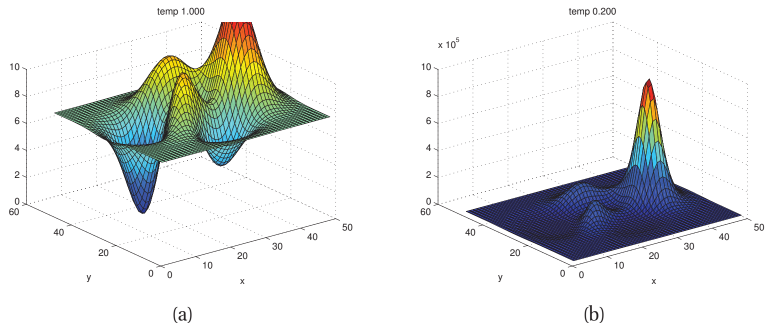


Figure 24.17 An energy surface at different temperatures. Note the different vertical scales. (a) $T = 1$. (b) $T = 0.5$. Figure generated by `saDemoPeaks`.

24.6.1 Simulated annealing

Simulated annealing (Kirkpatrick et al. 1983) is a stochastic algorithm that attempts to find the global optimum of a black-box function $f(\mathbf{x})$. It is closely related to the Metropolis-Hastings algorithm for generating samples from a probability distribution, which we discussed in Section 24.3. SA can be used for both discrete and continuous optimization.

The method is inspired by statistical physics. The key quantity is the **Boltzmann distribution**, which specifies that the probability of being in any particular state $\mathbf{x}$ is given by

$$p(\mathbf{x}) \propto \exp(-f(\mathbf{x})/T) \tag{24.100}$$

where $f(\mathbf{x})$ is the "energy" of the system and T is the computational temperature. As the temperature approaches 0 (so the system is cooled), the system spends more and more time in its minimum energy (most probable) state.

Figure 24.17 gives an example of a 2d function at different temperatures. At high temperatures, $T \gg 1$, the surface is approximately flat, and hence it is easy to move around (i.e., to avoid local optima). As the temperature cools, the largest peaks become larger, and the smallest peaks disappear. By cooling slowly enough, it is possible to "track" the largest peak, and thus find the global optimum. This is an example of a **continuation method**.

We can generate an algorithm from this as follows. At each step, sample a new state according to some proposal distribution $\mathbf{x}' \sim q(\cdot|\mathbf{x}_k)$. For real-valued parameters, this is often simply a random walk proposal, $\mathbf{x}' = \mathbf{x}_k + \boldsymbol{\epsilon}_k$, where $\boldsymbol{\epsilon}_k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$. For discrete optimization, other kinds of local moves must be defined.

Having proposed a new state, we compute

$$\alpha = \exp\left((f(\mathbf{x}) - f(\mathbf{x}'))/T\right) \tag{24.101}$$

We then accept the new state (i.e., set $\mathbf{x}_{k+1} = \mathbf{x}'$) with probability $\min(1, \alpha)$, otherwise we stay in the current state (i.e., set $\mathbf{x}_{k+1} = \mathbf{x}_k$). This means that if the new state has lower energy (is more probable), we will definitely accept it, but it it has higher energy (is less probable), we might still accept, depending on the current temperature. Thus the algorithm allows "down-hill" moves in probability space (up-hill in energy space), but less frequently as the temperature drops.

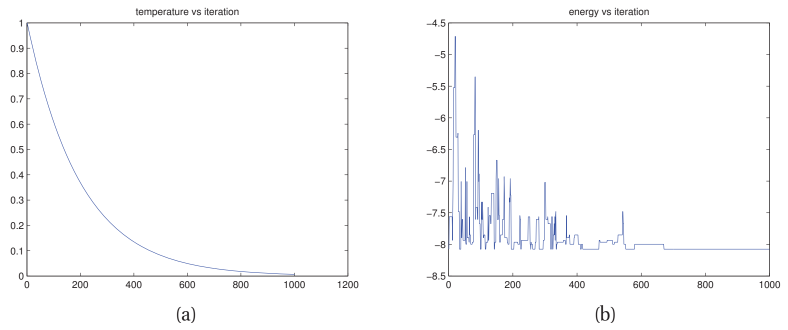


Figure 24.18 A run of simulated annealing on the energy surface in Figure 24.17. (a) Temperature vs iteration. (b) Energy vs iteration. Figure generated by `saDemoPeaks`.

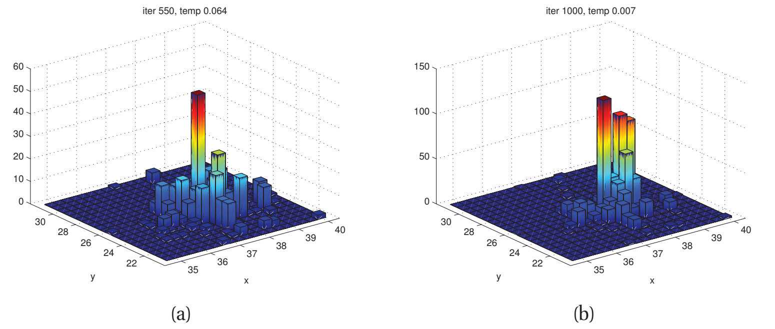


Figure 24.19 Histogram of samples from the annealed "posterior" at 2 different time points produced by simulated annealing on the energy surface shown in Figure 24.17. Note that at cold temperatures, most of the samples are concentrated near the peak at (38,25). Figure generated by `saDemoPeaks`.

The rate at which the temperature changes over time is called the **cooling schedule**. It has been shown (Kirkpatrick et al. 1983) that if one cools sufficiently slowly, the algorithm will provably find the global optimum. However, it is not clear what "sufficient slowly" means. In practice it is common to use an **exponential cooling schedule** of the following form: $T_k = T_0 C^k$, where T_0 is the initial temperature (often $T_0 \sim 1$) and C is the cooling rate (often $C \sim 0.8$). See Figure 24.18(a) for a plot of this cooling schedule. Cooling too quickly means one can get stuck in a local maximum, but cooling too slowly just wastes time. The best cooling schedule is difficult to determine; this is one of the main drawbacks of simulated annealing.

Figure 24.18(b) shows an example of simulated annealing applied to the function in Figure 24.17 using a random walk proposal. We see that the method stochastically reduces the energy over time. Figures 24.19 illustrate (a histogram of) samples drawn from the cooled probability distribution over time. We see that most of the samples are concentrated near the global maximum. When the algorithm has converged, we just return the largest value found.

24.6.2 Annealed importance sampling

We now describe a method known as **annealed importance sampling** (Neal 2001) that combines ideas from simulated annealing and importance sampling in order to draw independent samples from difficult (e.g., multimodal) distributions.

Suppose we want to sample from $p_0(\mathbf{x}) \propto f_0(\mathbf{x})$, but we cannot do so easily; for example, this might represent a multimodal posterior. Suppose however that there is an easier distribution which we can sample from, call it $p_n(\mathbf{x}) \propto f_n(\mathbf{x})$; for example, this might be the prior. We can now construct a sequence of intermediate distributions than move slowly from p_n to p_0 as follows:

$$f_j(\mathbf{x}) = f_0(\mathbf{x})^{\beta_j} f_n(\mathbf{x})^{1-\beta_j} \tag{24.102}$$

where $1 = \beta_0 > \beta_1 > \cdots > \beta_n = 0$, where β_j is an inverse temperature. (Contrast this to the scheme used by simulated annealing which has the form $f_j(\mathbf{x}) = f_0(\mathbf{x})^{\beta_j}$; this makes it hard to sample from p_n.) Furthermore, suppose we have a series of Markov chains $T_j(\mathbf{x}, \mathbf{x}')$ (from $\mathbf{x}$ to $\mathbf{x}'$) which leave each p_j invariant. Given this, we can sample $\mathbf{x}$ from p_0 by first sampling a sequence $\mathbf{z} = (\mathbf{z}_{n-1}, \ldots, \mathbf{z}_0)$ as follows: sample $\mathbf{z}_{n-1} \sim p_n$; sample $\mathbf{z}_{n-2} \sim T_{n-1}(\mathbf{z}_{n-1}, \cdot)$; ...; sample $\mathbf{z}_0 \sim T_1(\mathbf{z}_1, \cdot)$. Finally we set $\mathbf{x} = \mathbf{z}_0$ and give it weight

$$w = \frac{f_{n-1}(\mathbf{z}_{n-1})}{f_n(\mathbf{z}_{n-1})} \frac{f_{n-2}(\mathbf{z}_{n-2})}{f_{n-1}(\mathbf{z}_{n-2})} \cdots \frac{f_1(\mathbf{z}_1)}{f_2(\mathbf{z}_1)} \frac{f_0(\mathbf{z}_0)}{f_1(\mathbf{z}_0)} \tag{24.103}$$

This can be shown to be correct by viewing the algorithm as a form of importance sampling in an extended state space $\mathbf{z} = (\mathbf{z}_0, \ldots, \mathbf{z}_{n-1})$. Consider the following distribution on this state space:

$$p(\mathbf{z}) \propto f(\mathbf{z}) = f_0(\mathbf{z}_0)\tilde{T}_1(\mathbf{z}_0, \mathbf{z}_1)\tilde{T}_2(\mathbf{z}_1, \mathbf{z}_2) \cdots \tilde{T}_{n-1}(\mathbf{z}_{n-2}, \mathbf{z}_{n-1}) \tag{24.104}$$

where $\tilde{T}_j$ is the reversal of T_j:

$$\tilde{T}_j(\mathbf{z}, \mathbf{z}') = T_j(\mathbf{z}', \mathbf{z})p_j(\mathbf{z}')/p_j(\mathbf{z}) = T_j(\mathbf{z}', \mathbf{z})f_j(\mathbf{z}')/f_j(\mathbf{z}) \tag{24.105}$$

It is clear that $\sum_{\mathbf{z}_1, \ldots, \mathbf{z}_{n-1}} f(\mathbf{z}) = f_0(\mathbf{z}_0)$, so we can safely just use the $\mathbf{z}_0$ part of these sequences to recover the original ditribution.

Now consider the proposal distribution defined by the algorithm:

$$q(\mathbf{z}) \propto g(\mathbf{z}) = f_n(\mathbf{z}_{n-1})T_{n-1}(\mathbf{z}_{n-1}, \mathbf{z}_{n-2}) \cdots T_2(\mathbf{z}_2, \mathbf{z}_1)T_1(\mathbf{z}_1, \mathbf{z}_0) \tag{24.106}$$

One can show that the importance weights $w = \frac{f(\mathbf{z}_0, \ldots, \mathbf{z}_{n-1})}{g(\mathbf{z}_0, \ldots, \mathbf{z}_{n-1})}$ are given by Equation 24.103.

24.6.3 Parallel tempering

Another way to combine MCMC and annealing is to run multiple chains in parallel at different temperatures, and allow one chain to sample from another chain at a neighboring temperature. In this way, the high temperature chain can make long distance moves through the state space, and have this influence lower temperature chains. This is known as **parallel tempering**. See e.g., (Earl and Deem 2005) for details.

24.7 Approximating the marginal likelihood

The marginal likelihood $p(\mathcal{D}|M)$ is a key quantity for Bayesian model selection, and is given by

$$p(\mathcal{D}|M) = \int p(\mathcal{D}|\boldsymbol{\theta}, M)p(\boldsymbol{\theta}|M)d\boldsymbol{\theta} \tag{24.107}$$

Unfortunately, this integral is often intractable to compute, for example if we have non conjugate priors, and/or we have hidden variables. In this section, we briefly discuss some ways to approximate this expression using Monte Carlo. See (Gelman and Meng 1998) for a more extensive review.

