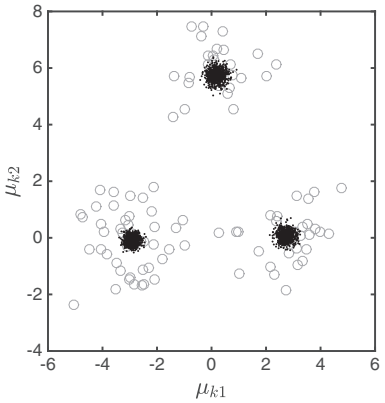
$$P(z_{nk} = 1|\ldots) \propto \pi_k p(\mathbf{x}_n|\boldsymbol{\mu}_k).$$

To normalise, all we need to do is divide by the summation over all possible values of $\mathbf{z}_n$, which is simply $\sum_j \pi_j p(\mathbf{x}_n|\boldsymbol{\mu}_j)$. In particular

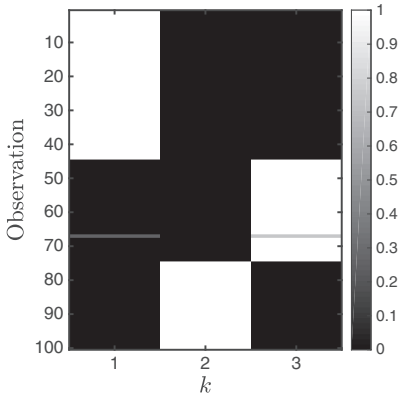
$$P(z_{nk} = 1|\ldots) = \frac{\pi_k p(\mathbf{x}_n|\boldsymbol{\mu}_k)}{\sum_{j=1}^{K} \pi_j p(\mathbf{x}_n|\boldsymbol{\mu}_j)}. \tag{10.5}$$



(a) Marginal posteriors for π_k. Stars show the true values.



(b) Posterior samples of $\boldsymbol{\mu}_k$ (dots show samples and light circles the data).



(c) Posterior probability of cluster membership.

FIGURE 10.3 Results of applying the Gibbs sampler to the mixture model. One thousand samples were drawn, with the first 200 discarded as a burn-in phase.

We now have all of the conditional distributions required to build our Gibbs sampler. The procedure is as follows:

1. Initialise $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ and $\boldsymbol{\pi}$ with samples from their respective prior densities.
2. Resample each $\mathbf{x}_n$ conditioned on $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ and $\boldsymbol{\pi}$, via Equation 10.5.
3. Resample each $\boldsymbol{\mu}_k$ conditioned on $\mathbf{Z}$, via Equation 10.4.
4. Resample $\boldsymbol{\pi}$ conditioned on $\mathbf{Z}$, via Equation 10.2.

5. Return to step 2 until enough samples have been generated.

We can see the result of this procedure in Figure 10.3 (MATLAB script: `gmix2d.m`). Here we have taken 1000 posterior samples and discarded the first 100 as a burn-in (see Section 9.4.1). K was set to 3 and $\boldsymbol{\alpha} = [1/3, 1/3, 1/3]$. $\boldsymbol{\mu}_0$ was set to a vector of zeros, and $\boldsymbol{\Sigma}_0$ to the identity matrix.

In Figure 10.3(a) we can see histograms of the posterior samples for the components of $\boldsymbol{\pi}$. The stars show the true values, and it is clear that the true values correspond to reasonably high posterior values. Note that, because we are only sampling $N = 100$ data points, we would expect to observe some deviation from the true value. Figure 10.3(b) shows posterior samples of the three cluster means. The samples are shown as black dots with the data shown (for reference) as lighter circles. We can see that the posterior samples for the means correspond very strongly to the positions of the true means. Finally, in Figure 10.3(c) we show the posterior probability of cluster membership, obtained by averaging over the number of times each data point was assigned to each cluster. In this case we can see that points are always assigned to the correct clusters (in the figure the data points (rows) are grouped in their clusters).

Gibbs sampling is popular for mixture models where the various prior and likelihood terms are chosen such that we can compute and sample from all of the conditional distributions. In this example, we assumed the covariance of the Gaussian components was known but this is not necessary. There is a conjugate prior for the mean and covariance of a Gaussian (the Normal-Inverse-Wishart) which would allow us to obtain posterior distributions over both the mean and covariance. Other conjugate prior-likelihood pairs, such as the beta-binomial and the Dirichlet-multinomial, also result in computationally attractive sampling schemes. To work through an example, see Exercise 10.3.

The sampler we have described appears to work very well (albeit on a very simple example). However, it actually only explores a small part of the posterior. We have defined three cluster components (call them 1, 2 and 3). The assignment of these components to the data clusters in Figure 10.1 is arbitrary. We could assign number 1 to the bottom left cluster, number 2 to the bottom right and number 3 to the top one, or any other combination. The value of the posterior for each assignment would be identical (the model doesn't change when we change the name of a cluster) and so we can deduce that the posterior has multiple identical nodes, one for each way of assigning the cluster names to the actual clusters. If you are not sure about this point, imagining implementing the sampler. You would perhaps have arrays storing the current values of $\boldsymbol{\mu}$, $\boldsymbol{\pi}$ and $\mathbf{Z}$. Let's assume that, in each case, the column of the array indicates the cluster (e.g. the kth column of $\mathbf{Z}$ gives the memberships to cluster k and the kth column of $\boldsymbol{\mu}$ gives the value of $\boldsymbol{\mu}_k$). If you permuted the columns of all arrays in the same way, the model wouldn't have changed at all.

Now, a good sampler ought to explore the whole of the posterior, but, in our example, it has only explored one mode (one mapping of columns to cluster). Because we know that all of the modes are identical, this is not a practical problem. In fact, it would make it harder to interpret the output of the sampler if it did explore them all. However, it is important to bear in mind, particularly if we are monitoring convergence using multiple chains, as two chains are unlikely to converge to the same mode (see Exercise 10.2).

So far, we have applied Gibbs sampling (from Chapter 9) to a mixture model (from Chapter 6). Although this is useful, we haven't done anything particularly

new. We will now move on to show that within the Gibbs sampler, we can avoid having to specify the number of components (K). In fact, we will infer the posterior distribution over the number of components from the data. We will do this by first extending the mixture model to having an infinite number of components and then see how this can also be viewed from the perspective of **Dirichlet processes**. Before we do this, we must first see how Gibbs sampling allows us to get rid of some variables from the model (collapsed Gibbs sampling).

10.2 COLLAPSED GIBBS SAMPLING

In the previous section we described a Gibbs sampler for a mixture model. Part of the model was the prior distribution over mixture components, $\boldsymbol{\pi}$. The kth component of $\boldsymbol{\pi}$, π_k tells us the prior probability a data point should be assigned to component k (i.e. the probability before we see the actual value of the data point). As such, the posterior values of this variable reflect the proportion of points in each cluster.

We have some additional control over this parameter through a Dirichlet prior. By varying the Dirichlet parameters ($\boldsymbol{\alpha}$), we are able to provide an a priori preference for vectors $\boldsymbol{\pi}$ that are quite uniform (all values are roughly the same) or are quite non-uniform (some very high and some very low values). Being able to specify a prior over $\boldsymbol{\pi}$ feels quite useful, but are we actually interested in the posterior? Probably not – we are normally only interested in how the data points cluster. Because of the conjugacy of the Dirichlet prior and multinomial likelihood, it turns out that we can actually remove $\boldsymbol{\pi}$ from our sampler. Doing so will open up a useful way of extending our model to overcome the limitation of having to specify K.

Consider resampling the cluster membership of the nth data point ($\mathbf{z}_n$). In the previous section, we derived the conditional distribution to be

$$P(z_{nk} = 1 | \ldots) \propto \pi_k p(\mathbf{x}_n | \boldsymbol{\mu}_k). \tag{10.6}$$

At this stage in the sampler, what do we know about $\boldsymbol{\pi}$? We know its prior (a Dirichlet with parameters $\boldsymbol{\alpha}$) and we know the likelihood of the observed data (all of the other $\mathbf{z}_m$) given $\boldsymbol{\pi}$ (a multinomial). As the Dirichlet prior is conjugate to the multinomial, we can compute the posterior over $\boldsymbol{\pi}$. Note that this is conditional on all of the other assignments so the posterior for $\boldsymbol{\pi}$ will potentially be different when resampling each data point.

To derive this posterior, we need to expand our notation a bit. We will use $\mathbf{Z}^{-n}$ to be the set of all assignments *except* the nth one, and c_k^{-n} to be the number of objects assigned to component k, not including the assignment of $\mathbf{x}_n$, i.e. $c_k^{-n} = \sum_{m \neq n} z_{mk}$.

The likelihood of the other observations given $\boldsymbol{\pi}$ is given by

$$P(\mathbf{Z}^{-n} | \boldsymbol{\pi}) \propto \prod_{m \neq n} \prod_{k=1}^{K} \pi_k^{z_{mk}} = \prod_{k=1}^{K} \pi_k^{c_k^{-n}},$$

where we have ignored the normalisation constant, as it does not depend on $\boldsymbol{\pi}$. The prior is Dirichlet with parameter $\boldsymbol{\alpha}$, which is (again ignoring the constant) given by

$$p(\boldsymbol{\pi} | \boldsymbol{\alpha}) \propto \prod_{k=1}^{K} \pi_k^{\alpha_k - 1}.$$

Because the prior and likelihood are conjugate, we know that the posterior will

also be a Dirichlet. The parameters of this posterior Dirichlet (call them $\boldsymbol{\beta}$) can be obtained by equating the posterior to the product of the likelihood and prior:

$$\begin{aligned}\prod_{k=1}^{K}\pi_k^{\beta_k-1} &= \left[\prod_{k=1}^{K}\pi_k^{\alpha_k-1}\right]\left[\prod_{k=1}^{K}\pi_k^{c_k^{-n}}\right]\\ &= \prod_{k=1}^{K}\pi_k^{\alpha_k+c_k^{-n}-1}\\ \beta_k &= \alpha_k + c_k^{-n}.\end{aligned}$$

The posterior Dirichlet parameter for component k is simply the prior parameter plus the number of data points in the component. So, at this point in the sampler, there is no reason why we could not sample a value for $\boldsymbol{\pi}$ conditioned on $\mathbf{Z}^{-n}$ and $\boldsymbol{\alpha}$ and then use this to sample $\mathbf{z}_n$. But, we can actually go one step further and remove $\boldsymbol{\pi}$ from the expression completely.

If we were to sample a value for $\boldsymbol{\pi}$ and then $\mathbf{z}_n$, we would effectively be drawing a sample from the joint density over $\mathbf{z}_n$ and $\boldsymbol{\pi}$ conditioned on everything else: $p(\mathbf{z}_n, \boldsymbol{\pi}|\mathbf{Z}^{-n}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \alpha, \mathbf{x}_n)$. Decomposing this into the two separate steps gives

$$p(\mathbf{z}_n, \boldsymbol{\pi}|\mathbf{Z}^{-n}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \alpha, \mathbf{x}_n) = p(\mathbf{z}_n|\boldsymbol{\pi}, \mathbf{x}_n, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K)p(\boldsymbol{\pi}|\mathbf{Z}^{-n}, \boldsymbol{\alpha}),$$

where $p(\boldsymbol{\pi}|\mathbf{Z}^{-n}, \boldsymbol{\alpha})$ is the posterior Dirichlet that we just computed. Expanding the first term in the right hand side (and ignoring the normalisation over k), we have

$$p(\mathbf{z}_n, \boldsymbol{\pi}|\mathbf{Z}^{-n}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \alpha, \mathbf{x}_n) \propto \left[\prod_{k=1}^{K}\pi_k p(\mathbf{x}_n|\boldsymbol{\mu}_k)\right]^{z_{nk}} p(\boldsymbol{\pi}|\mathbf{Z}^{-n}, \boldsymbol{\alpha}).$$

If we look at a particular value of $\mathbf{z}_n$, say the one where the kth element is 1 (and all others are zero), this probability becomes

$$p(z_{nk} = 1, \boldsymbol{\pi}|\mathbf{Z}^{-n}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \alpha, \mathbf{x}_n) \propto \pi_k p(\mathbf{x}_n|\boldsymbol{\mu}_k)p(\boldsymbol{\pi}|\mathbf{Z}^{-n}, \boldsymbol{\alpha}),$$

which, because π_k is (by definition) $P(\mathbf{z}_{nk} = 1|\boldsymbol{\pi})$ (for the $\mathbf{z}_n$ that has a 1 in the kth element), we can further rewrite as

$$p(z_{nk} = 1, \boldsymbol{\pi}|\mathbf{Z}^{-n}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \alpha, \mathbf{x}_n) \propto p(\mathbf{x}_n|\boldsymbol{\mu}_k)p(z_{nk} = 1|\boldsymbol{\pi})p(\boldsymbol{\pi}|\mathbf{Z}^{-n}, \boldsymbol{\alpha}).$$

If we integrate both sides with respect to $\boldsymbol{\pi}$, we can remove it completely from the expression (using $\boldsymbol{\mu}$ to denote $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$):

$$\begin{aligned}\int p(z_{nk} = 1, \boldsymbol{\pi}|\mathbf{Z}^{-n}, \boldsymbol{\mu}, \alpha, \mathbf{x}_n)\, d\boldsymbol{\pi} &\propto \int p(\mathbf{x}_n|\boldsymbol{\mu}_k)P(z_{nk} = 1|\boldsymbol{\pi})p(\boldsymbol{\pi}|\mathbf{Z}^{-n}, \boldsymbol{\alpha})\, d\boldsymbol{\pi}\\ P(z_{nk} = 1|\mathbf{Z}^{-n}, \boldsymbol{\mu}, \alpha, \mathbf{x}_n) &\propto p(\mathbf{x}_n|\boldsymbol{\mu}_k)\int P(z_{nk} = 1|\boldsymbol{\pi})p(\boldsymbol{\pi}|\mathbf{Z}^{-n}, \boldsymbol{\alpha})\, d\boldsymbol{\pi}\\ &\propto p(\mathbf{x}_n|\boldsymbol{\mu}_k)P(z_{nk} = 1|\mathbf{Z}^{-n}, \boldsymbol{\alpha}).\end{aligned}$$

So, if we can evaluate the integral on the right hand side, then we can update $\mathbf{z}_n$ without worrying about $\boldsymbol{\pi}$ at all. In fact, we need never worry about $\boldsymbol{\pi}$ in our sampler. Note that the right hand side of the final expression is proportional to the

probability we want. But, as with the original expression, it is easy to normalise: we simply divide by the sum across all of the components:

$$P(z_{nk}=1|\mathbf{Z}^{-n},\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_K,\alpha,\mathbf{x}_n)=\frac{p(\mathbf{x}_n|\boldsymbol{\mu}_k)P(z_{nk}=1|\mathbf{Z}^{-n},\boldsymbol{\alpha})}{\sum_{j=1}^{K}p(\mathbf{x}_n|\boldsymbol{\mu}_j)P(z_{nj}=1|\mathbf{Z}^{-n},\boldsymbol{\alpha})}.$$

So, we need to be able to evaluate the following integral:

$$P(z_{nk}=1|\mathbf{Z}^{-n},\boldsymbol{\alpha})=\int P(z_{nk}=1|\boldsymbol{\pi})p(\boldsymbol{\pi}|\mathbf{Z}^{-n},\boldsymbol{\alpha})\;d\boldsymbol{\pi}. \tag{10.7}$$

The first term is just π_k whilst the second term is the posterior Dirichlet that we derived earlier (with parameters $\beta_k=\alpha_k+c_k^{-n}$). Writing these out in full gives us

$$P(z_{nk}=1|\mathbf{Z}^{-n},\boldsymbol{\alpha})=\frac{\Gamma\left(\sum_{j=1}^{K}\beta_j\right)}{\prod_{j=1}^{K}\Gamma(\beta_j)}\int\pi_k\prod_{j=1}^{J}\pi_j^{\beta_j-1}\;d\boldsymbol{\pi},$$

where we have taken the Dirichlet constant out of the integral, as it does not depend on $\boldsymbol{\pi}$. If we define δ_{jk} as 1 if $j=k$ and 0 otherwise, we can rewrite the integrand to give

$$P(z_{nk}=1|\mathbf{Z}^{-n},\boldsymbol{\alpha})=\frac{\Gamma\left(\sum_{j=1}^{K}\beta_j\right)}{\prod_{j=1}^{K}\Gamma(\beta_j)}\int\prod_{j=1}^{J}\pi_j^{\beta_j+\delta_{jk}-1}\;d\boldsymbol{\pi}.$$

The integrand now looks like an unnormalised Dirichlet with parameters $\beta_j+\delta_{jk}$. Its integral *must* therefore be the inverse of the Dirichlet constant. Therefore, our expression becomes

$$P(z_{nk}=1|\mathbf{Z}^{-n},\boldsymbol{\alpha})=\frac{\Gamma\left(\sum_{j=1}^{K}\beta_j\right)}{\prod_{j=1}^{K}\Gamma(\beta_j)}\frac{\prod_{j=1}^{K}\Gamma(\beta_j+\delta_{jk})}{\Gamma\left(\sum_{j=1}^{K}\beta_j+\delta_{jk}\right)}.$$

This expression can be simplified considerably. Firstly, note that $\sum_{j=1}^{K}\delta_{jk}=1$ and therefore

$$\Gamma\left(\sum_{j=1}^{K}\beta_j+\delta_{jk}\right)=\Gamma\left(1+\sum_{j=1}^{K}\beta_j\right).$$

Secondly, because $\delta_{jk}=0$ for all $j\neq k$, all of the product terms in the numerator and denominator cancel except for the terms where $j=k$. This leaves

$$P(z_{nk}=1|\mathbf{Z}^{-n},\boldsymbol{\alpha})=\frac{\Gamma\left(\sum_{j=1}^{K}\beta_j\right)}{\Gamma\left(1+\sum_{j=1}^{K}\beta_j\right)}\frac{\Gamma(\beta_k+1)}{\Gamma(\beta_k)}.$$

To simplify further, we need to make use of a property of the gamma function. In particular

$$\Gamma(z+1)=z\Gamma(z)$$

In our expression we can make use of this twice and cancel almost everything:

$$P(z_{nk}=1|\mathbf{Z}^{-n},\boldsymbol{\alpha})=\frac{\Gamma\left(\sum_{j=1}^{K}\beta_j\right)}{\left(\sum_{j=1}^{K}\beta_j\right)\Gamma\left(\sum_{j=1}^{K}\beta_j\right)}\frac{\beta_k\Gamma(\beta_k)}{\Gamma(\beta_k)}=\frac{\beta_k}{\sum_{j=1}^{K}\beta_j}.$$

If we substitute $\beta_k = c_k^{-n} + \alpha_k$, we get

$$P(z_{nk} = 1|\mathbf{Z}^{-n}, \boldsymbol{\alpha}) = \frac{c_k^{-n} + \alpha_k}{\sum_{j=1}^{K} c_j^{-n} + \alpha_j},$$

which is a remarkably simple expression. It tells us that the prior probability of assigning data point n to component k is proportional to the number of other points in component k plus the prior Dirichlet parameter. In other words, data points are more likely to be assigned to bigger clusters (high c_k^{-n}) than smaller clusters (low c_k^{-n}). The strength of this clustering force is controlled by α_k. If α_k is high, it will dominate and the probability won't change much as the cluster changes size. If α_k is small, then the number of other data points in the cluster will dominate.

It is important to realise that this clustering force, controlled by α, is not a feature of marginalising $\boldsymbol{\pi}$ (the marginalisation doesn't really *change* the model); it is a feature of our use of the Dirichlet prior for $\boldsymbol{\pi}$. The two plots in Comment 10.1 show how changing $\boldsymbol{\alpha}$ changes the Dirichlet from having its mode somewhere near the center (high α values) to having modes at each of the corners (low α values). This corresponds exactly with what we see in our new marginalised prior expression. Consider a Dirichlet where all parameters are the same: $\alpha_k = \alpha \ \ \forall k$. High α values dominate the marginalised prior expression, resulting in roughly constant cluster membership probabilities. This would result in (on average) a uniform number of points in each component, which is also reflected in the mode of the prior having roughly equal probabilities for each component (see plot in Comment 10.1). Low values of α would leave the c_k^{-n} to dominate, resulting in all of the points being drawn to a small number of large components, which is reflected in the prior mode of $\boldsymbol{\pi}$ being in a corner or an edge.

The result of our marginalisation is a new expression for the resampling of $\mathbf{z}_n$:

$$P(z_{nk} = 1|\mathbf{Z}^{-n}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\alpha}, \mathbf{x}_n) \propto \frac{c_k^{-n} + \alpha_k}{\sum_{j=1}^{K} c_j^{-n} + \alpha_j} p(\mathbf{x}_n|\boldsymbol{\mu}_k),$$

which we can substitute into our sampling procedure, allowing us to never bother sampling a value for $\boldsymbol{\pi}$. Note that, if we were interested in $\boldsymbol{\pi}$, we could still sample a value as before, but we will no longer use it anywhere else in the sampler. This is known as a **Collapsed Gibbs** sampler, as we have *collapsed* $\boldsymbol{\pi}$ from the sampler. The two parts in this expression can be considered as a prior:

$$\frac{c_k^{-n} + \alpha_k}{\sum_{j=1}^{K} c_j^{-n} + \alpha_j}$$

and a likelihood ($p(\mathbf{x}_n|\boldsymbol{\mu}_k)$). The prior is conditioned on all of the other assignments so it doesn't look like a prior we have seen before. However, it tells us the probabilities of the different components *without* considering the data, and so in that sense can very naturally be thought of as a prior and we will refer to it as that in the following.