24.7.1 The candidate method

There is a simple method for approximating the marginal likelihood known as the **Candidate method** (Chib 1995). This exploits the following identity:

$$p(\mathcal{D}|M) = \frac{p(\mathcal{D}|\boldsymbol{\theta}, M)p(\boldsymbol{\theta}|M)}{p(\boldsymbol{\theta}|\mathcal{D}, M)} \tag{24.108}$$

This holds for any value of $\boldsymbol{\theta}$. Once we have picked some value, we can evaluate $p(\mathcal{D}|\boldsymbol{\theta}, M)$ and $p(\boldsymbol{\theta}|M)$ quite easily. If we have some estimate of the posterior near $\boldsymbol{\theta}$, we can then evaluate the denominator as well. This posterior is often approximated using MCMC.

The flaw with this method is that it relies on the assumption that $p(\boldsymbol{\theta}|\mathcal{D}, M)$ has marginalized over all the modes of the posterior, which in practice is rarely possible. Consequently the method can give very inaccurate results in practice (Neal 1998).

24.7.2 Harmonic mean estimate

Newton and Raftery (1994) proposed a simple method for approximating $p(\mathcal{D})$ using the output of MCMC, as follows:

$$1/p(\mathcal{D}) \approx \frac{1}{S}\sum_{s=1}^{S} \frac{1}{p(\mathcal{D}|\boldsymbol{\theta}^s)} \tag{24.109}$$

where $\boldsymbol{\theta}^s \sim p(\boldsymbol{\theta}|\mathcal{D})$. This expression is the harmonic mean of the likelihood of the data under each sample. The theoretical correctness of this expression follows from the following identity:

$$\int \frac{1}{p(\mathcal{D}|\boldsymbol{\theta})}p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta} = \int \frac{1}{p(\mathcal{D}|\boldsymbol{\theta})}\frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})}d\boldsymbol{\theta} = \frac{1}{p(\mathcal{D})}\int p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta} = \frac{1}{p(\mathcal{D})} \tag{24.110}$$

Unfortunately, in practice this method works very poorly. Indeed, Radford Neal called this "the worst Monte Carlo method ever".[6]. The reason it is so bad is that it depends only on samples drawn from the posterior. But the posterior is often very insensitive to the prior, whereas the marginal likelihood is not. We only mention this method in order to warn against its use. We present a better method below.

6. Source: `radfordneal.wordpress.com/2008/08/17/the-harmonic-mean-of-the-likelihood-worst-monte-carlo-method-ever`.

24.7.3 Annealed importance sampling

We can use annealed importance sampling (Section 24.6.2) to evaluate a ratio of partition functions. Notice that $Z_0 = \int f_0(\mathbf{x})d\mathbf{x} = \int f(\mathbf{z})d\mathbf{z}$, and $Z_n = \int f_n(\mathbf{x})d\mathbf{x} = \int g(\mathbf{z})d\mathbf{z}$. Hence

$$\frac{Z_0}{Z_n} = \frac{\int f(\mathbf{z})d\mathbf{z}}{\int g(\mathbf{z})d\mathbf{z}} = \frac{\int \frac{f(\mathbf{z})}{g(\mathbf{z})} g(\mathbf{z})d\mathbf{z}}{\int g(\mathbf{z})d\mathbf{z}} = \mathbb{E}_q\left[\frac{f(\mathbf{z})}{g(\mathbf{z})}\right] \approx \frac{1}{S}\sum_{s=1}^{S} w_s \tag{24.111}$$

If f_n is a prior and f_0 is the posterior, we can estimate $Z_n = p(\mathcal{D})$ using the above equation, provided the prior has a known normalization constant Z_0. This is generally considered the method of choice for evaluating difficult partition functions.

Exercises

Exercise 24.1 Gibbs sampling from a 2D Gaussian

Suppose $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu} = (1, 1)$ and $\boldsymbol{\Sigma} = (1, -0.5; -0.5, 1)$. Derive the full conditionals $p(x_1|x_2)$ and $p(x_2|x_1)$. Implement the algorithm and plot the 1d marginals $p(x_1)$ and $p(x_2)$ as histograms. Superimpose a plot of the exact marginals.

Exercise 24.2 Gibbs sampling for a 1D Gaussian mixture model

Consider applying Gibbs sampling to a univariate mixture of Gaussians, as in Section 24.2.3. Derive the expressions for the full conditionals. Hint: if we know $z_n = j$ (say), then μ_j gets "connected" to x_n, but all other values of μ_i, for all $i \neq j$, are irrelevant. (This is an example of context-specific independence, where the structure of the graph simplifies once we have assigned values to some of the nodes.) Hence, given all the z_n values, the posteriors of the μ's should be independent, so the conditional of μ_j should be independent of $\boldsymbol{\mu}_{-j}$. (Similarly for σ_j.)

Exercise 24.3 Gibbs sampling from the Potts model

Modify the code in `gibbsDemoIsing` to draw samples from a Potts prior at different temperatures, as in Figure 19.8.

Exercise 24.4 Full conditionals for hierarchical model of Gaussian means

Let us reconsider the Gaussian-Gaussian model in Section 5.6.2 for modelling multiple related mean parameters θ_j. In this exercise we derive a Gibbs sampler instead of using EB. Suppose, following (Hoff 2009, p134)), that we use the following conjugate priors on the hyper-parameters:

$$\mu \sim \mathcal{N}(\mu_0, \gamma_0^2) \tag{24.112}$$

$$\tau^2 \sim \mathrm{IG}(\eta_0/2, \eta_0\tau_0^2/2) \tag{24.113}$$

$$\sigma^2 \sim \mathrm{IG}(\nu_0/2, \nu_0\sigma_0^2/2) \tag{24.114}$$

We can set $\boldsymbol{\eta} = (\mu_0, \gamma_0, \eta_0, \tau_0, \nu_0, \sigma_0)$ to uninformative values. Given this model specification, show that the full conditionals for μ, τ, σ and the θ_j are as follows:

$$p(\mu|\theta_{1:D}, \tau^2) = \mathcal{N}(\mu|\frac{D\overline{\theta}/\tau^2 + \mu_0/\gamma_0^2}{D/\tau^2 + 1/\gamma_0^2}, [D/\tau^2 + 1/\gamma_0^2]^{-1}) \tag{24.115}$$

$$p(\theta_j|\mu, \tau^2, \mathcal{D}_j, \sigma^2) = \mathcal{N}(\theta_j|\frac{N_j\overline{x}_j/\sigma^2 + 1/\tau^2}{N_j/\sigma^2 + 1/\tau^2}, [N_j/\sigma^2 + 1/\tau^2]^{-1}) \tag{24.116}$$

$$p(\tau^2|\theta_{1:D}, \mu) = \text{IG}(\tau^2|\frac{\eta_0 + D}{2}, \frac{\eta_0\tau_0^2 + \sum_j(\theta_j - \mu)^2}{2}) \tag{24.117}$$

$$p(\sigma^2|\boldsymbol{\theta}_{1:D}, \mathcal{D}) = \text{IG}(\sigma^2|\frac{1}{2}[\nu_0 + \sum_{j=1}^{D} N_j], \frac{1}{2}[\nu_0\sigma_0^2 + \sum_{j=1}^{D}\sum_{i=1}^{N_j}(x_{ij} - \theta_j)^2]) \tag{24.118}$$

Exercise 24.5 Gibbs sampling for robust linear regression with a Student t likelihood

Modify the EM algorithm in Exercise 11.12 to perform Gibbs sampling for $p(\mathbf{w}, \sigma^2, \mathbf{z}|\mathcal{D}, \nu)$.

Exercise 24.6 Gibbs sampling for probit regression

Modify the EM algorithm in Section 11.4.6 to perform Gibbs sampling for $p(\mathbf{w}, \mathbf{z}|\mathcal{D})$. Hint: we can sample from a truncated Gaussian, $\mathcal{N}(z|\mu, \sigma)\mathbb{I}(a \leq z \leq b)$ in two steps: first sample $u \sim U(\Phi((a - \mu)/\sigma), \Phi((b - \mu)/\sigma))$, then set $z = \mu + \sigma\Phi^{-1}(u)$ (Robert 1995).

Exercise 24.7 Gibbs sampling for logistic regression with the Student approximation

Derive the full conditionals for the joint model defined by Equations 24.88 to 24.91.

25 *Clustering*

25.1 Introduction

Clustering is the process of grouping similar objects together. There are two kinds of inputs we might use. In **similarity-based clustering**, the input to the algorithm is an $N \times N$ **dissimilarity matrix** or **distance matrix** $\mathbf{D}$. In **feature-based clustering**, the input to the algorithm is an $N \times D$ feature matrix or design matrix $\mathbf{X}$. Similarity-based clustering has the advantage that it allows for easy inclusion of domain-specific similarity or kernel functions (Section 14.2). Feature-based clustering has the advantage that it is applicable to "raw", potentially noisy data. We will see examples of both below.

In addition to the two types of input, there are two possible types of output: **flat clustering**, also called **partitional clustering**, where we partition the objects into disjoint sets; and **hierarchical clustering**, where we create a nested tree of partitions. We will discuss both of these below. Not surprisingly, flat clusterings are usually faster to create ($O(ND)$ for flat vs $O(N^2 \log N)$ for hierarchical), but hierarchical clusterings are often more useful. Furthermore, most hierarchical clustering algorithms are deterministic and do not require the specification of K, the number of clusters, whereas most flat clustering algorithms are sensitive to the initial conditions and require some model selection method for K. (We will discuss how to choose K in more detail below.)

The final distinction we will make in this chapter is whether the method is based on a probabilistic model or not. One might wonder why we even bother discussing non-probabilistic methods for clustering. The reason is two-fold: first, they are widely used, so readers should know about them; second, they often contain good ideas, which can be used to speed up inference in a probabilistic models.

25.1.1 Measuring (dis)similarity

A dissimilarity matrix $\mathbf{D}$ is a matrix where $d_{i,i} = 0$ and $d_{i,j} \geq 0$ is a measure of "distance" between objects i and j. Subjectively judged dissimilarities are seldom distances in the strict sense, since the **triangle inequality**, $d_{i,j} \leq d_{i,k} + d_{j,k}$, often does not hold. Some algorithms require $\mathbf{D}$ to be a true distance matrix, but many do not. If we have a similarity matrix $\mathbf{S}$, we can convert it to a dissimilarity matrix by applying any monotonically decreasing function, e.g., $\mathbf{D} = \max(\mathbf{S}) - \mathbf{S}$.

The most common way to define dissimilarity between objects is in terms of the dissimilarity of their attributes:

$$\Delta(\mathbf{x}_i, \mathbf{x}_{i'}) = \sum_{j=1}^{D} \Delta_j(x_{ij}, x_{i'j}) \tag{25.1}$$

Some common attribute dissimilarity functions are as follows:

- Squared (Euclidean) distance:

  $$\Delta_j(x_{ij}, x_{i'j}) = (x_{ij} - x_{i'j})^2 \tag{25.2}$$

  Of course, this only makes sense if attribute j is real-valued.

- Squared distance strongly emphasizes large differences (because differences are squared). A more robust alternative is to use an ℓ_1 distance:

  $$\Delta_j(x_{ij}, x_{i'j}) = |x_{ij} - x_{i'j}| \tag{25.3}$$

  This is also called **city block distance**, since, in 2D, the distance can be computed by counting how many rows and columns we have to move horizontally and vertically to get from $\mathbf{x}_i$ to $\mathbf{x}_{i'}$.

- If $\mathbf{x}_i$ is a vector (e.g., a time-series of real-valued data), it is common to use the correlation coefficient (see Section 2.5.1). If the data is standardized, then $\text{corr}[\mathbf{x}_i, \mathbf{x}_{i'}] = \sum_j x_{ij}x_{i'j}$, and hence $\sum_j (x_{ij} - x_{i'j})^2 = 2(1 - \text{corr}[\mathbf{x}_i, \mathbf{x}_{i'}])$. So clustering based on correlation (similarity) is equivalent to clustering based on squared distance (dissimilarity).