Before we move on to see how this helps us overcome the problem of determining the number of components, we can use this procedure to remove another set of variables from our model: $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$. Using exactly the same reasoning, we could, when resampling $\mathbf{z}_n$ compute the posterior over $\boldsymbol{\mu}_k$ based on the current assignments without the nth point ($\mathbf{Z}^{-n}$). We could then, if we wished, sample a $\boldsymbol{\mu}_k$ and then

compute $p(\mathbf{x}_n|\boldsymbol{\mu}_k)$. But, the prior over $\boldsymbol{\mu}_k$ and the likelihood $p(\mathbf{x}_n|\boldsymbol{\mu}_k)$ are both Gaussian and therefore we can integrate out $\boldsymbol{\mu}_k$. We leave it as an exercise (see Exercise 10.4) to show that the posterior over $\boldsymbol{\mu}_k$ is given by

$$p(\boldsymbol{\mu}_k|\mathbf{Z}^{-n}, \mathbf{X}^{-n}, \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) = \mathcal{N}(\mathbf{d}, \mathbf{E}),$$

where $\mathbf{X}^n$ is all of the data apart from $\mathbf{x}_n$ and

$$\mathbf{E} = \left(\boldsymbol{\Sigma}_0^{-1} + \mathbf{I}\sum_{m\neq n} z_{mk}\right)^{-1}, \quad \mathbf{d} = \mathbf{E}\left(\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0 + \sum_{m\neq n} z_{mk}\mathbf{x}_m\right),$$

and therefore

$$p(\mathbf{x}_n|\mathbf{Z}^{-n}, \mathbf{X}^{-n}, \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) = \mathcal{N}(\mathbf{d}, \mathbf{E} + \mathbf{I}). \tag{10.8}$$

This removes the need to sample $\boldsymbol{\mu}_k$ and leaves us with the following expression for resampling $\mathbf{z}_n$:

$$P(z_{nk} = 1|\mathbf{x}_n, \mathbf{Z}^{-n}, \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0, \boldsymbol{\alpha}) \propto \frac{c_k^{-n} + \alpha_k}{\sum_{j=1}^K c_j^{-n} + \alpha_j}\mathcal{N}(\mathbf{x}_n|\mathbf{d}, \mathbf{E} + \mathbf{I}).$$

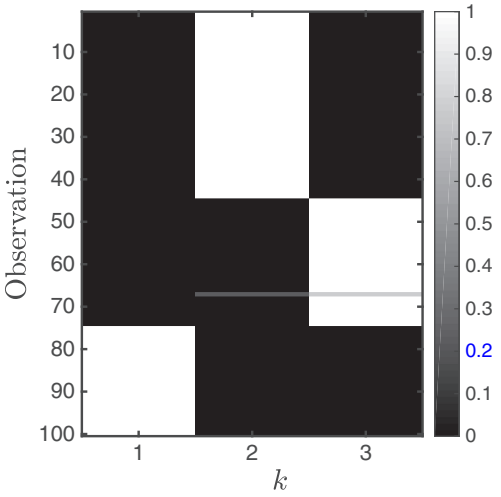


FIGURE 10.4 The posterior component memberships for the collapsed Gibbs sampler.

Our Gibbs sampler now only consists of resampling each $\mathbf{z}_n$ according to this expression! To demonstrate that this works, in Figure 10.4 we have plotted the posterior probabilities of component membership (MATLAB script: `gmix2d_marg.m`). We can see that these are very similar to those shown in Figure 10.3(c) (with a change in cluster labels – see the discussion of multiple symmetric modes at the end

of the last section).

So far, we have not really justified *why* we might prefer a collapsed Gibbs sampler over sampling everything. Our primary motivation is that marginalising $\boldsymbol{\pi}$ will enable us to overcome the problem of having to specify the number of components. However, it has other benefits. In particular, removing parameters can often improve MCMC convergence (e.g. cluster memberships can change more quickly if not conditioned on particular values of $\boldsymbol{\mu}_k$ etc). We won't go into this in detail now but there are some pointers to further reading at the end of the chapter (see Exercise 10.5).

10.3 AN INFINITE MIXTURE MODEL

In the previous section we showed how $\boldsymbol{\pi}$ could be marginalised from the update for $\mathbf{z}_n$ to give a new expression for the prior probability of joining component k:

$$P(z_{nk} = 1|\boldsymbol{\alpha}, \mathbf{Z}^{-n}) = \frac{c_k^{-n} + \alpha_k}{\sum_{j=1}^{K} c_j^{-n} + \alpha_j}.$$

Often we will set all of the Dirichlet prior parameters, $\boldsymbol{\alpha}$, to the same value (we have no a priori reason to prefer any particular component over any other) and, for reasons that will become apparent in a little while, we will set this parameter to α/K – i.e. a constant tunable parameter α divided by the number of components K. Our Dirichlet prior is therefore

$$p(\boldsymbol{\pi}|\alpha) = Dir(\alpha/K, \ldots, \alpha/K)$$

and our sampling prior for $z_{nk} = 1$ becomes

$$P(z_{nk} = 1|\alpha, \mathbf{Z}^{-n}) = \frac{c_k^{-n} + \alpha/K}{\alpha + N - 1},$$

where we have used the fact that $\sum_{j=1}^{K} \alpha/K = \alpha$ and $\sum_{j=1}^{K} c_j^{-n} = N-1$, because all N data points must be in a component except the nth one that we are reassigning.

What happens to this expression if we make K very large? In particular, what if we have an infinite number of components: $K = \infty$? Well, if $K = \infty$, then $\alpha/K = 0$ (assuming that $\alpha > 0$, which it always will be). The prior expression becomes

$$P(z_{nk} = 1|\alpha, \mathbf{Z}^{-n}) = \frac{c_k^{-n}}{\alpha + N - 1}.$$

So, when resampling the allocation of the nth data point, the prior probability that it goes into the kth component is proportional to the number of objects currently in that component. The count, c_k^{-n}, can only be greater than zero for at most $N-1$ of the infinite number of components. What is the probability of assigning to one of the currently empty ones? We cannot compute the probability of any individual cluster (there are an infinite number of empty ones so they can't each have a non-zero probability without the total probability exceeding 1). However, we can compute the probability that it doesn't go into a component that currently has members (i.e. it goes into *any* of the empty ones) by computing one minus the total probability of going into the non-empty components. Using k_* to denote the empty clusters

$$P(z_{nk_*} = 1|\alpha, \mathbf{Z}^{-n}) = 1 - \sum_{k=1}^{K} \frac{c_k^{-n}}{\alpha + N - 1} = 1 - \frac{N-1}{\alpha + N - 1} = \frac{\alpha}{\alpha + N - 1}.$$

This expression tells us that, under our prior, a data point is assigned to a component that has members with probability proportional to the number of members and to any one of the empty components with probability proportional to α. Hence, if α is high, we will see lots of small components (many data points will go to currently empty components), and, if α is low, we will see a small number of very large components. We can see this in action by just doing Gibbs sampling with this prior (i.e. ignoring the original likelihood term $p(\mathbf{x}_n|\boldsymbol{\mu}_k)$ or the collapsed likelihood term $p(\mathbf{x}_n|\mathbf{Z}^{-n}, \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$, neither of which we actually know how to compute yet for empty components). We can start by assigning all data points to one component and then resample the assignment of each data point with probability

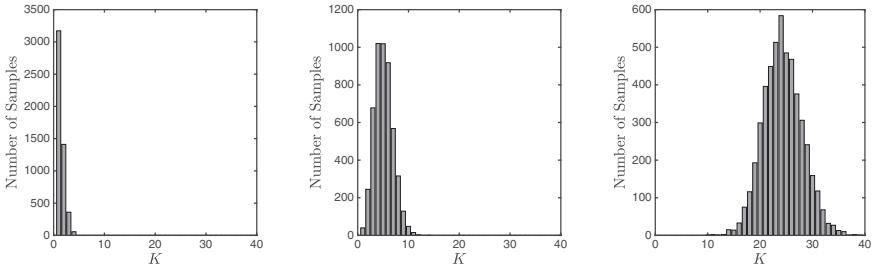
$$P(z_{nk} = 1|\mathbf{Z}^{-n}, \alpha) = \begin{cases} \frac{c_k^{-n}}{\alpha+N-1} & \text{for } c_k^{-n} > 0 \\ \frac{\alpha}{\alpha+N-1} & \text{for empty components} \end{cases}$$

We need a bit of extra bookkeeping. If we assign a data point to the empty components, we need to create a new component to put it in. If we are resampling a data point that is currently the only member of a component, when we remove it from the component to resample it, we must delete the now empty component. Although officially the number of components, K, is equal to infinity, in practice we use K to denote the number of *non-empty* components. Sampling from this prior therefore involves the following steps:

1. Randomly assign the N data points into K components (e.g. put them all in one component; $K = 1$).
2. For each data point $n = 1 \ldots N$:
   (a) Remove point n from its current component.
   (b) If its current component is now empty, delete it (and set $K = K - 1$).
   (c) Compute the probabilities of all non-empty components and the combined probability of empty components.
   (d) Sample a new component k.
   (e) If the component is non-empty, set $z_{nk} = 1$.
   (f) If it is empty, create a new component (set $K = K + 1$) and assign the nth point to it.
3. Return to 2 until enough samples have been drawn.

Both the assignment of data points and the number of non-empty components can change throughout this process. We have therefore moved from a prior that defines how things group together within a fixed number of components to a prior distribution over all possible ways we can partition the N data points into any number of components (in reality, we are limited to between 1 and N components – we cannot partition N objects into $> N$ components!).

This distribution is controlled by a single parameter: α. Figure 10.5 (MATLAB script: `infprior.m`) shows the number of non-empty components when we sample from this conditional prior for three different values of α (0.1,1,10). In each case, the sampler was initialised with all data points in a single component and then run for 5000 samples (i.e. the assignment of each data point was resampled 5000 times). As the parameter is increased (from left to right in Figure 10.5) we see more and more non-empty components. In this example, $N = 100$, so, when $\alpha = 10$, the probability



(a) $\alpha = 0.1$ (b) $\alpha = 1$ (c) $\alpha = 10$

FIGURE 10.5 Number of non-empty components at each sample for three different values of α. Five thousand samples were drawn in each case.

of a data point starting a new component is equal to $10/(10 + 100 - 1) = 0.09$ (new components are created approximately 10% of the time). When $\alpha = 0.1$, this probability drops to 0.001 (new components 0.1% of the time). It is clear why we see more components when α is increased!

Sampling partitions from the prior is one thing, but what we are really interested in is fitting our mixture model, and for that we need to include the data. Before we look at how that is done, we will describe a common analogy for this sampling scheme – the Chinese restaurant process.

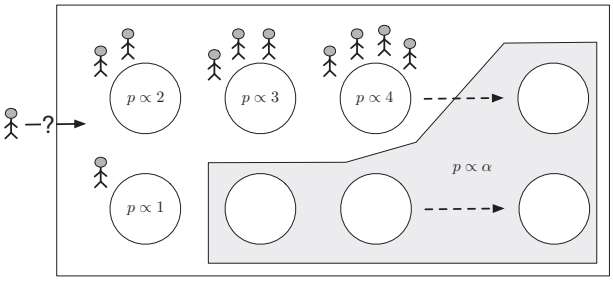


FIGURE 10.6 A cartoon depiction of the Chinese restaurant process. A new diner sits at a non-empty table with probability proportional to the number of diners and sits at a new table with probability proportonal to α.

10.3.1 The Chinese restaurant process

Imagine an enormous restaurant that has room for an infinite number of tables (!). Diners are sitting at some of the tables. Some diners are eating alone, some in groups. As you enter the restaurant, you need to decide where to sit. The **Chinese**

restaurant process (CRP) is a stochastic process that defines the probabilities of you sitting at a particular table. We will use k to index the tables, and assume that the first K tables are occupied. c_k is the number of diners at table k, and the total number of diners in the restaurant is $\sum_{k=1}^{K} c_k = N$. The CRP defines the probabilities as

$$P(k) \propto \begin{cases} c_k & \text{a currently occupied table} \\ \alpha & \text{a currently empty table} \end{cases} \tag{10.9}$$

Normalising the probabilities is straightforward – we just divide each one by the total probability (you have to sit somewhere: $\alpha + \sum_{k=1}^{K} c_k = \alpha + N$). A cartoon depiction of the CRP can be seen in Figure 10.6 (see Exercise 10.6).

We can see a clear correspondence between the CRP and the sampling scheme we derived in the previous section (by marginalising $\boldsymbol{\pi}$ from our model). In fact, our prior is a CRP where we assume that the data point that we are resampling is always the last diner to arrive in the restaurant, and everyone else is already sitting (i.e. we condition on $\mathbf{Z}^{-n}$). In our previous expressions we have $N-1$ diners sitting (because we have removed one) rather than N in this description of the CRP.

How can we explain α within the dining metaphor? Well, high α values correspond to populations of people who are slightly anti-social – they like starting new tables and are likely to end up dining alone. On the other hand, low α values make it likely that diners will be attracted to tables that already have many people at them – they are very social diners. We can see how this relates very clearly to the numbers of non-empty components shown in Figure 10.5.

We have included a description of the CRP for completeness – many machine learning papers that make use of infinite mixture models describe it, and it can often be a useful metaphor for describing extensions to the model, for example, the Hierarchical Dirichlet process (see Section 10.4.1).

10.3.2 Inference in the infinite mixture model

In Section 10.3 we saw how we could extend our model to have an infinite number of components and in doing so remove the need to fix the number of components. The result was a prior distribution for each z_{nk} conditioned on all of the other assignments. However, our expression is just a prior – it assigns probabilities to partitions of the N data points, but doesn't use the actual data to do so. In order to do inference in the new model, we must incorporate the data.

For non-empty components, this is no difference from the likelihood for the finite model. We can either use $p(\mathbf{x}_n|\boldsymbol{\mu}_k)$ or collapse the $\boldsymbol{\mu}_k$ and use $p(\mathbf{x}_n|\mathbf{Z}^{-n}, \mathbf{X}^{-n}, \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$. For empty components, things are not so straightforward – what is $\boldsymbol{\mu}_k$ for a component that doesn't have any data points assigned to it? If we were to update the value of $\boldsymbol{\mu}_k$ for all of the empty tables at each stage of the sampler, we would sample the values from the prior ($\mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$), as none of them have any data assigned to them. So we could (if the fact that there were an infinite number of them wasn't a problem) cycle through each one at each iteration and sample a new value from the prior. Given a value for $\boldsymbol{\mu}_k$ for each empty table, we could compute the likelihood.

However, we don't assign data points to *particular* empty tables – due to their infinite number, each one has probability zero. Instead we assign points to the set

of empty tables. Therefore, we need to marginalise over all possible assignments to empty components. This sounds difficult, as there are an infinite number of these components, but, in fact, its quite straightforward. Recall that we can use samples to approximate expectations. For example, for S samples from $p(z)$, $z_1, \ldots, z_S$, we can approximate arbitrary expectations via

$$\int f(z)p(z)\ dz \approx \frac{1}{S}\sum_{s=1}^{S} f(z_s).$$

The more samples we draw, the better the approximation becomes, and, in the limit that we have an infinite number of samples, the expectation becomes exact. In our impossible sampling scheme, we have an infinite number of samples (one for each empty component) so averaging over them must be the same as computing the expectation

$$p(\mathbf{x}_n|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) = \int p(\mathbf{x}_n|\boldsymbol{\mu})p(\boldsymbol{\mu}|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)\ d\boldsymbol{\mu},$$

which, in our example, is possible because $p(\mathbf{x}_n|\boldsymbol{\mu})$ is a Gaussian with mean $\boldsymbol{\mu}$ and identity covariance. The result is given by

$$p(\mathbf{x}_n|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0 + \mathbf{I}). \tag{10.10}$$

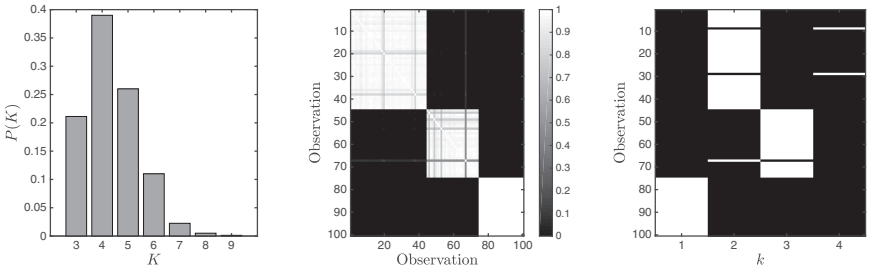
In general, we may not be able to compute this expectation, but all is not lost – we can approximate it with any number of samples from the prior that we like. The more samples we draw the better, but even approximating the expectation by drawing one sample has been shown to work fairly well.

Our infinite collapsed Gibbs sampling scheme now involves resampling each $\mathbf{z}_n$ according to the probabilities:

$$P(z_{nk} = 1|\ldots) \propto \begin{cases} c_k^{-n} p(\mathbf{x}_n|\mathbf{Z}^{-n}, \mathbf{X}^{-n}, \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) & \text{non-empty component} \\ \alpha p(\mathbf{x}_n|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) & \text{empty component} \end{cases} \tag{10.11}$$

where we have relegated the denominator of the prior to the overall normalisation constant, and $p(\mathbf{x}_n|\mathbf{Z}^{-n}, \mathbf{X}^{-n}, \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$ and $p(\mathbf{x}_n|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$ are given by Equations 10.8 and 10.10, respectively.

We will illustrate this infinite sampling scheme with the same data we used for the finite mixture model at the start of this chapter (MATLAB script: `gibbsinf.m`). The data are shown in Figure 10.1, and, as before, we set $\boldsymbol{\mu}_0$ to a vector of zeros and $\boldsymbol{\Sigma}_0$ to the identity matrix. α was set to 1. As for the finite model, 1000 samples were taken, of which the first 200 were discarded as a burn-in phase. For the finite model, we plotted the posterior samples for $\boldsymbol{\mu}_k$ and $\boldsymbol{\pi}$ as well as the posterior component membership. For the infinite model we cannot plot any of these things – we no longer sample $\boldsymbol{\mu}_k$ and $\boldsymbol{\pi}$, and, as components are created and destroyed, there isn't a static set of components to compute posterior probabilities for. Interpreting the output of the infinite model is more challenging than for the finite model, and how we do it will often depend on what we are using the mixture model for. Two things that are often visualised are the marginal posterior over the number of non-empty components and the pairwise probabilities that two data points are in the same component. These



(a) Marginal posterior over the number of non-empty components.

(b) Marginal posterior over pairwise component membership.

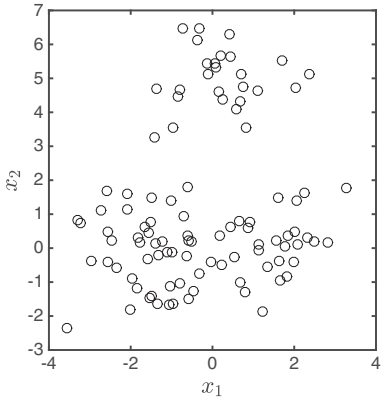
(c) One randomly chosen posterior sample of component membership.

FIGURE 10.7 Output of the infinite mixture model for the data shown in Figure 10.1.

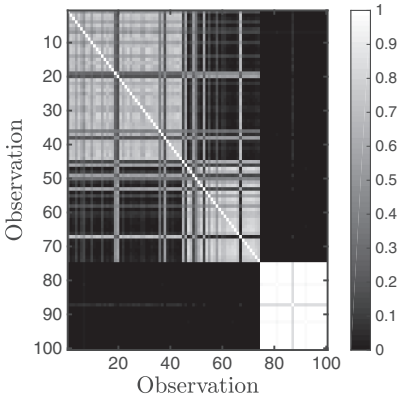
are shown in Figure 10.7. The marginal distribution over the number of non-empty components K is shown in Figure 10.7(a). To obtain this distribution we keep track of the number of non-empty components after each iteration of the sampler. The posterior seems to overestimate the number of components (there should be three). If we look at a randomly chosen set of assignments from the sampler (Figure 10.7(c)) we can see why. Whenever we resample the membership of a data point, there is a finite probability that it will form a new component. This component might attract one or two more members (see $k = 4$ in Figure 10.7(c)) but will most likely collapse again within a couple of iterations. So, there are always data points jumping out and forming new components that don't survive and hence we overestimate the number of components. One way of interpreting Figure 10.7(a) is that it needs *at least* $K = 3$ components (we never see $K = 2$), which seems reasonable.