- For ordinal variables, such as {low, medium, high}, it is standard to encode the values as real-valued numbers, say $1/3, 2/3, 3/3$ if there are 3 possible values. One can then apply any dissimilarity function for quantitative variables, such as squared distance.

- For categorical variables, such as {red, green, blue}, we usually assign a distance of 1 if the features are different, and a distance of 0 otherwise. Summing up over all the categorical features gives

  $$\Delta(\mathbf{x}_i, \mathbf{x}_i) = \sum_{j=1}^{D} \mathbb{I}(x_{ij} \neq x_{i'j}) \tag{25.4}$$

  This is called the **hamming distance**.

25.1.2 Evaluating the output of clustering methods *

The validation of clustering structures is the most difficult and frustrating part of cluster analysis. Without a strong effort in this direction, cluster analysis will remain a black art accessible only to those true believers who have experience and great courage. — Jain and Dubes (Jain and Dubes 1988)

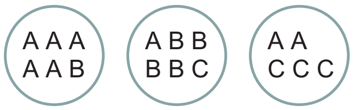


Figure 25.1 Three clusters with labeled objects inside. Based on Figure 16.4 of (Manning et al. 2008).

Clustering is an unupervised learning technique, so it is hard to evaluate the quality of the output of any given method. If we use probabilistic models, we can always evaluate the likelihood of a test set, but this has two drawbacks: first, it does not directly assess any clustering that is discovered by the model; and second, it does not apply to non-probabilistic methods. So now we discuss some performance measures not based on likelihood.

Intuitively, the goal of clustering is to assign points that are similar to the same cluster, and to ensure that points that are dissimilar are in different clusters. There are several ways of measuring these quantities e.g., see (Jain and Dubes 1988; Kaufman and Rousseeuw 1990). However, these internal criteria may be of limited use. An alternative is to rely on some external form of data with which to validate the method. For example, suppose we have labels for each object, as in Figure 25.1. (Equivalently, we can have a reference clustering; given a clustering, we can induce a set of labels and vice versa.) Then we can compare the clustering with the labels using various metrics which we describe below. We will use some of these metrics later, when we compare clustering methods.

25.1.2.1 Purity

Let N_{ij} be the number of objects in cluster i that belong to class j, and let $N_i = \sum_{j=1}^{C} N_{ij}$ be the total number of objects in cluster i. Define $p_{ij} = N_{ij}/N_i$; this is the empirical distribution over class labels for cluster i. We define the **purity** of a cluster as $p_i \triangleq \max_j p_{ij}$, and the overall purity of a clustering as

$$\text{purity} \triangleq \sum_i \frac{N_i}{N} p_i \tag{25.5}$$

For example, in Figure 25.1, we have that the purity is

$$\frac{6}{17}\frac{5}{6} + \frac{6}{17}\frac{4}{6} + \frac{5}{17}\frac{3}{5} = \frac{5+4+3}{17} = 0.71 \tag{25.6}$$

The purity ranges between 0 (bad) and 1 (good). However, we can trivially achieve a purity of 1 by putting each object into its own cluster, so this measure does not penalize for the number of clusters.

25.1.2.2 Rand index

Let $U = \{u_1, \ldots, u_R\}$ and $V = \{v_1, \ldots, V_C\}$ be two different partitions of the N data points, i.e., two different (flat) clusterings. For example, U might be the estimated clustering and V is reference clustering derived from the class labels. Now define a 2×2 contingency table, containing the following numbers: TP is the number of pairs that are in the same cluster in both U and V (true positives); TN is the number of pairs that are in the different clusters in both U and V (true negatives); FN is the number of pairs that are in the different clusters in U but the same cluster in V (false negatives); and FP is the number of pairs that are in the same cluster in U but different clusters in V (false positives). A common summary statistic is the **Rand index**:

$$R \triangleq \frac{TP + TN}{TP + FP + FN + TN} \tag{25.7}$$

This can be interpreted as the fraction of clustering decisions that are correct. Clearly $0 \leq R \leq 1$.

For example, consider Figure 25.1, The three clusters contain 6, 6 and 5 points, so the number of "positives" (i.e., pairs of objects put in the same cluster, regardless of label) is

$$TP + FP = \binom{6}{2} + \binom{6}{2} + \binom{5}{2} = 40 \tag{25.8}$$

Of these, the number of true positives is given by

$$TP = \binom{5}{2} + \binom{5}{2} + \binom{3}{2} + \binom{2}{2} = 20 \tag{25.9}$$

where the last two terms come from cluster 3: there are $\binom{3}{2}$ pairs labeled C and $\binom{2}{2}$ pairs labeled A. So $FP = 40 - 20 = 20$. Similarly, one can show $FN = 24$ and $TN = 72$. So the Rand index is $(20 + 72)/(20 + 20 + 24 + 72) = 0.68$.

The Rand index only achieves its lower bound of 0 if $TP = TN = 0$, which is a rare event. One can define an **adjusted Rand index** (Hubert and Arabie 1985) as follows:

$$AR \triangleq \frac{\text{index} - \text{expected index}}{\text{max index} - \text{expected index}} \tag{25.10}$$

Here the model of randomness is based on using the generalized hyper-geometric distribution, i.e., the two partitions are picked at random subject to having the original number of classes and objects in each, and then the expected value of $TP + TN$ is computed. This model can be used to compute the statistical significance of the Rand index.

The Rand index weights false positives and false negatives equally. Various other summary statistics for binary decision problems, such as the F-score (Section 5.7.2.2), can also be used. One can compute their frequentist sampling distribution, and hence their statistical significance, using methods such as bootstrap.

25.1.2.3 Mutual information

Another way to measure cluster quality is to compute the mutual information between U and V (Vaithyanathan and Dom 1999). To do this, let $p_{UV}(i,j) = \frac{|u_i \cap v_j|}{N}$ be the probability that a randomly chosen object belongs to cluster u_i in U and v_j in V. Also, let $p_U(i) = |u_i|/N$ be the be the probability that a randomly chosen object belongs to cluster u_i in U; define

$p_V(j) = |v_j|/N$ similarly. Then we have

$$\mathbb{I}(U,V) = \sum_{i=1}^{R}\sum_{j=1}^{C} p_{UV}(i,j) \log \frac{p_{UV}(i,j)}{p_U(i)p_V(j)} \tag{25.11}$$

This lies between 0 and $\min\{\mathbb{H}(U), \mathbb{H}(V)\}$. Unfortunately, the maximum value can be achieved by using lots of small clusters, which have low entropy. To compensate for this, we can use the **normalized mutual information**,

$$NMI(U,V) \triangleq \frac{\mathbb{I}(U,V)}{(\mathbb{H}(U) + \mathbb{H}(V))/2} \tag{25.12}$$

This lies between 0 and 1. A version of this that is adjusted for chance (under a particular random data model) is described in (Vinh et al. 2009). Another variant, called **variation of information**, is described in (Meila 2005).

25.2 Dirichlet process mixture models

The simplest approach to (flat) clustering is to use a finite mixture model, as we discussed in Section 11.2.3. This is sometimes called **model-based clustering**, since we define a probabilistic model of the data, and optimize a well-defined objective (the likelihood or posterior), as opposed to just using some heuristic algorithm.

The principle problem with finite mixture models is how to choose the number of components K. We discussed several techniques in Section 11.5. However, in many cases, there is no well-defined number of clusters. Even in the simple 2d height-weight data (Figure 1.8), it is not clear if the "correct" value of K should be 2, 3, or 4. It would be much better if we did not have to choose K at all.

In this section, we discuss **infinite mixture models**, in which we do not impose any a priori bound on K. To do this, we will use a **non-parametric prior** based on the **Dirichlet process** (DP). This allows the number of clusters to grow as the amount of data increases. It will also prove useful later when we discuss hiearchical clustering.

The topic of **non-parametric Bayes** is currently very active, and we do not have space to go into details (see (Hjort et al. 2010) for a recent book on the topic). Instead we just give a brief review of the DP and its application to mixture modeling, based on the presentation in (Sudderth 2006, sec 2.2).

25.2.1 From finite to infinite mixture models

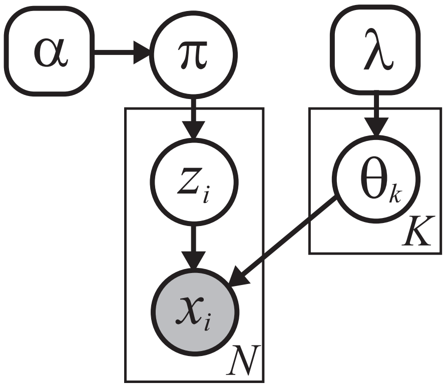
Consider a finite mixture model, as shown in Figure 25.2(a). The usual representation is as follows:

$$p(\mathbf{x}_i|z_i = k, \boldsymbol{\theta}) = p(\mathbf{x}_i|\boldsymbol{\theta}_k) \tag{25.13}$$

$$p(z_i = k|\boldsymbol{\pi}) = \pi_k \tag{25.14}$$

$$p(\boldsymbol{\pi}|\alpha) = \mathrm{Dir}(\boldsymbol{\pi}|(\alpha/K)\mathbf{1}_K) \tag{25.15}$$

The form of $p(\boldsymbol{\theta}_k|\lambda)$ is chosen to be conjugate to $p(\mathbf{x}_i|\boldsymbol{\theta}_k)$. We can write $p(\mathbf{x}_i|\boldsymbol{\theta}_k)$ as $\mathbf{x}_i \sim F(\boldsymbol{\theta}_{z_i})$, where F is the observation distribution. Similarly, we can write $\boldsymbol{\theta}_k \sim H(\lambda)$, where H is the prior.



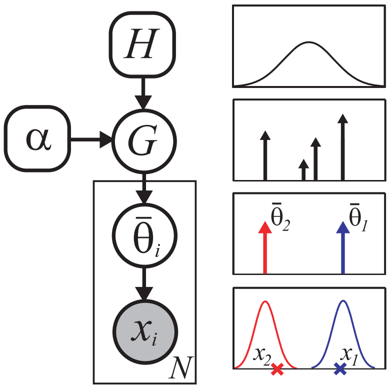


Figure 25.2 Two different representations of a finite mixture model. Left: traditional representation. Right: representation where parameters are samples from G, a discrete measure. The picture on the right illustrates the case where $K = 4$, and we sample 4 Gaussian means θ_k from a Gaussian prior $H(.|\lambda)$. The height of the spikes reflects the mixing weights π_k. This weighted sum of delta functions is G. We then generate two parameters, $\overline{\theta}_1$ and $\overline{\theta}_2$, from G, one per data point. Finally, we generate two data points, x_1 and x_2, from $\mathcal{N}(\overline{\theta}_1, \sigma^2)$ and $\mathcal{N}(\overline{\theta}_2, \sigma^2)$. Source: Figure 2.9 of (Sudderth 2006) . Used with kind permission of Erik Sudderth.

An equivalent representation for this model is shown in Figure 25.2(b). Here $\overline{\boldsymbol{\theta}}_i$ is the parameter used to generate observation $\mathbf{x}_i$; these parameters are sampled from distribution G, which has the form

$$G(\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \delta_{\boldsymbol{\theta}_k}(\boldsymbol{\theta}) \tag{25.16}$$

where $\boldsymbol{\pi} \sim \text{Dir}(\frac{\alpha}{K}\mathbf{1})$, and $\boldsymbol{\theta}_k \sim H$. Thus we see that G is a finite mixture of delta functions, centered on the cluster parameters $\boldsymbol{\theta}_k$. The probability that $\overline{\boldsymbol{\theta}}_i$ is equal to $\boldsymbol{\theta}_k$ is exactly π_k, the prior probability for that cluster.

If we sample from this model, we will always (with probability one) get exactly K clusters, with data points scattered around the cluster centers. We would like a more flexible model, that can generate a variable number of clusters. Furthermore, the more data we generate, the more likely we should be to see a new cluster. The way to do this is to replace the discrete distribution G with a **random probability measure**. Below we will show that the Dirichlet process, denoted $G \sim \text{DP}(\alpha, H)$, is one way to do this.

Before we go into the details, we show some samples from this non-parametric model in Figure 25.3. We see that it has the desired properties of generating a variable number of clusters, with more clusters as the amount of data increases. The resulting samples look much more like real data than samples from a finite mixture model.

Of course, working with an "infinite" model sounds scary. Fortunately, as we show below, even though this model is potentially infinite, we can perform inference using an amount of computation that is not only tractable, but is often much less than that required to fit a set

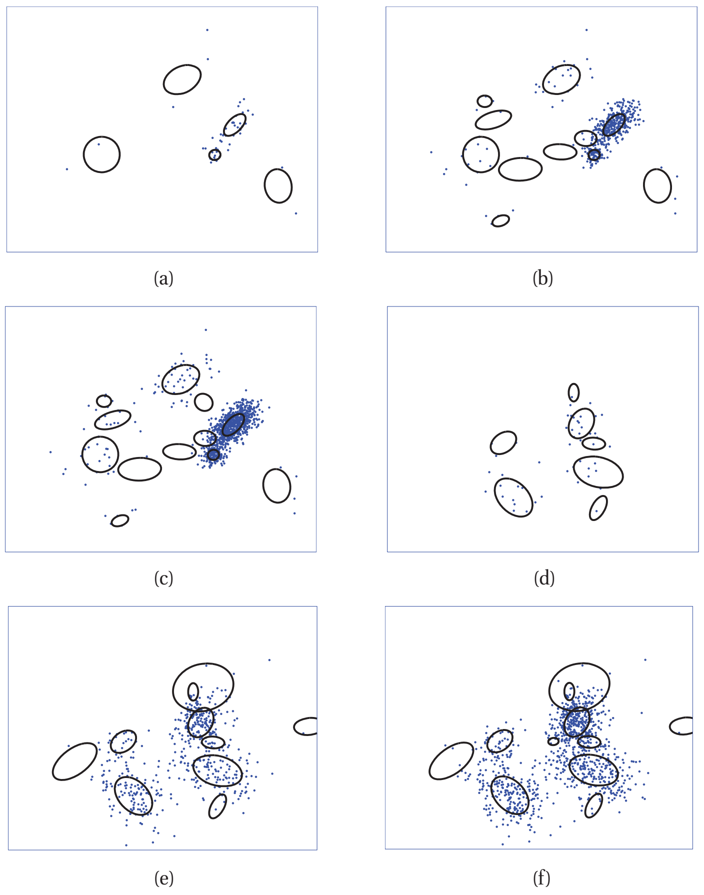


Figure 25.3 Some samples from a Dirichlet process mixture model of 2D Gaussians, with concentration parameter $\alpha = 1$. From left to right, we show $N = 50$, $N = 500$ and $N = 1000$ samples. Each row is a different run. We also show the model parameters as ellipses, which are sampled from a vague NIW base distribution. Based on Figure 2.25 of (Sudderth 2006). Figure generated by `dpmSampleDemo`, written by Yee-Whye Teh.

of finite mixture models for different K. The intuitive reason is that we can get evidence that certain values of K are appropriate (have high posterior support) long before we have been able to estimate the parameters, so we can focus our computational efforts on models of appropriate complexity. Thus going to the infinite limit can sometimes be faster. This is especially true when we have multiple model selection problems to solve.

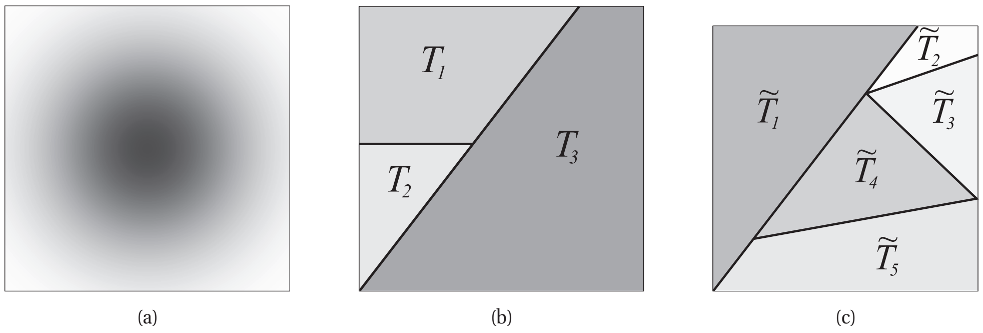


Figure 25.4 (a) A base measure H on a 2d space Θ. (b) One possible partition into $K = 3$ regions, where the shading of cell T_k is proportional to $\mathbb{E}[G(T_k)] = H(T_k)$. (c) A refined partition into $K = 5$ regions. Source: Figure 2.21 of (Sudderth 2006). Used with kind permission of Erik Sudderth.

25.2.2 The Dirichlet process

Recall from Chapter 15 that a Gaussian process is a distribution over functions of the form $f : \mathcal{X} \rightarrow R$. It is defined implicitly by the requirement that $p(f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N))$ be jointly Gaussian, for any set of points $\mathbf{x}_i \in \mathcal{X}$. The parameters of this Gaussian can be computed using a mean function $\mu()$ and covariance (kernel) function $K()$. We write $f \sim \text{GP}(\mu(), K())$. Furthermore, the GP is consistently defined, so that $p(f(\mathbf{x}_1))$ can be derived from $p(f(\mathbf{x}_1), f(\mathbf{x}_2))$, etc.

A **Dirichlet process** is a distribution over probability measures $G : \Theta \rightarrow \mathbb{R}^+$, where we require $G(\theta) \geq 0$ and $\int_\Theta G(\theta) d\theta = 1$. The DP is defined implicitly by the requirement that $(G(T_1), \ldots, G(T_K))$ has a joint Dirichlet distribution

$$\text{Dir}(\alpha H(T_1), \ldots, \alpha H(T_K)) \tag{25.17}$$

for any finite partition $(T_1, \ldots, T_K)$ of Θ. If this is the case, we write $G \sim \text{DP}(\alpha, H)$, where α is called the **concentration parameter** and H is called the **base measure**.[1]

An example of a DP is shown in Figure 25.4, where the base measure is a 2d Gaussian. The distribution over all the cells, $p(G(T_1), \ldots, G(T_K))$, is Dirichlet, so the marginals in each cell are beta distributed:

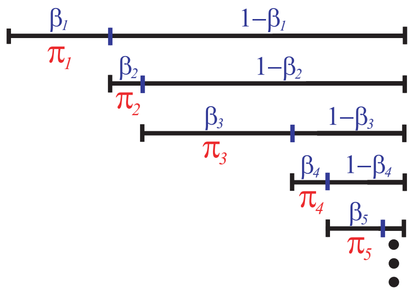
$$\text{Beta}(\alpha H(T_i), \alpha \sum_{j \neq i} H(T_j)) \tag{25.18}$$

The DP is consistently defined in the sense that if T_1 and T_2 form a partition of $\tilde{T}_1$, then $G(T_1) + G(T_2)$ and $G(\tilde{T}_1)$ both follow the same beta distribution.

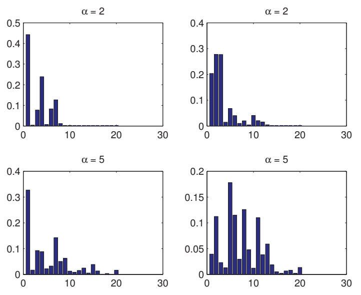
Recall that if $\boldsymbol{\pi} \sim \text{Dir}(\boldsymbol{\alpha})$, and $z|\boldsymbol{\pi} \sim \text{Cat}(\boldsymbol{\pi})$, then we can integrate out $\boldsymbol{\pi}$ to get the predictive distribution for the Dirichlet-multinoulli model:

$$z \sim \text{Cat}(\alpha_1/\alpha_0, \ldots, \alpha_K/\alpha_0) \tag{25.19}$$

1. Unlike a GP, knowing something about $G(T_k)$ does not tell us anything about $G(T_{k'})$, beyond the sum-to-one constraint; we say that the DP is a **neutral process**. Other stochastic processes can be defined that do not have this property, but they are not so computationally convenient.



(a)



(b)

Figure 25.5 Illustration of the stick breaking construction. (a) We have a unit length stick, which we break at a random point β_1; the length of the piece we keep is called π_1; we then recursively break off pieces of the remaining stick, to generate $\pi_2, \pi_3, \ldots$. Source: Figure 2.22 of (Sudderth 2006). Used with kind permission of Erik Sudderth. (b) Samples of π_k from this process for $\alpha = 2$ (top row) and $\alpha = 5$ (bottom row). Figure generated by `stickBreakingDemo`, written by Yee-Whye Teh.

where $\alpha_0 = \sum_k \alpha_k$. In other words, $p(z = k|\boldsymbol{\alpha}) = \alpha_k/\alpha_0$. Also, the updated posterior for $\boldsymbol{\pi}$ given one observation is given by

$$\boldsymbol{\pi}|z \sim \text{Dir}(\alpha_1 + \mathbb{I}(z = 1), \ldots, \alpha_K + \mathbb{I}(z = K)) \tag{25.20}$$

The DP generalizes this to arbitrary partitions. If $G \sim \text{DP}(\alpha, H)$, then $p(\boldsymbol{\theta} \in T_i) = H(T_i)$ and the posterior is

$$p(G(T_1), \ldots, G(T_K)|\boldsymbol{\theta}, \alpha, H) = \text{Dir}(\alpha H(T_1) + \mathbb{I}(\boldsymbol{\theta} \in T_1), \ldots, \alpha H(T_K) + \mathbb{I}(\boldsymbol{\theta} \in T_K)) \tag{25.21}$$

This holds for any set of partitions. Hence if we observe multiple samples $\overline{\boldsymbol{\theta}}_i \sim G$, the new posterior is given by

$$G|\overline{\boldsymbol{\theta}}_1, \ldots, \overline{\boldsymbol{\theta}}_N, \alpha, H \sim \text{DP}\left(\alpha + N, \frac{1}{\alpha + N}\left(\alpha H + \sum_{i=1}^N \delta_{\boldsymbol{\theta}_i}\right)\right) \tag{25.22}$$

Thus we see that the DP effectively defines a conjugate prior for arbitrary measurable spaces. The concentration parameter α is like the effective sample size of the base measure H.

25.2.2.1 Stick breaking construction of the DP

Our discussion so far has been very abstract. We now give a constructive definition for the DP, known as the **stick-breaking construction**.

Let $\pi = \{\pi_k\}_{k=1}^{\infty}$ be an infinite sequence of mixture weights derived from the following process:

$$\beta_k \sim \text{Beta}(1, \alpha) \tag{25.23}$$

$$\pi_k = \beta_k \prod_{l=1}^{k-1}(1 - \beta_l) = \beta_k(1 - \sum_{l=1}^{k-1} \pi_l) \tag{25.24}$$

This is often denoted by

$$\pi \sim \mathrm{GEM}(\alpha) \tag{25.25}$$

where GEM stands for Griffiths, Engen and McCloskey (this term is due to (Ewens 1990)). Some samples from this process are shown in Figure 25.5. One can show that this process process will terminate with probability 1, although the number of elements it generates increases with α. Furthermore, the size of the π_k components decreases on average.

Now define

$$G(\boldsymbol{\theta}) = \sum_{k=1}^{\infty} \pi_k \delta_{\boldsymbol{\theta}_k}(\boldsymbol{\theta}) \tag{25.26}$$

where $\pi \sim \mathrm{GEM}(\alpha)$ and $\boldsymbol{\theta}_k \sim H$. Then one can show that $G \sim \mathrm{DP}(\alpha, H)$.