Figure 10.7(b) visualises the pairwise membership probabilities. The posterior probability is indicated by the colour at a particular row and column. For example, the strong white colour in the upper left block indicates that the first 40 or so points almost always end up in the same component. We obtain these probabilities by counting how often each pair of data points was assigned to the same component (regardless of which component it was). The data points (observations) are ordered according to their true component, and we can clearly see the block structure that we would expect if the clustering was correct. It's not perfect though – there are some data points that don't always get assigned to the correct component. And, as we have already seen, each data point will occasionally jump off on its own.

This data is a bit easy to model, as the three groups of data are highly separated. In Figure 10.8(a) we show a tougher dataset where the groups (particularly the lower two) are significantly closer together. Figure 10.8(b) shows quite clearly the additional posterior uncertainty in this problem – although the majority of the first two components make up tidy blocks, there are a lot of data points that switch between the two. The final component is still quite separated (it is the top one in Figure 10.8(a)) and the posterior probabilities still form a nice clean block.

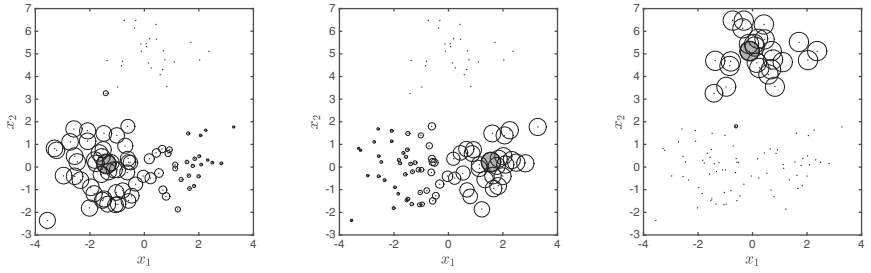


(a) A tricker dataset.



(b) Pairwise posterior membership probabilities.

FIGURE 10.8 The performance of the infinite mixture on a slightly tougher dataset.



(a) Component 1 (b) Component 2 (c) Component 3

FIGURE 10.9 Visualising the pairwise probabilities between the point closest to the center of each component (shaded) and the other points. Circle area corresponds to probability, with the areas of the shaded points being 1 (points are always in the same component as themselves).

Finally, in Figure 10.9 we visualise the pairwise posterior probabilities in a different way. Here, for each of the three components, we show the pairwise probability between each data point and the data point closest to the true centre of that component (i.e. the centre used to generate the data). The circle size (area) corresponds to the probability. The shaded point is the one being compared to and therefore has a probability of 1 (it is always in the same component as itself). Focusing on components 1 and 2 (component 3 is far enough away that points rarely switch), we can see exactly what we might expect. The probability of being in the same mem-

bership as the central point in component 1 drops as we move towards component 2 – points in the middle flip between the components more often. None of the circles is as large as the shaded one, as all points will occasionally jump out into their own components.

The steps we have taken are not restricted to Gaussian densities. To try an example with a different prior and likelihood, see Exercise 10.7.

10.3.3 Summary

In this section we have introduced the idea of an *infinite mixture model* – a model that has an infinite number of components, only some of which will ever be occupied. This has removed the need to specify a priori how many components there are – the model fills up as many as it needs. This is a useful and computationally straightforward model that has found many uses within machine learning. The prior degree of clustering expected is controlled by α. High values of α lead to a preference towards many small clusters, low values to a smaller number of large clusters.

One drawback of this approach is interpretation of the output – it is hard to summarise the thousands of samples from the posterior because the number of components in each sample can be different. However, this problem can be overcome if what is ultimately of interest are marginal distributions (e.g. the number of components, pairwise probabilities, etc.), or expectations that can be computed from the samples (e.g. anything that involves a real-valued function that takes a clustering as its input).

We obtained this model via increasing K to infinity. In the following section we will see that this is in fact equivalent to a more general stochastic process – the Dirichlet process.

10.4 DIRICHLET PROCESSES

In this section we will provide a brief description of the Dirichlet process. Dirichlet processes are stochastic processes from which we can sample sequences of values that have interesting properties. To go into DPs in detail would require a deeper understanding of probability theory than we have provided, so we'll keep our description quite informal. At the end of the chapter are pointers to further reading for those wanting to explore more.

Let's assume we are interested in sampling continuous values θ. To define a DP, we will have to choose a *base* distribution for θ (more on this later). To stick with the most common DP notation, we will use H to represent this distribution. More specifically, because θ is a continuous random variable, it will have pdf $h(\theta)$. Recall that the probability that θ lies in some subset of possible values (e.g. the probability that θ is between 1 and 2) is obtained by integrating the pdf over that subset. In general, we will use A to denote subsets of possible θ values and use $H(A)$ to denote the probability that θ lies in the subset defined by A. For example, perhaps A is the subset of values between some values a and b:

$$H(A) = P(\theta \in A) = P(a < \theta < b) = \int_a^b h(\theta)\, d\theta.$$

As well as thinking of $H(A)$ as the integral of the pdf over A, we can also think of

it more generally as a function that takes a subset of θ values and assigns a probability. You might find this a more useful approach when thinking about DPs. This touches on **measure theory** – a subset of mathematics that formalises probability. We won't go into any details here, but a couple of good introductory books are mentioned at the end of the chapter.

So far, this description has been very abstract. Don't worry – things will get more concrete later on! Now, consider a set of M subsets, such that all possible values of θ are in one and only one subset (a **partition**). If the mth subset is denoted A_m, the following must be true:

$$\sum_{m=1}^{M} H(A_m) = 1$$

because the subsets cover all possible values of θ and don't overlap.

Although it defines probabilites, H itself is deterministic – it always gives the same probability for some subset A. Would it be possible to produce some kind of function that acts on subsets of θ values and produces probabilities just as H does, but is itself stochastic and produces random probabilities? The answer is yes, and a DP is exactly this! In particular, consider a new function $G(A)$. G is a Dirichlet process if the vector of values $[G(A_1), \ldots, G(A_M)]$ is distributed according to the Dirichlet distribution (see Comment 10.1) with parameters $\alpha H(A_1), \ldots, \alpha H(A_M)$ where α is a positive constant.

Formally, G is a DP if:

$$[G(A_1), \ldots, G(A_M)] \sim Dir(\alpha H(A_1), \ldots, \alpha H(A_M)). \tag{10.12}$$

This is quite an abstract definition. For some partition $A_1, \ldots, A_M$, we could sample vectors $[G(A_1), \ldots, G(A_M)]$ from the Dirichlet described above, but what then? Insight into the DP is obtained if we generate samples from it but it's not obvious how to do that from this expression. Fortunately, we can make some progress by marginalising G.

Consider two random variables x, y and the two conditional distributions $p(x|y)$ and $p(y|z)$. Assuming z is known, one way to sample values of x is to sample a value of y from $p(y|z)$ and then sample x from $p(x|y)$. Another way would be to marginalise y and just sample a value of x directly conditioned on z:

$$p(x|z) = \int p(x|y)p(y|z)\ dy.$$

As far as the samples of x are concerned, the two processes are statistically identical. We can do the same for our DP to marginalise the GP. In particular

$$\begin{aligned} P(\theta \in A_m|H) &= \int P(\theta \in A_m|G)P(G|H)\ dG \\ &= \int G(A_m)P(G(A_m|H)\ dG(A_m) = \mathbf{E}\left\{G(A_m)\right\}, \end{aligned}$$

which is the expected value of $G(A_m)$. As $G(A_m)$ is the mth component of a Dirichlet, its expected value is the corresponding Dirichlet parameter divided by the sum of the Dirichlet parameters:

$$\mathbf{E}\left\{G(A_m)\right\} = \frac{\alpha H(A_m)}{\sum_{j=1}^{M} \alpha H(A_j)} = \frac{\alpha H(A_m)}{\alpha \sum_{j=1}^{M} H(A_j)} = H(A_m).$$

So, the probability of θ being in A_m is $H(A_m)$, which is just the base probability. We can therefore easily draw a single sample of θ from the DP: we just draw a sample from H. In other words, the pdf $g(\theta)$ is identical to the pdf $h(\theta)$.

This doesn't seem very interesting, but it gets more interesting when we consider multiple samples from a DP. A DP actually gives us a sequence of samples, each of which is sampled conditioned on the previous ones (it is a stochastic process): $g(\theta_1)$, $g(\theta_2|\theta_1)$, $g(\theta_3|\theta_1, \theta_2)$, etc. We will see some interesting properties of these samples below, but to give you a sneak preview, consider a Gaussian base distribution. The probability that any two values drawn directly from the base are identical is zero – we would never draw exactly the same value twice. In contrast, the probability that two (or more) identical values are drawn from a DP with this same Gaussian as its base is non-zero!

To see why, imagine the situation where we have observed N observations, $\theta_1, \ldots, \theta_N$. For some partition $A_1, \ldots, A_M$, our DP says that

$$[G(A_1), \ldots, G(A_M)] \sim Dir(\alpha H(A_1), \ldots, H(A_M)).$$

We can think of this as a prior over the probabilities of θ lying in each of the m regions. Having observed some data, we ought to update these probabilities – regions in which we have observed lots of θ values ought to get more likely, whilst regions in which we have observed very few ought to become less likely. If we use c_m to denote the number of samples that lie in A_m, we can therefore use a multinomial likelihood to update the Dirichlet. We have already seen that in a Dirichlet-multinomial model, the posterior Dirichlet has parameters equal to the prior parameters plus the associated counts. The posterior is therefore

$$[G(A_1), \ldots, G(A_M)]|\theta_1, \ldots, \theta_N \sim Dir(\alpha H(A_1) + c_1, \ldots, H(A_M) + c_M).$$

So, having observed some data, the vector $[G(A_1), \ldots, G(A_M)]$ is distributed according to a Dirichlet. If we can write the parameters of the Dirichlet as the product of a base (say H') and a parameter (α'), then this posterior must also be a DP. We can find H' and α' with a little algebra:

$$\begin{aligned} \alpha' H'(A_m) &= \alpha H(A_m) + c_m \\ \sum_m \alpha' H'(A_m) &= \sum_m \alpha H(A_m) + c_m \\ \alpha' &= \alpha + N \\ \text{and therefore} \quad H'(A_m) &= \frac{\alpha H(A_m) + c_m}{\alpha + N}. \end{aligned}$$

It will be useful to write this slightly more generally and replace c_m with $\sum_{n=1}^{N} \delta_{\theta_n}(A_m)$ where $\delta_{\theta_n}(A_m)$ is 1 if $\theta_n \in A_m$ and 0 otherwise. Having observed some data, we therefore have a new DP with parameter $\alpha' = \alpha + N$ and base

$$H'(A) = \frac{\alpha H(A) + \sum_{n=1}^{N} \delta_{\theta_n}(A)}{\alpha + N}.$$

We already know how to take one sample from a DP – just generate a sample from the base. What would a sample from this new base look like? It helps to rewrite it:

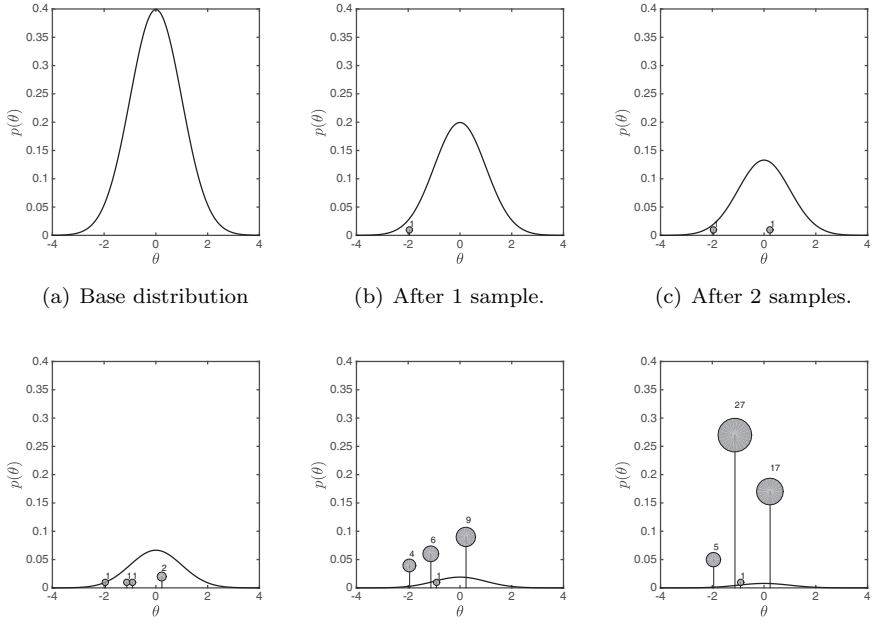
$$H'(A) = \frac{\alpha}{\alpha + N} H(A) + \frac{1}{\alpha + N} \sum_{n=1}^{N} \delta_{\theta_n}(A).$$

This is a mixture where the first component is the original base and the second component is some function of the data that has been observed. The mixture weights are given by $\alpha/(\alpha + N)$ for the original base and $N/(\alpha + N)$ for all of the previous observations combined. We know how to sample from a mixture – we first choose a component based on the weights and then sample a value from the chosen component. If we choose the first component, we draw a sample from the original base. The second component is a bit more complex. In fact, as it is a summation, we can think of it as another mixture. Each component has equal weight and corresponds to one of the previous observations. Each of these components has the form

$$Q(A) = \delta_{\theta_n}(A),$$

i.e. the probability of the sample being in A is 1 if θ_n is in A and zero otherwise. How can we sample from Q? Well, all samples from Q must belong to any range A that includes θ_n. The only value of θ that can satisfy this is θ_n itself. So, a sample from this component has to be identical to θ_n.

So, to sample from the posterior DP we first decide whether or not to use the original base distribution with probability $\alpha/(\alpha+N)$. If we do, we generate a sample from the base. If we do not, we pick one of the previous samples (uniformly) and replicate it. Hence the non-zero probability of obtaining identical samples.



(a) Base distribution (b) After 1 sample. (c) After 2 samples.

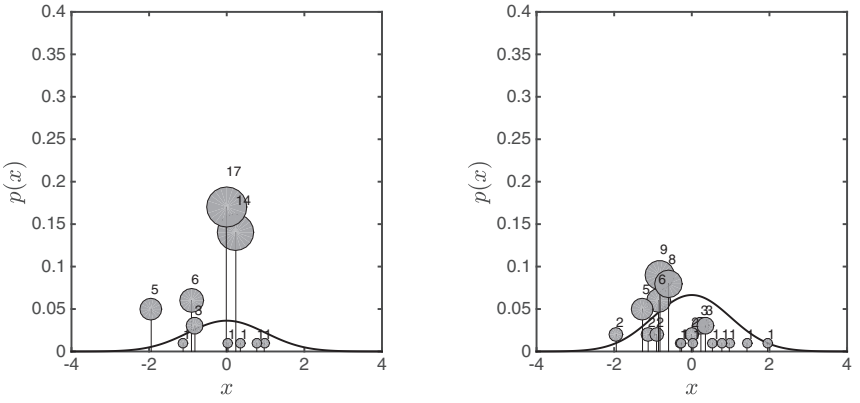
(d) After 5 samples. (e) After 20 samples. (f) After 50 samples.

FIGURE 10.10 Samples from a DP with a Gaussian base distribution $(\mu = 0, \sigma^2 = 1)$ and $\alpha = 1$.

Let's consider a more concrete example. Assume that H is a Gaussian with zero

mean and unit standard deviation. The first sample, θ_1, is simply a sample from this Gaussian. Assuming $\alpha = 1$ (more about this parameter later) when sampling the second sample, we have $N = 1$ previous samples, and therefore sample from H again with probability $1/(1+1) = 0.5$ and otherwise copy θ_1. This process then repeats for subsequent samples. The fact that, when sampling sets of values from a DP, there is a non-zero probability of many of them being identical is a key property of the DP. It is exactly this property that makes them very useful in mixture models. Before we see the connection, let's draw some samples from a real DP.

Figure 10.10 shows this sampling process for a DP with a Gaussian base distribution ($\mu = 0, \sigma^2 = 1$) and $\alpha = 1$ (MATLAB script: `dpsamp.m`). The first plot shows the base distribution from which the first sample will be drawn. This sample is shown in the second plot, along with the base distribution that we've reweighted by $\alpha/(\alpha + N)$ to show that for subsequent samples it is less likely that we will draw from it. The second sample will either be drawn from the base (with probability $1/(1 + 1) = 0.5$) or be equal to the first sample (also with probability 0.5). In this example, the base won and the new sample (with the base reweighted further) can be seen in the third plot. The state after five samples can be seen in the fourth plot. Four unique values have been drawn, one of which has been drawn twice (shown by a slightly larger circle, plotted slightly higher). After five samples, the probability of the sixth sample coming from the base is $1/(1+5) = 1/6$. This is lower than the value of 0.5 (after the first sample) and will keep decreasing as we generate more samples. In other words, the more samples we draw, the less likely it is that we will draw an original one, and the more likely that we will copy an old one (our re-weighting of the base in the figures is to demonstrate this). In the final two panels, we see the state of the DP after 20 and 50 samples. In the latter, we can clearly see that a small number of values are dominating. In fact, no new values have been sampled from the base – we have the same four values that we had after five samples (see Exercise 10.8).



(a) $\alpha = 5$.

(b) $\alpha = 10$.

FIGURE 10.11 Fifty samples from a DP with $\alpha = 5$ and $\alpha = 10$.

Let's now look at what happens if we increase α. We know that, after N samples, the probability of generating a new value is equal to $\alpha/(\alpha + N)$, suggesting that increasing α will result in more new values and therefore more unique values after, say, 50 samples. Figure 10.11 shows the first 50 samples for $\alpha = 5$ and $\alpha = 10$ and we can see clearly the increase in unique values as α gets larger. Note also that the base is larger in these two plots than it was after 50 samples when $\alpha = 1$ (see Figure 10.10). This is because, when $\alpha = 10$, the probability of a new value after 50 samples is still quite large $(10/(10 + 50) = 1/6$, for $\alpha = 1$ the corresponding figure is $1/51)$. In the DP world, α is known as the **concentration parameter**.

Hopefully you are beginning to see the parallels between this process and the infinite mixture model described previously. To help, we can rewrite our posterior base distribution a little. The posterior base is given by

$$H'(A) = \frac{\alpha}{\alpha + N} H(A) + \frac{1}{\alpha + N} \sum_{n=1}^{N} \delta_{\theta_n}(A).$$

Within the N previous samples there will be K unique values $(K \leq N)$. Rather than pick from the previous samples uniformly, we can instead pick from the previous unique values with probability proportional to the number of times they have been sampled. Using n_k to denote the number of times unique value θ_k has been sampled, we have

$$H'(A) = \frac{\alpha}{\alpha + N} H(A) + \frac{1}{\alpha + N} \sum_{k=1}^{K} n_k \delta_{\theta_k}(A).$$

To sample from this, we first choose whether or not we should sample a new value, with probability proportional to α, and, if not, replicate a previous value with probability proportional to the number of times it has been sampled. If we replace the word value with component, this is exactly the prior we obtained for an infinite mixture model in Section 10.3! The value we are sampling in the DP is equivalent to the component mean (μ_k) in the infinite mixture model.

There is one distinction: In the infinite mixture, at any point in the sampler, we considered only K mean values. In the DP, every data point has its own mean value – we always sample a θ, but amongst the N values, there are only K unique ones. This distinction has no practical implications, but it is an important feature of the DP.

When used in mixture models, the DP is a prior on the component parameters for each observation. It gives a non-zero probability that multiple observations will have identical parameters and therefore belong to the same component. To complete the mixture model, we just need to add the likelihood term that links the component parameters with the observed data (e.g. a Gaussian). We can therefore generate data by first (for each data point) sampling component parameters from the DP and then sampling data conditioned on these components. Two example datasets drawn from such a model with a Gaussian base, $\alpha = 1$ and an isotropic Gaussian likelihood (with $\sigma^2 = 1$) can be seen in Figure 10.12 (MATLAB script: `dpsampdata.m`). The mean values from the DP prior are shown as filled circles; the data are the open circles and the contours show the base distribution. Unsurprisingly, these datasests have clear cluster structure. If we were to derive Gibbs updates for this model to perform inference, they would be identical to those derived for the infinite mixture in Section 10.3, so we won't repeat them here.

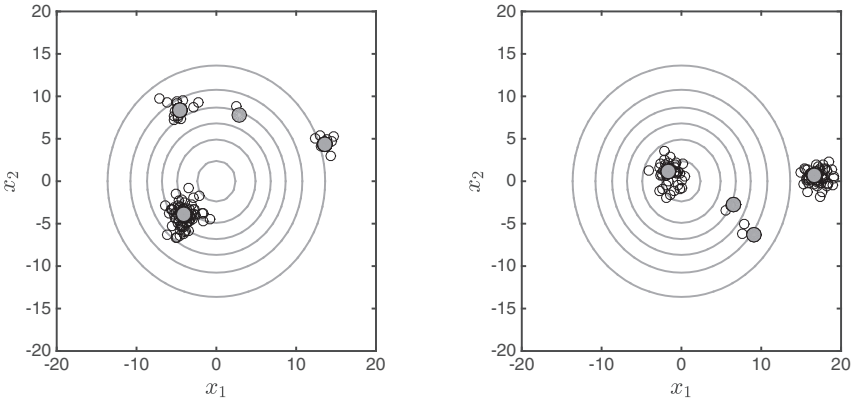


FIGURE 10.12 Two datasets drawn from a model with a DP prior with a Gaussian base (contours shown) and an isotropic Gaussian likelihood ($\sigma^2 = 1$).