As a consequence of this construction, we see that samples from a DP are **discrete with probability one**. In other words, if you keep sampling it, you will get more and more repetitions of previously generated values. So if we sample $\overline{\boldsymbol{\theta}}_i \sim G$, we will see repeated values; let us number the unique values $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$, etc. Data sampled from $\overline{\boldsymbol{\theta}}_i$ will therefore cluster around the $\boldsymbol{\theta}_k$. This is evident in Figure 25.3, where most data comes from the Gaussians with large π_k values, represented by ellipses with thick borders. This is our first indication that the DP might be useful for clustering.

25.2.2.2 The Chinese restaurant process (CRP)

Working with infinite dimensional sticks is problematic. However, we can exploit the clustering property to draw samples form a GP, as we now show.

The key result is this: If $\overline{\boldsymbol{\theta}}_i \sim G$ are N observations from $G \sim \mathrm{DP}(\alpha, H)$, taking on K distinct values $\boldsymbol{\theta}_k$, then the predictive distribution of the next observation is given by

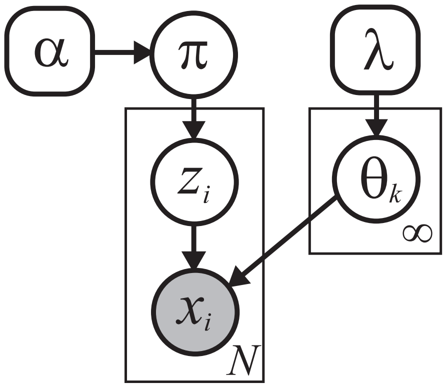
$$p(\overline{\boldsymbol{\theta}}_{N+1} = \boldsymbol{\theta} | \overline{\boldsymbol{\theta}}_{1:N}, \alpha, H) = \frac{1}{\alpha + N}\left(\alpha H(\boldsymbol{\theta}) + \sum_{k=1}^{K} N_k \delta_{\overline{\boldsymbol{\theta}}_k}(\boldsymbol{\theta})\right) \tag{25.27}$$

where N_k is the number of previous observations equal to $\boldsymbol{\theta}_k$. This is called the **Polya urn** or **Blackwell-MacQueen** sampling scheme. This provides a constructive way to sample from a DP.

It is much more convenient to work with discrete variables z_i which specify which value of $\boldsymbol{\theta}_k$ to use. That is, we define $\overline{\boldsymbol{\theta}}_i = \boldsymbol{\theta}_{z_i}$. Based on the above expression, we have

$$p(z_{N+1} = z | \mathbf{z}_{1:N}, \alpha) = \frac{1}{\alpha + N}\left(\alpha \mathbb{I}(z = k^*) + \sum_{k=1}^{K} N_k \mathbb{I}(z = k)\right) \tag{25.28}$$

where k^* represents a new cluster index that has not yet been used. This is called the **Chinese restaurant process** or **CRP**, based on the seemingly infinite supply of tables at certain Chinese restaurants. The analogy is as follows: The tables are like clusters, and the customers are like observations. When a person enters the restaurant, he may choose to join an existing table with probability proportional to the number of people already sitting at this table (the N_k); otherwise, with a probability that diminishes as more people enter the room (due to the $1/(\alpha + N)$ term),



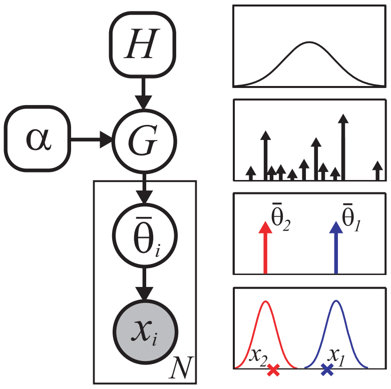


Figure 25.6 Two views of a DP mixture model. Left: infinite number of clusters parameters, $\boldsymbol{\theta}_k$, and $\pi \sim \text{GEM}(\alpha)$. Right: G is drawn from a DP. Compare to Figure 25.2. Source: Figure 2.24 of (Sudderth 2006). Used with kind permission of Erik Sudderth.

he may choose to sit at a new table k^*. The result is a distribution over **partitions of the integers**, which is like a distribution of customers to tables.

The fact that currently occupied tables are more likely to get new customers is sometimes called the **rich get richer** phenomenon. Indeed, one can derive an expression for the distribution of cluster sizes induced by this prior process; it is basically a power law. The number of occupied tables K almost surely approaches $\alpha \log(N)$ as $N \to \infty$, showing that the model complexity will indeed grow logarithmically with dataset size. More flexible priors over cluster sizes can also be defined, such as the two-parameter **Pitman-Yor process**.

25.2.3 Applying Dirichlet processes to mixture modeling

The DP is not particularly useful as a model for data directly, since data vectors rarely repeat exactly. However, it is useful as a prior for the parameters of a stochastic data generating mechanism, such as a mixture model. To create such a model, we follow exactly the same setup as Section 11.2, but we define $G \sim \text{DP}(\alpha, H)$. Equivalently, we can write the model as follows:

$$\boldsymbol{\pi} \sim \text{GEM}(\alpha) \quad (25.29)$$

$$z_i \sim \boldsymbol{\pi} \quad (25.30)$$

$$\boldsymbol{\theta}_k \sim H(\lambda) \quad (25.31)$$

$$\mathbf{x}_i \sim F(\boldsymbol{\theta}_{z_i}) \quad (25.32)$$

This is illustrated in Figure 25.6. We see that G is now a random draw of an unbounded number of parameters $\boldsymbol{\theta}_k$ from the base distribution H, each with weight π_k. Each data point $\mathbf{x}_i$ is generated by sampling its own "private" parameter $\overline{\boldsymbol{\theta}}_i$ from G. As we get more and more data, it becomes increasingly likely that $\overline{\boldsymbol{\theta}}_i$ will be equal to one of the $\boldsymbol{\theta}_k$'s we have seen before, and thus $\mathbf{x}_i$ will be generated close to an existing datapoint.

25.2.4 Fitting a DP mixture model

The simplest way to fit a DPMM is to modify the collapsed Gibbs sampler of Section 24.2.4. From Equation 24.23 we have

$$p(z_i = k|\mathbf{z}_{-i}, \mathbf{x}, \alpha, \boldsymbol{\lambda}) \quad \propto \quad p(z_i = k|\mathbf{z}_{-i}, \alpha)p(\mathbf{x}_i|\mathbf{x}_{-i}, z_i = k, \mathbf{z}_{-i}, \boldsymbol{\lambda}) \tag{25.33}$$

By exchangeability, we can assume that z_i is the last customer to enter the restaurant. Hence the first term is given by

$$p(z_i|\mathbf{z}_{-i}, \alpha) = \frac{1}{\alpha + N - 1}\left(\alpha\mathbb{I}(z_i = k^*) + \sum_{k=1}^{K} N_{k,-i}\mathbb{I}(z_i = k)\right) \tag{25.34}$$

where K is the number of clusters used by $\mathbf{z}_{-i}$, and k^* is a new cluster. Another way to write this is as follows:

$$p(z_i = k|\mathbf{z}_{-i}, \alpha) \quad = \quad \begin{cases} \frac{N_{k,-i}}{\alpha+N-1} & \text{if } k \text{ has been seen before} \\ \frac{\alpha}{\alpha+N-1} & \text{if } k \text{ is a new cluster} \end{cases} \tag{25.35}$$

Interestingly, this is equivalent to Equation 24.26, which has the form $p(z_i = k|\mathbf{z}_{-i}, \alpha) = \frac{N_{k,-i}+\alpha/K}{\alpha+N-1}$, in the $K \to \infty$ limit (Rasmussen 2000; Neal 2000).

To compute the second term, $p(\mathbf{x}_i|\mathbf{x}_{-i}, z_i = k, \mathbf{z}_{-i}, \boldsymbol{\lambda})$, let us partition the data $\mathbf{x}_{-i}$ into clusters based on $\mathbf{z}_{-i}$. Let $\mathbf{x}_{-i,c} = \{\mathbf{x}_j : z_j = c, j \neq i\}$ be the data assigned to cluster c. If $z_i = k$, then $\mathbf{x}_i$ is conditionally independent of all the data points except those assigned to cluster k. Hence we have

$$p(\mathbf{x}_i|\mathbf{x}_{-i}, \mathbf{z}_{-i}, z_i = k, \boldsymbol{\lambda}) \quad = \quad p(\mathbf{x}_i|\mathbf{x}_{-i,k}, \boldsymbol{\lambda}) = \frac{p(\mathbf{x}_i, \mathbf{x}_{-i,k}|\boldsymbol{\lambda})}{p(\mathbf{x}_{-i,k}|\boldsymbol{\lambda})} \tag{25.36}$$

where

$$p(\mathbf{x}_i, \mathbf{x}_{-i,k}|\boldsymbol{\lambda}) \quad = \quad \int p(\mathbf{x}_i|\boldsymbol{\theta}_k)\left[\prod_{j\neq i:z_j=k} p(\mathbf{x}_j|\boldsymbol{\theta}_k)\right] H(\boldsymbol{\theta}_k|\boldsymbol{\lambda})d\boldsymbol{\theta}_k \tag{25.37}$$

is the marginal likelihood of all the data assigned to cluster k, including i, and $p(\mathbf{x}_{-i,k}|\boldsymbol{\lambda})$ is an analogous expression excluding i. Thus we see that the term $p(\mathbf{x}_i|\mathbf{x}_{-i}, \mathbf{z}_{-i}, z_i = k, \boldsymbol{\lambda})$ is the posterior preditive distribution for cluster k evaluated at $\mathbf{x}_i$.

If $z_i = k^*$, corresponding to a new cluster, we have

$$p(\mathbf{x}_i|\mathbf{x}_{-i}, \mathbf{z}_{-i}, z_i = k^*, \boldsymbol{\lambda}) = p(\mathbf{x}_i|\boldsymbol{\lambda}) = \int p(\mathbf{x}_i|\boldsymbol{\theta})H(\boldsymbol{\theta}|\boldsymbol{\lambda})d\boldsymbol{\theta} \tag{25.38}$$

which is just the prior predictive distribution for a new cluster evaluated at $\mathbf{x}_i$.

See Algorithm 1 for the pseudocode. (This is called "Algorithm 3" in (Neal 2000).) This is very similar to collapsed Gibbs for finite mixtures except that we have to consider the case $z_i = k^*$.

An example of this procedure in action is shown in Figure 25.7. The sample clusterings, and the induced posterior over K, seems reasonable. The method tends to rapidly discover a good clustering. By contrast, Gibbs sampling (and EM) for a finite mixture model often gets stuck in

Algorithm 25.1: Collapsed Gibbs sampler for DP mixtures

```
1  for each $i = 1 : N$ in random order do
2      Remove $\mathbf{x}_i$'s sufficient statistics from old cluster $z_i$ ;
3      for each $k = 1 : K$ do
4          Compute $p_k(\mathbf{x}_i) = p(\mathbf{x}_i|\mathbf{x}_{-i}(k))$;
5          Set $N_{k,-i} = \dim(\mathbf{x}_{-i}(k))$ ;
6          Compute $p(z_i = k|\mathbf{z}_{-i}, \mathcal{D}) = \frac{N_{k,-i}}{\alpha+N-1}$;
7      Compute $p_*(\mathbf{x}_i) = p(\mathbf{x}_i|\boldsymbol{\lambda})$;
8      Compute $p(z_i = *|\mathbf{z}_{-i}, \mathcal{D}) = \frac{\alpha}{\alpha+N-1}$;
9      Normalize $p(z_i|\cdot)$;
10     Sample $z_i \sim p(z_i|\cdot)$ ;
11     Add $\mathbf{x}_i$'s sufficient statistics to new cluster $z_i$ ;
12     If any cluster is empty, remove it and decrease $K$;
```

poor local optima (not shown). This is because the DPMM is able to create extra redundant clusters early on, and to use them to escape local optima. Figure 25.8 shows that most of the time, the DPMM converges more rapidly than a finite mixture model.