10.4.1 Hierarchical Dirichlet processes

The standard clustering problem assumes that we have one dataset and would like to group the data into a set of components. In some applications we might have more than one dataset to cluster. We could cluster them separately, but it might be interesting to know if some of the clusters exist across multiple datasets. Hierarchical Dirichlet processes (HDPs) were created for exactly this problem. A separate DP prior is used for the data points in each file where the base distribution is itself a DP that is shared across all of the files. Formally, we can describe this as follows:

$$\begin{aligned} G_0 &\sim DP(\gamma, H) \\ G_1 &\sim DP(\alpha_1, G_0) \\ &\vdots \\ G_J &\sim DP(\alpha_J, G_0) \end{aligned}$$

where there are $j = 1 \ldots J$ files.

At first glance this looks a bit daunting but we can gain some insight by imagining drawing samples. We will denote the nth sample in the jth file as θ_{jn}. For the first sample in the first file, we need a sample from G_1. No other samples have been drawn so this is just a sample from the base, G_0. No samples have been drawn from G_0 either, so we have to generate a new sample from its base, H. We will use β_k to denote the kth unique sample from G_0 and n_k to denote the number of times it has been sampled. So, for the first sample, we sample β_1 from the base and set n_1 to 1. Within file 1, we need to keep track of the number of different values we sample from G_0. We will denote the ith new value with δ_{ji}, so in this case, $\delta_{11} = \beta_1$ and $\theta_{11} = \delta_{11}$. To keep track of the number of objects assigned to δ_{ji}, we will use m_{ji}.

Therefore, $m_{11} = 1$. Now consider the second sample in file 1, θ_{12}. It will either be equal to δ_{11} (and therefore equal to θ_{11}) or we must sample a new value δ_{12} from G_0. If it is the former, G_0 is unchanged (G_0 only changes when we sample from it) and we set $\theta_{12} = \delta_{11}$ and $m_{11} = 2$. If the latter, we generate a sample from G_0, set δ_{12} to equal this value and set $m_{12} = 1$. This new sample could either be identical to β_1 or will be a new sample from H. Note that, if it is identical to β_1, then δ_{21} is also identical to δ_{11}! However, as these are separate draws from G_0, we must treat them as separate objects. This will be more obvious when we relate this process back to the Chinese restaurant (below).

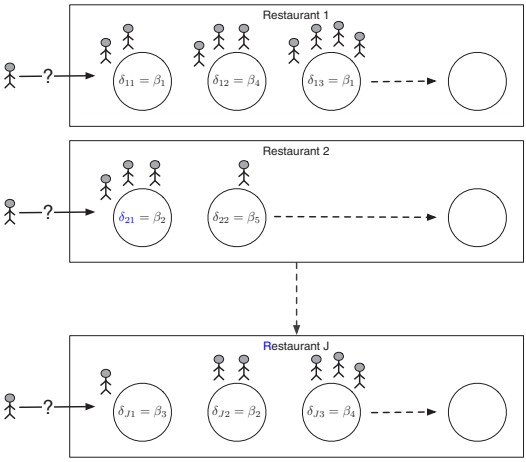


FIGURE 10.13 The Chinese restaurant franchise.

After two samples from file 1, we now consider the first sample from file 2. So far, we have no samples from G_2 so we must generate a new sample (δ_{21}) from G_0. This time there have already been samples from G_0, so δ_{21} could be, say, identical to β_1. Let's assume it is, so $\delta_{21} = \beta_1$ and $\theta_{21} = \delta_{21}$. Now imagine that we are also generating data, x_{jn}, which comes from a Gaussian with mean θ_{jn} and unit variance. x_{21} will have mean $\theta_{21} = \delta_{21} = \beta_1 = \delta_{11} = \theta_{11}$. In other words, x_{21} will share a Gaussian mean with x_{11}. With a small leap of faith, hopefully you can see that, when it comes to inference, this means that mixture components are effectively shared across the different files.

As we have just demonstrated, this is quite a tricky process to describe in words. Fortunately, the Chinese restaurant metaphor (see Figure 10.6) has been extended to this case resulting in the Chinese restaurant franchise. Each file is one branch of a chain of restaurants that all share the same menu. At each table, diners all eat the same item from this menu. This is depicted in Figure 10.13. Each of the diners has an associated θ_{jn} value which is equal to the δ_{ji} value of the table at which they are sitting. Each δ_{ji} value is identical to one of the dishes, β_k. In this example there are five unique dishes, $\beta_1, \ldots, \beta_5$.

As with the standard Chinese restaurant process, a diner sits at a table with probability proportional to the number of diners at the table, or at a new table with

probability $\alpha_j/(\alpha_j + N_j)$ (where N_j is the number of diners in restaurant j). If they sit at a current table, their value of θ is set to the table's δ value. If a new table, we draw a δ from G_0. A sample from G_0 will be a dish (β) that already exists or a new dish. New dishes are generated with probability proportional to γ, whilst old dishes are resampled with probability proportional to the number of tables across the franchise that are using that dish. We can see now why it is important to keep track of all draws from G_0 within a restaurant (through the δs), even if the draw is identical to a β in use on another table in the same restaurant – it is still a new draw from G_0.

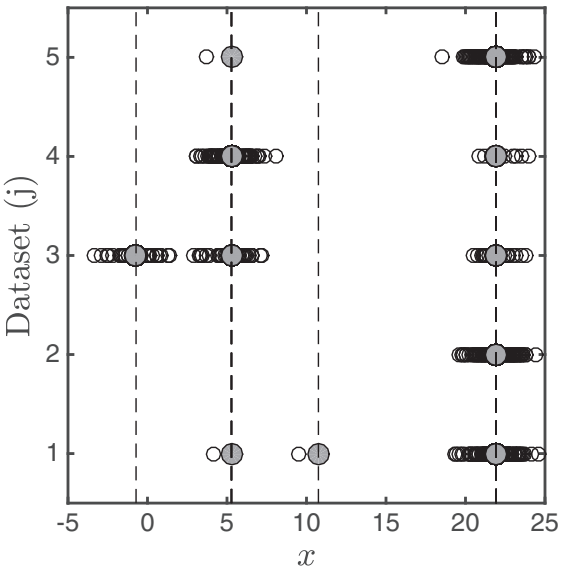


FIGURE 10.14 One hundred samples for each of five datasets. H is a Gaussian with mean 0 and $\sigma^2 = 100$. All concentration parameters are set to 1 ($\gamma = \alpha_1 = \ldots = \alpha_J = 1$).

An example of five one-dimensional datasets drawn from an HDP can be seen in Figure 10.14 (MATLAB script: `hdp.m`). In this example, the overall base was set to a Gaussian with $\mu = 0$ and $\sigma^2 = 100$. All concentration parameters $(\gamma, \alpha_1, \ldots, \alpha_J)$ were set to 1. The data are one dimensional, and each dataset is shown at a different height on the y-axis. The filled circles show the cluster means, and the dashed lines the unique values of β. We can clearly see the sharing of clusters between datasets. The highest (rightmost) value of β is shared by all five datasets and all other values appear in one or more datasets. Changing the concentration parameters controls how much sharing goes on across the datasets. For example, if we set γ to a high value, then most samples from G_0 will be from the base, and we will get very few shared β values. Increasing individual α values changes how likely it is that a new draw from G_0 will be required in each dataset. Different datasets need not necessarily be

given the same α_j values. We leave it as a practical exercise to experiment with the setting of these values (see Exercise 10.9).

We will not go into detail on the inference for HDPs here. Gibbs sampling is fairly straightforward (with some slightly more complex bookkeeping than the standard DP) and we leave it as an exercise (see Exercise 10.10). Readers are referred to the references at the end of the chapter for more information.

10.4.2 Summary

In the last two sections we have provided a quick, informal introduction to Dirichlet processes and Hierarchical Dirichlet processes. To go any deeper would require the reader to have some familiarity with measure theory, and that is a whole book in itself (see the references at the end of this chapter). Within the context of mixture models, the DP can be used as a prior distribution over the component parameters associated with each data point. Crucially, the DP has a non-zero probability that some of these parameters will be identical and hence data points can be thought of as coming from the same component. Hierarchical Dirichlet processes allow us to perform DP mixture modelling on separate datasets whilst sharing information between them.

Finally, we will look at what happens when we relax one of the key assumptions underpinning mixture models – that each data point is generated by one component.

10.5 BEYOND STANDARD MIXTURES – TOPIC MODELS

In the previous sections we have shown how we can use mixture models without specifying a priori how many components are required. This extension can, in theory, be used for any type of mixture component, although inference might be tricky if the prior and likelihood don't form a conjugate pair. When the pair is conjugate, deriving the updates required for Gibbs sampling is fairly formulaic, so we will not provide details of the updates for other conjugate pairs here. Instead, we will briefly introduce a model that relaxes the key mixture model assumption – that each data point is generated by one particular mixture component.

Consider clustering the text in web pages. We could use a standard mixture model that assumes each web page is generated by one mixture component. In this case, each mixture component would be defined by a multinomial distribution over words, a suitable prior for which is the Dirichlet. Each component could be considered to be a topic, and each page generated by that component is about that topic.

In reality, however, this definition is too restrictive. Many pages would be about more than one topic. For example, if we allowed humans to tag pages with topics, then a page about climate change might be tagged with two topics: *environment* and *politics*. A page about football transfers might be tagged with *football* and *economics*. Trying to model these documents with a mixture will result in many very specific mixture components that will not capture topics as humans might understand them.

Latent Dirichlet Allocation (LDA) is a probabilistic model that overcomes this limitation. It has proven very popular in many machine learning applications (including text modelling and biological data – see the references at the end of the chapter). Like a standard mixture model, LDA consists of K components (it can be made infinite, but we will stick to the finite version here). In the text example,

each component is a multinomial distribution over words. The difference is that, rather than each document being generated by one topic, it is assumed to have been generated by multiple topics. The generative process is as follows:

1. For each document:
   (a) Sample a multinomial over the K topics, $\boldsymbol{\theta}_n$, from a Dirichlet $p(\boldsymbol{\theta}_n|\boldsymbol{\alpha})$.
   (b) For each word in the document:
      i. Sample a topic from the K topics where the kth topic has probability θ_{nk}.
      ii. Sample the word from the chosen topic.