A variety of other fitting methods have been proposed. (Daume 2007a) shows how one can use A star search and beam search to quickly find an approximate MAP estimate. (Mansinghka et al. 2007) discusses how to fit a DPMM online using particle filtering, which is a like a stochastic version of beam search. This can be more efficient than Gibbs sampling, particularly for large datasets. (Kurihara et al. 2006) develops a variational approximation that is even faster (see also (Zobay 2009)). Extensions to the case of non-conjugate priors are discussed in (Neal 2000).

Another important issue is how to set the hyper-parameters. For the DP, the value of α does not have much impact on predictive accuracy, but it does affect the number of clusters. One approach is to put a $\text{Ga}(a, b)$ prior for α, and then to from its posterior, $p(\alpha|K, N, a, b)$, using auxiliary variable methods (Escobar and West 1995). Alternatively, one can use empirical Bayes (McAuliffe et al. 2006). Similarly, for the base distribution, we can either sample the hyper-parameters $\boldsymbol{\lambda}$ (Rasmussen 2000) or use empirical Bayes (McAuliffe et al. 2006).

25.3 Affinity propagation

Mixture models, whether finite or infinite, require access to the raw $N \times D$ data matrix, and need to specify a generative model of the data. An alternative approach takes as input an $N \times N$ similarity matrix, and then tries to identify examplars, which will act as cluster centers. The K-medoids or **K-centers** algorithm (Section 14.4.2) is one approach, but it can suffer from local minima. Here we describe an alternative approach called **affinity propagation** (Frey and Dueck 2007) that works substantially better in practice.

The idea is that each data point must choose another data point as its exemplar or centroid; some data points will choose themselves as centroids, and this will automatically determine the number of clusters. More precisely, let $c_i \in \{1, \ldots, N\}$ represent the centroid for datapoint i.

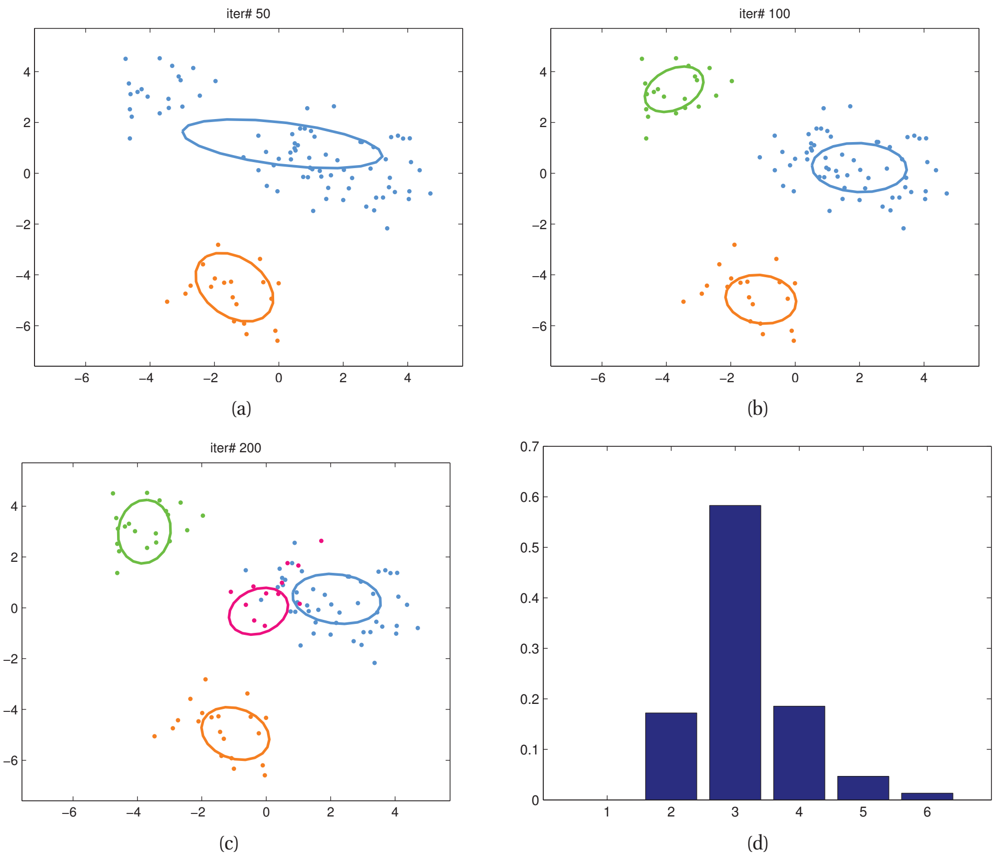


Figure 25.7 100 data points in 2d are clustered using a DP mixture fit with collapsed Gibbs sampling. We show samples from the posterior after 50,100, 200 samples. We also show the posterior over K, based on 200 samples, discarding the first 50 as burnin. Figure generated by `dpmGauss2dDemo`, written by Yee Whye Teh.

The goal is to maximize the following function

$$S(\mathbf{c}) = \sum_{i=1}^{N} s(i, c_i) + \sum_{k=1}^{N} \delta_k(\mathbf{c}) \tag{25.39}$$

The first term measures the similarity of each point to its centroid. The second term is a penalty term that is $-\infty$ if some data point i has chosen k as its exemplar (i.e., $c_i = k$), but k has not chosen itself as an exemplar (i.e., we do not have $c_k = k$). More formally,

$$\delta_k(\mathbf{c}) = \begin{cases} -\infty & \text{if } c_k \neq k \text{ but } \exists i : c_i = k \\ 0 & \text{otherwise} \end{cases} \tag{25.40}$$

The objective function can be represented as a factor graph. We can either use N nodes,

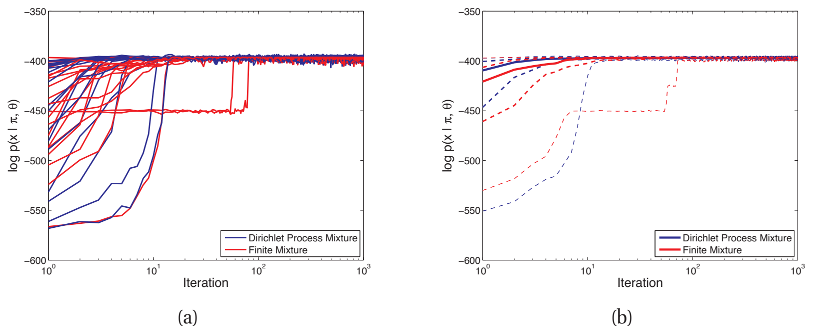


Figure 25.8 Comparison of collapsed Gibbs samplers for a DP mixture (dark blue) and a finite mixture (light red) with $K = 4$ applied to $N = 300$ data points (shown in Figure 25.7). Left: logprob vs iteration for 20 different starting values. Right: median (thick line) and quantiles (dashed lines) over 100 different starting values. Source: Figure 2.27 of (Sudderth 2006). Used with kind permission of Erik Sudderth.

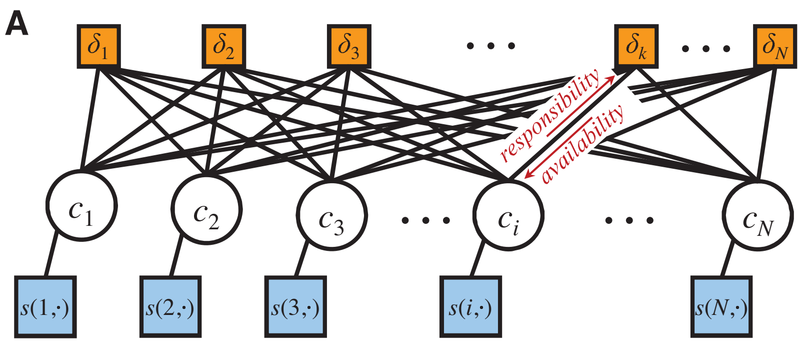


Figure 25.9 Factor graphs for affinity propagation. Circles are variables, squares are factors. Each c_i node has N possible states. From Figure S2 of (Frey and Dueck 2007). Used with kind permission of Brendan Frey.

each with N possible values, as shown in Figure 25.9, or we can use N^2 binary nodes (see (Givoni and Frey 2009) for the details). We will assume the former representation.

We can find a strong local maximum of the objective by using max-product loopy belief propagation (Section 22.2). Referring to the model in Figure 25.9, each variable nodes c_i sends a message to each factor node δ_k. It turns out that this vector of N numbers can be reduced to a scalar message, denote $r_{i \to k}$, known as the responsibility. This is a measure of how much i thinks k would make a good exemplar, compared to all the other exemplars i has looked at. In addition, each factor node δ_k sends a message to each variable node c_i. Again this can be reduced to a scalar message, $a_{i \leftarrow k}$, known as the availability. This is a measure of how strongly k believes it should an exemplar for i, based on all the other data points k has looked at.

As usual with loopy BP, the method might oscillate, and convergence is not guaranteed.

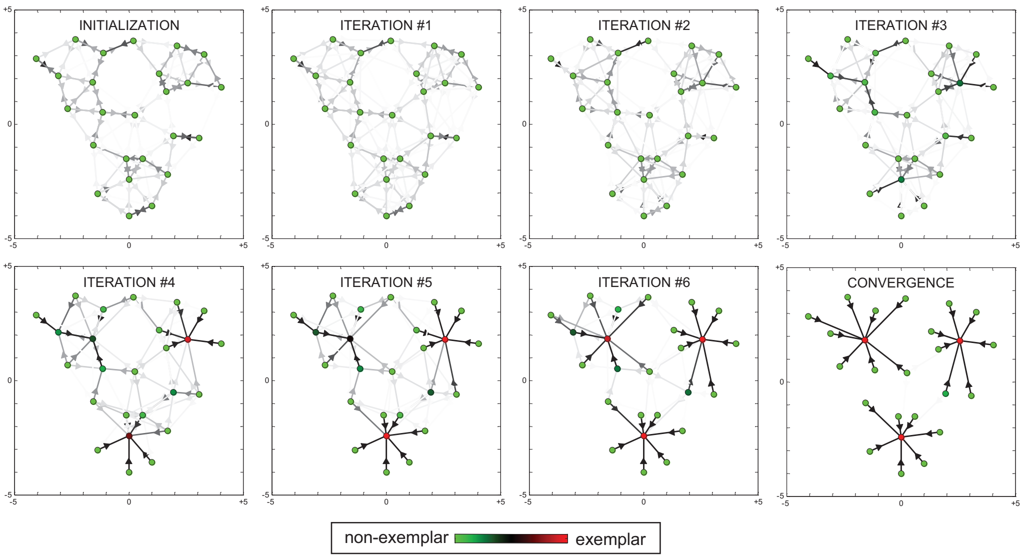


Figure 25.10 Example of affinity propagation. Each point is colored coded by how much it wants to be an exemplar (red is the most, green is the least). This can be computed by summing up all the incoming availability messages and the self-similarity term. The darkness of the $i \to k$ arrow reflects how much point i wants to belong to exemplar k. From Figure 1 of (Frey and Dueck 2007). Used with kind permission of Brendan Frey.

However, by using damping, the method is very reliable in practice. If the graph is densely connected, message passing takes $O(N^2)$ time, but with sparse similarity matrices, it only takes $O(E)$ time, where E is the number of edges or non-zero entries in $\mathbf{S}$.

The number of clusters can be controlled by scaling the diagonal terms $S(i,i)$, which reflect how much each data point wants to be an exemplar. Figure 25.10 gives a simple example of some 2d data, where the negative Euclidean distance was used to measured similarity. The $S(i,i)$ values were set to be the median of all the pairwise similarities. The result is 3 clusters. Many other results are reported in (Frey and Dueck 2007), who show that the method significantly outperforms K-medoids.

25.4 Spectral clustering

An alternative view of clustering is in terms of **graph cuts**. The idea is we create a weighted undirected graph $\mathbf{W}$ from the similarity matrix $\mathbf{S}$, typically by using the nearest neighbors of each point; this ensures the graph is sparse, which speeds computation. If we want to find a partition into K clusters, say $A_1, \ldots, A_K$, one natural criterion is to minimize

$$\text{cut}(A_1, \ldots, A_K) \triangleq \frac{1}{2} \sum_{k=1}^{K} W(A_k, \overline{A}_k) \tag{25.41}$$

where $\overline{A}_k = V \setminus A_k$ is the complement of A_k, and $W(A, B) \triangleq \sum_{i \in A, j \in B} w_{ij}$. For $K = 2$ this problem is easy to solve. Unfortunately the optimal solution often just partitions off a single data point from the rest. To ensure the sets are reasonably large, we can define the **normalized cut** to be

$$\text{Ncut}(A_1, \ldots, A_K) \triangleq \frac{1}{2} \sum_{k=1}^{K} \frac{\text{cut}(A_k, \overline{A}_k)}{\text{vol}(A_k)} \tag{25.42}$$

where $\text{vol}(A) \triangleq \sum_{i \in A} d_i$, and $d_i = \sum_{j=1}^{N} w_{ij}$ is the weighted degree of node i. This splits the graph into K clusters such that nodes within each cluster are similar to each other, but are different to nodes in other clusters.

We can formulate the Ncut problem in terms of searching for binary vectors $\mathbf{c}_i \in \{0, 1\}^N$, where $c_{ik} = 1$ if point i belongs to cluster k, that minimize the objective. Unfortunately this is NP-hard (Wagner and Wagner 1993). Affinity propagation is one way to solve the problem. Another is to relax the constraints that $\mathbf{c}_i$ be binary, and allow them to be real-valued. The result turns into an eigenvector problem known as **spectral clustering** (see e.g., (Shi and Malik 2000)). In general, the technique of performing eigenalysis of graphs is called **spectral graph theory** (Chung 1997).

Going into the details would take us too far afield, but below we give a very brief summary, based on (von Luxburg 2007), since we will encounter some of these ideas later on.

25.4.1 Graph Laplacian

Let $\mathbf{W}$ be a symmetric weight matrix for a graph, where $w_{ij} = w_{ji} \geq 0$. Let $\mathbf{D} = \text{diag}(d_i)$ be a diaogonal matrix containing the weighted degree of each node. We define the **graph Laplacian** as follows:

$$\mathbf{L} \triangleq \mathbf{D} - \mathbf{W} \tag{25.43}$$

This matrix has various important properties. Because each row sums to zero, we have that $\mathbf{1}$ is an eigenvector with eigenvalue 0. Furthermore, the matrix is symmetric and positive semi-definite. To see this, note that

$$\mathbf{f}^T\mathbf{L}\mathbf{f} = \mathbf{f}^T\mathbf{D}\mathbf{f} - \mathbf{f}^T\mathbf{W}\mathbf{f} = \sum_i d_i f_i^2 - \sum_{i,j} f_i f_j w_{ij} \tag{25.44}$$

$$= \frac{1}{2}\left(\sum_i d_i f_i^2 - 2\sum_{i,j} f_i f_j w_{ij} + \sum_j d_j f_j^2\right) = \frac{1}{2}\sum_{i,j} w_{ij}(f_i - f_j)^2 \tag{25.45}$$

Hence $\mathbf{f}^T\mathbf{L}\mathbf{f} \geq 0$ for all $\mathbf{f} \in \mathbb{R}^N$. Consequently we see that $\mathbf{L}$ has N non-negative, real-valued eigenvalues, $0 \leq \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_N$.

To get some intuition as to why $\mathbf{L}$ might be useful for graph-based clustering, we note the following result.

Theorem 25.4.1. *The set of eigenvectors of* $\mathbf{L}$ *with eigenvalue 0 is spanned by the indicator vectors* $\mathbf{1}_{A_1}, \ldots, \mathbf{1}_{A_K}$*, where* A_k *are the* K *connected components of the graph.*

Proof. Let us start with the case $K = 1$. If $\mathbf{f}$ is an eigenvector with eigenvalue 0, then $0 = \sum_{ij} w_{ij}(f_i - f_j)^2$. If two nodes are connected, so $w_{ij} > 0$, we must have that $f_i = f_j$. Hence $\mathbf{f}$ is constant for all vertices which are connected by a path in the graph. Now suppose $K > 1$. In this case, $\mathbf{L}$ will be block diagonal. A similar argument to the above shows that we will have K indicator functions, which "select out" the connected components. $\square$

This suggests the following algorithm. Compute the first K eigenvectors $\mathbf{u}_k$ of $\mathbf{L}$. Let $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_K]$ be an $N \times K$ matrix with the eigenvectors in its columns. Let $\mathbf{y}_i \in \mathbb{R}^K$ be the i'th row of $\mathbf{U}$. Since these $\mathbf{y}_i$ will be piecewise constant, we can apply K-means clustering to them to recover the connected components. Now assign point i to cluster k iff row i of $\mathbf{Y}$ was assigned to cluster k.

In reality, we do not expect a graph derived from a real similarity matrix to have isolated connected components — that would be too easy. But it is reasonable to suppose the graph is a small "perturbation" from such an ideal. In this case, one can use results from perturbation theory to show that the eigenvectors of the perturbed Laplacian will be close to these ideal indicator functions (Ng et al. 2001).

Note that this approach is related to kernel PCA (Section 14.4.4). In particular, KPCA uses the largest eigenvectors of $\mathbf{W}$; these are equivalent to the smallest eigenvectors of $\mathbf{I} - \mathbf{W}$. This is similar to the above method, which computes the smallest eigenvectors of $\mathbf{L} = \mathbf{D} - \mathbf{W}$. See (Bengio et al. 2004) for details. In practice, spectral clustering gives much better results than KPCA.

25.4.2 Normalized graph Laplacian

In practice, it is important to normalize the graph Laplacian, to account for the fact that some nodes are more highly connected than others. There are two comon ways to do this. One method, used in e.g., (Shi and Malik 2000; Meila 2001), creates a stochastic matrix where each row sums to one:

$$\mathbf{L}_{rw} \triangleq \mathbf{D}^{-1}\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{W} \tag{25.46}$$

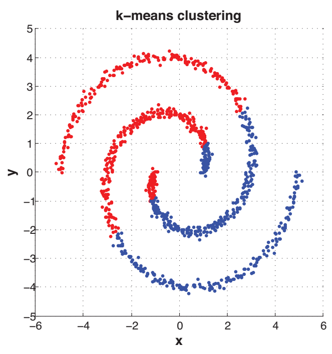
The eigenvalues and eigenvectors of $\mathbf{L}$ and $\mathbf{L}_{rw}$ are closely related to each other (see (von Luxburg 2007) for details). Furthemore, one can show that for $\mathbf{L}_{rw}$, the eigenspace of 0 is again spanned by the indicator vectors $\mathbf{1}_{A_k}$. This suggests the following algorithm: find the smallest K eigenvectors of $\mathbf{L}_{rw}$, create $\mathbf{U}$, cluster the rows of $\mathbf{U}$ using K-means, then infer the partitioning of the original points (Shi and Malik 2000). (Note that the eigenvectors/ values of $\mathbf{L}_{rw}$ are equivalent to the generalized eigenvectors/ values of $\mathbf{L}$, which solve $\mathbf{Lu} = \lambda\mathbf{DU}$.)

Another method, used in e.g., (Ng et al. 2001), creates a symmetric matrix

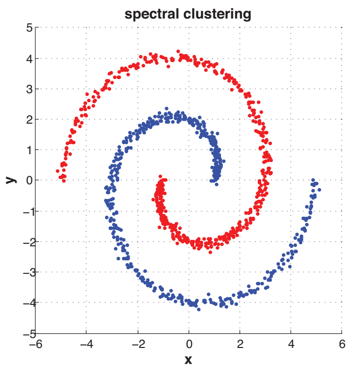
$$\mathbf{L}_{sym} \triangleq \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}} \tag{25.47}$$

This time the eigenspace of 0 is spanned by $\mathbf{D}^{\frac{1}{2}}\mathbf{1}_{A_k}$. This suggest the following algorithm: find the smallest K eigenvectors of $\mathbf{L}_{sym}$, create $\mathbf{U}$, normalize each row to unit norm by creating $t_{ij} = u_{ij}/\sqrt{(\sum_k u_{ik}^2)}$, cluster the rows of $\mathbf{T}$ using K-means, then infer the partitioning of the original points (Ng et al. 2001).

There is an interesting connection between Ncuts and random walks on a graph (Meila 2001). First note that $\mathbf{P} = \mathbf{D}^{-1}\mathbf{W} = \mathbf{I} - \mathbf{L}_{rw}$ is a stochastic matrix, where $p_{ij} = w_{ij}/d_i$



(a)



(b)

Figure 25.11 Clustering data consisting of 2 spirals. (a) K-means. (b) Spectral clustering. Figure generated by `spectralClusteringDemo`, written by Wei-Lwun Lu.

can be interpreted as the probability of going from i to j. If the graph is connected and non-bipartite, it possesses a unique stationary distribution $\boldsymbol{\pi} = (\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_N)$, where $\pi_i = d_i/\text{vol}(V)$. Furthermore, one can show that

$$\text{Ncut}(A, \overline{A}) = p(\overline{A}|A) + p(A|\overline{A}) \tag{25.48}$$

This means that we are looking for a cut such that a random walk rarely makes transitions from A to $\overline{A}$ or vice versa.

25.4.3 Example

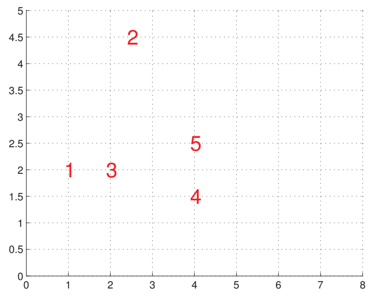
Figure 25.11 illustrates the method in action. In Figure 25.11(a), we see that K-means does a poor job of clustering, since it implicitly assumes each cluster corresponds to a spherical Gaussian. Next we try spectral clustering. We define a similarity matrix using the Gaussian kernel. We compute the first two eigenvectors of the Laplacian. From this we can infer the clustering in Figure 25.11(b).

Since the method is based on finding the smallest K eigenvectors of a sparse matrix, it takes $O(N^3)$ time. However, a variety of methods can be used to scale it up for large datasets (see e.g., (Yan et al. 2009)).

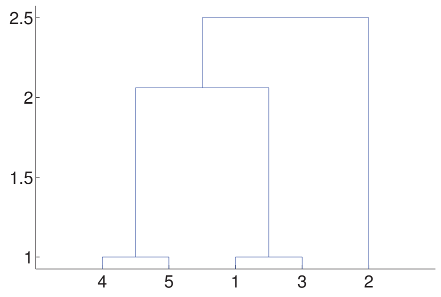
25.5 Hierarchical clustering

Mixture models, whether finite or infinite, produce a "flat" clustering. Often we want to learn a **hierarchical clustering**, where clusters can be nested inside each other.