This process reduces to the standard mixture if the distribution over topics ($\boldsymbol{\theta}_n$) for each document has probability 1 for one topic and probability 0 for all others. In other words, the standard mixture model is a special case of this more general model.

Given a set of documents (or other data types), inference in LDA is only a little more complex than for a standard mixture. The extra complexity lies in rather than keeping track of one set of indicator variables for each document, we now need to keep one set for each word in each document. Formally, the model is defined as

$$\begin{aligned} p(\boldsymbol{\theta}_n|\boldsymbol{\alpha}) &= Dir(\boldsymbol{\alpha}) \\ p(z_{ni} = k|\boldsymbol{\theta}_n) &= \theta_{nk} \\ P(x_{ni} = w|z_{ni} = k) &= \beta_{kw} \\ p(\boldsymbol{\beta}_k|\boldsymbol{\gamma}) &= Dir(\boldsymbol{\gamma}) \end{aligned}$$

where i indexes the words in any particular document, $z_{ni} = k$ if the ith word in document n was generated by topic k and $x_{ni} = w$ if the ith word in document n is word w. $\boldsymbol{\beta}_k$ are the word probabilities that define the kth topic and $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ are the hyperparameters defining the Dirichlet priors over the document to topic multinomials and the topic to word multinomials.

Because we are only dealing with Dirichlets and multinomials, we can use collapsed Gibbs sampling to marginalise all of the multinomial parameters, resulting in a Gibbs sampling scheme that requires just updating z_{ni}. In particular, we leave it as an exercise (see Exercise 10.11) to show that the conditional probability that $z_{ni} = k$ is given by

$$P(z_{ni} = k|x_{ni} = w, \ldots) \propto (c_{nk}^{-i} + \alpha_k) \times \frac{v_{kw} + \gamma_w}{\sum_{w'} v_{kw'} + \gamma_{w'}}. \tag{10.13}$$

The first term comes from marginalising $\boldsymbol{\theta}_n$ and the second from marginalising $\boldsymbol{\beta}_k$. c_{nk}^{-i} is the number of other words in document n that are currently assigned to topic k, and v_{kw} is the number of other times word w has been assigned to topic k (across the whole dataset).

We will illustrate this with a simple example (MATLAB script: `lda.m`). We will use a vocabulary of $W = 5$ words and generate data from $K = 3$ topics. The word probabilities for each topic are each sampled from a Dirichlet with uniform parameter $\gamma = 0.1$. These three topics can be seen in Figure 10.15. For each of $N = 100$ documents, $\boldsymbol{\theta}_n$ is sampled from a Dirichlet with uniform parameter $\alpha = 10$ and for each document 50 words are sampled using the process described above.

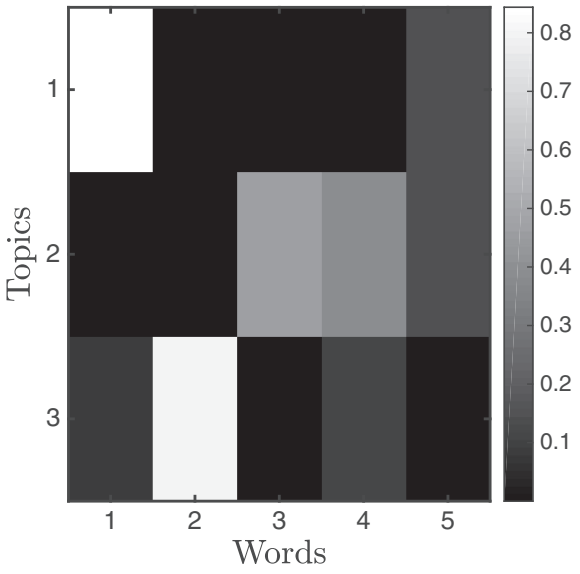


FIGURE 10.15 The true word parameters for the LDA example. Each row is a topic and each column a word.

We then use collapsed Gibbs sampling to sample from the posterior distribution over z_{ni}. Although we don't explicitly sample them, we are jointly inferring the distributions over topics for each document ($\boldsymbol{\theta}_n$) and the distribution over words for each topic ($\boldsymbol{\beta}_k$). At any point in the sampler, we can compute the posterior Dirichlet over each $\boldsymbol{\theta}_n$ and $\boldsymbol{\beta}_k$. In the former case, the distribution is Dirichlet with parameters $\alpha_k + c_{nk}$ and in the latter Dirichlet with parameters $v_{kw} + \gamma_w$ (if you're unsure about this, try Exercise 10.11). Comparing the expected value of $\boldsymbol{\beta}_k$ with the true value allows us to assess how well we are able to learn the true topics. The expected value of the posterior Dirichlet over the topic to word probabilities ($\boldsymbol{\beta}_k$) can be seen in Figure 10.16. Comparison with the true values in Figure 10.15 shows that the probabilities match very well, suggesting that the sampling scheme is indeed able to recreate the true topic probabilities.

LDA has been used successfully in many applications, from discovering scientific topics to modelling cellular processes (see the references at the end of the chapter). Inference in LDA is not restricted to collapsed Gibbs sampling. In fact, in the original paper (see references at the end of the chapter), a Variational Bayes (see Chapter 7) scheme is presented.

10.6 CHAPTER SUMMARY

In this chapter we have described some extensions to the standard mixture model. In particular, how we can remove the need to specify the number of components

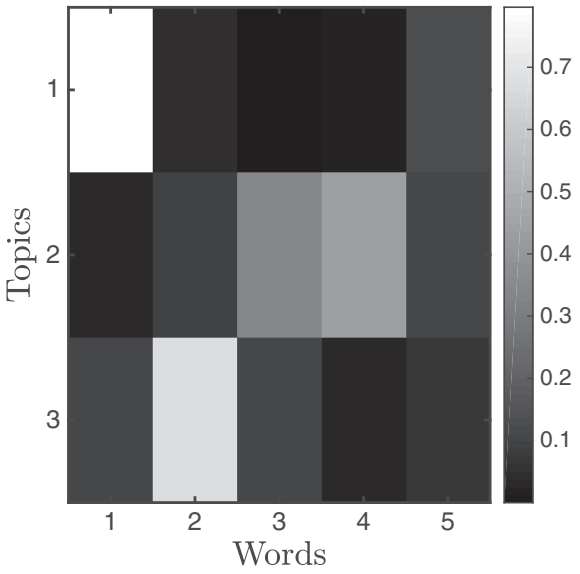


FIGURE 10.16 The inferred word parameters for the LDA example.

and how we can relax the original assumption that each observation comes from one component. Such extensions to standard mixture models have been very popular in machine learning in recent years, with many models being proposed and applied in a wide range of applications.

It was not our aim in this chapter to give a comprehensive survey of machine learning approaches in this area. Rather, we intended to provide an intuitive understanding of these approaches to give the reader the necessary background to explore and understand some of the interesting work in this area. The references presented below are a good starting point!

10.7 EXERCISES

10.1 For the mixture model where each component is a one-dimensional Gaussian with variance 1, show that the conditional distribution for the mean of the kth mixture component is that given by Equation 10.4.

10.2 Implement the Gibbs sampler for the mixture model. Compute $\hat{R}$ and the autocorrelation of the component means μ_k. How many samples are required for convergence? How much should the samples be thinned?

10.3 In Section 6.3.7 we used EM to fit a mixture model for binary data. Assuming a beta prior on the binomial parameters, derive the Gibbs sampler for this model.

10.4 Marginalise the Gaussian mean from the Gibbs sampler to give a conditional

distribution for z_{nk} conditioned only on the other assignments and the prior parameters $(\mu_0, \boldsymbol{\Sigma}_0, \alpha)$.

10.5 Implement the collapsed Gibbs sampler for the Gaussian mixture model. At each iteration in the sampler, sample the means μ_k even though they are not needed (sample them from the posterior you computed to marginalise them). Compare the autocorrelation and $\hat{R}$ for the μ_k with that obtained in Exercise 10.2.

10.6 Implement a sampler that samples from the Chinese restaurant process. Repeatedly resample assignment of customers to tables and plot a histogram over the number of tables. Investigate how the histogram changes as you change α.

10.7 Derive the Gibbs sampler for the infinite version of the binary mixture described in Exercise 10.3.

10.8 Write some code that will sample from a Dirichlet process. The code should store all previous samples and then copy previous samples or draw new samples from the base. Experiment with different base distributions and concentration parameters.

10.9 Using the MATLAB script provided (MATLAB script: `hdp.m`), experiment with setting the values of the concentration parameters for the top level DP (α) and the lower DPs $(\gamma_1, \ldots, \gamma_J)$.

10.10 Derive a Gibbs sampling scheme for inference in the Hierarchical Dirichlet process. Assume that the base distribution is Gaussian $(H = \mathcal{N}(\mu_0, \sigma_0^2))$ and the observations are one-dimensional real values.

10.11 Derive the collapsed Gibbs sampler for LDA given in Equation 10.13.

10.8 FURTHER READING

[1] David Blei, Andrew Ng, and Michael Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

This is the original paper describing LDA, including a Variational Bayes algorithm for inference.

[2] Thomas Griffiths and Mark Steyvers. Finding scientific topics. *PNAS*, 101:5228–5235, 2004.

An interesting paper where LDA with collapsed Gibbs sampling is used to discover topics from the scientific literature.

[3] David Pollard. *A User's Guide to Measure Theoretic Probability*. Cambridge University Press, first edition, 2002.

Another good introduction to measure theory.

[4] Carl Rasmussen. The infinite Gaussian mixture model. In *Advances in Neural Information Processing Systems 12*, pages 554–560, 2000.

A very nice introduction to the infinite mixture model. This is a good starting point for further reading on Dirichlet processes

[5] Simon Rogers, Mark Girolami, Colin Campbell, and Rainer Breitling. The latent process decomposition of cDNA microarray data sets. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 2(2):143–156, April 2005.

An article by the authors describing an LDA-inspired model for cDNA microarray data.

[6] Jeffrey Rosenthal. *A First Look at Rigorous Probability Theory*. World Scientific Publishing Co, second edition, 2007.

An accessible introduction to measure theory. Worth reading if you want to delve deeper into Dirichlet processes.

[7] Yee Whey Teh, Michael Jordan, Matther Beal, and David Blei. Hierarchical Dirichlet Processes. *Journal of the American Statistical Association*, 101:1566–1581, 2006.

The paper introducing hierarchical Dirichlet processes.