There are two main approaches to hierarchical clustering: bottom-up or **agglomerative clustering**, and top-down or **divisive clustering**. Both methods take as input a dissimilarity matrix between the objects. In the bottom-up approach, the most similar groups are merged at each

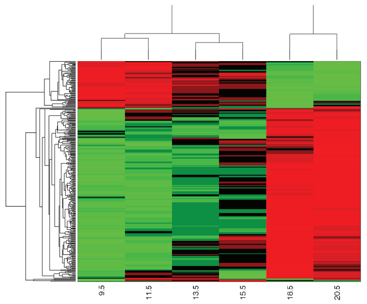


(a)

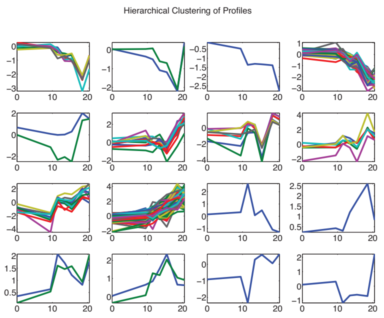


(b)

Figure 25.12 (a) An example of single link clustering using city block distance. Pairs (1,3) and (4,5) are both distance 1 apart, so get merged first. (b) The resulting dendrogram. Based on Figure 7.5 of (Alpaydin 2004). Figure generated by `agglomDemo`.



(a)



(b)

Figure 25.13 Hierarchical clustering applied to the yeast gene expression data. (a) The rows are permuted according to a hierarchical clustering scheme (average link agglomerative clustering), in order to bring similar rows close together. (b) 16 clusters induced by cutting the average linkage tree at a certain height. Figure generated by `hclustYeastDemo`.

step. In the top-down approach, groups are split using various different criteria. We give the details below.

Note that agglomerative and divisive clustering are both just heuristics, which do not optimize any well-defined objective function. Thus it is hard to assess the quality of the clustering they produce in any formal sense. Furthermore, they will always produce a clustering of the input data, even if the data has no structure at all (e.g., it is random noise). Later in this section we will discuss a probabilistic version of hierarchical clustering that solves both these problems.

Algorithm 25.2: Agglomerative clustering

```
1  initialize clusters as singletons: for $i \leftarrow 1$ to $n$ do $C_i \leftarrow \{i\}$;
2  initialize set of clusters available for merging: $S \leftarrow \{1, \ldots, n\}$;
3  repeat
4      Pick 2 most similar clusters to merge: $(j, k) \leftarrow \arg\min_{j,k \in S} d_{j,k}$;
5      Create new cluster $C_\ell \leftarrow C_j \cup C_k$;
6      Mark $j$ and $k$ as unavailable: $S \leftarrow S \setminus \{j, k\}$;
7      if $C_\ell \neq \{1, \ldots, n\}$ then
8          Mark $\ell$ as available, $S \leftarrow S \cup \{\ell\}$;
9      foreach $i \in S$ do
10         Update dissimilarity matrix $d(i, \ell)$;
11 until no more clusters are available for merging;
```

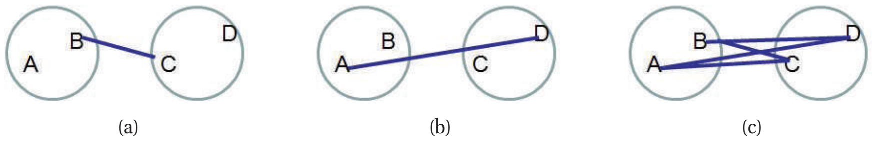


Figure 25.14 Illustration of (a) Single linkage. (b) Complete linkage. (c) Average linkage.

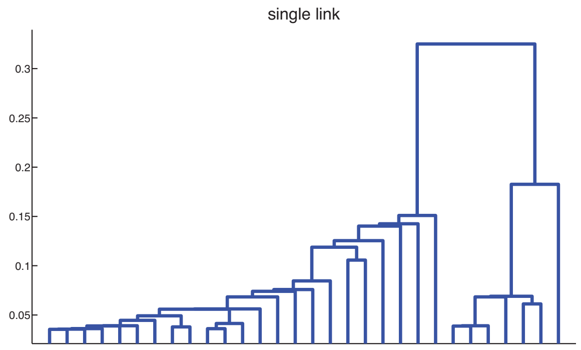
25.5.1 Agglomerative clustering

Agglomerative clustering starts with N groups, each initially containing one object, and then at each step it merges the two most similar groups until there is a single group, containing all the data. See Algorithm 11 for the pseudocode. Since picking the two most similar clusters to merge takes $O(N^2)$ time, and there are $O(N)$ steps in the algorithm, the total running time is $O(N^3)$. However, by using a priority queue, this can be reduced to $O(N^2 \log N)$ (see e.g., (Manning et al. 2008, ch. 17) for details). For large N, a common heuristic is to first run K-means, which takes $O(KND)$ time, and then apply hierarchical clustering to the estimated cluster centers.

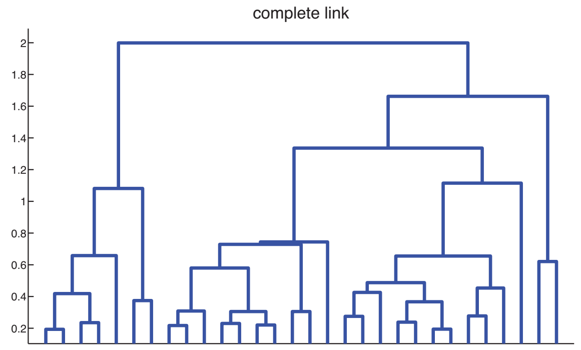
The merging process can be represented by a **binary tree**, called a **dendrogram**, as shown in Figure 25.12(b). The initial groups (objects) are at the leaves (at the bottom of the figure), and every time two groups are merged, we join them in the tree. The height of the branches represents the dissimilarity between the groups that are being joined. The root of the tree (which is at the top) represents a group containing all the data. If we cut the tree at any given height, we induce a clustering of a given size. For example, if we cut the tree in Figure 25.12(b) at height 2, we get the clustering $\{\{\{4, 5\}, \{1, 3\}\}, \{2\}\}$. We discuss the issue of how to choose the height/ number of clusters below.

A more complex example is shown in Figure 25.13(a), where we show some gene expression data. If we cut the tree in Figure 25.13(a) at a certain height, we get the 16 clusters shown in Figure 25.13(b).

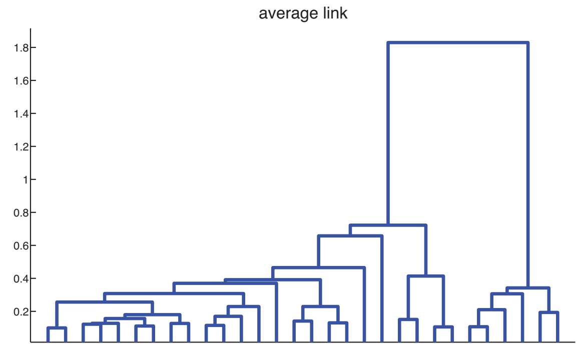
There are actually three variants of agglomerative clustering, depending on how we define the dissimilarity between groups of objects. These can give quite different results, as shown in



(a)



(b)



(c)

Figure 25.15 Hierarchical clustering of yeast gene expression data. (a) Single linkage. (b) Complete linkage. (c) Average linkage. Figure generated by `hclustYeastDemo`.

Figure 25.15. We give the details below.

25.5.1.1 Single link

In **single link clustering**, also called **nearest neighbor clustering**, the distance between two groups G and H is defined as the distance between the two closest members of each group:

$$d_{SL}(G, H) = \min_{i \in G, i' \in H} d_{i,i'} \tag{25.49}$$

See Figure 25.14(a).

The tree built using single link clustering is a minimum spanning tree of the data, which is a tree that connects all the objects in a way that minimizes the sum of the edge weights (distances). To see this, note that when we merge two clusters, we connect together the two closest members of the clusters; this adds an edge between the corresponding nodes, and this is guaranteed to be the "lightest weight" edge joining these two clusters. And once two clusters have been merged, they will never be considered again, so we cannot create cycles. As a consequence of this, we can actually implement single link clustering in $O(N^2)$ time, whereas the other variants take $O(N^3)$ time.

25.5.1.2 Complete link

In **complete link clustering**, also called **furthest neighbor clustering**, the distance between two groups is defined as the distance between the two most distant pairs:

$$d_{CL}(G, H) = \max_{i \in G, i' \in H} d_{i,i'} \tag{25.50}$$

See Figure 25.14(b).

Single linkage only requires that a single pair of objects be close for the two groups to be considered close together, regardless of the similarity of the other members of the group. Thus clusters can be formed that violate the **compactness** property, which says that all the observations within a group should be similar to each other. In particular if we define the **diameter** of a group as the largest dissimilarity of its members, $d_G = \max_{i \in G, i' \in G} d_{i,i'}$, then we can see that single linkage can produce clusters with large diameters. Complete linkage represents the opposite extreme: two groups are considered close only if all of the observations in their union are relatively similar. This will tend to produce clusterings with small diameter, i.e., compact clusters.

25.5.1.3 Average link

In practice, the preferred method is **average link clustering**, which measures the average distance between all pairs:

$$d_{avg}(G, H) = \frac{1}{n_G n_H} \sum_{i \in G} \sum_{i' \in H} d_{i,i'} \tag{25.51}$$

where n_G and n_H are the number of elements in groups G and H. See Figure 25.14(c).

Average link clustering represents a compromise between single and complete link clustering. It tends to produce relatively compact clusters that are relatively far apart. However, since it involves averaging of the $d_{i,i'}$'s, any change to the measurement scale can change the result. In contrast, single linkage and complete linkage are invariant to monotonic transformations of $d_{i,i'}$, since they leave the relative ordering the same.

25.5.2 Divisive clustering

Divisive clustering starts with all the data in a single cluster, and then recursively divides each cluster into two daughter clusters, in a top-down fashion. Since there are $2^{N-1} - 1$ ways to split a group of N items into 2 groups, it is hard to compute the optimal split, so various heuristics are used. One approach is pick the cluster with the largest diameter, and split it in two using the K-means or K-medoids algorithm with $K = 2$. This is called the **bisecting K-means** algorithm (Steinbach et al. 2000). We can repeat this until we have any desired number of clusters. This can be used as an alternative to regular K-means, but it also induces a hierarchical clustering.

Another method is to build a minimum spanning tree from the dissimilarity graph, and then to make new clusters by breaking the link corresponding to the largest dissimilarity. (This actually gives the same results as single link agglomerative clustering.)

Yet another method, called **dissimilarity analysis** (Macnaughton-Smith et al. 1964), is as follows. We start with a single cluster containing all the data, $G = \{1, \ldots, N\}$. We then measure the average dissimilarity of $i \in G$ to all the other $i' \in G$:

$$d_i^G = \frac{1}{n_G} \sum_{i' \in G} d_{i,i'} \tag{25.52}$$

We remove the most dissimilar object and put it in its own cluster H:

$$i^* = \arg\max_{i \in G} d_i^G, \quad G = G \setminus \{i^*\}, \quad H = \{i^*\} \tag{25.53}$$

We now continue to move objects from G to H until some stopping criterion is met. Specifically, we pick a point i^* to move that maximizes the average dissimilarity to each $i' \in G$ but minimizes the average dissimilarity to each $i' \in H$:

$$d_i^H = \frac{1}{n_H} \sum_{i' \in H} d_{i,i'}, \quad i^* = \arg\max_{i \in G} d_i^G - d_i^H \tag{25.54}$$

We continue to do this until $d_i^G - d_i^H$ is negative. The final result is that we have split G into two daughter clusters, G and H. We can then recursively call the algorithm on G and/or H, or on any other node in the tree. For example, we might choose to split the node G whose average dissimilarity is highest, or whose maximum dissimilarity (i.e., diameter) is highest. We continue the process until the average dissimilarity within each cluster is below some threshold, and/or all clusters are singletons.

Divisive clustering is less popular than agglomerative clustering, but it has two advantages. First, it can be faster, since if we only split for a constant number of levels, it takes just $O(N)$ time. Second, the splitting decisions are made in the context of seeing all the data, whereas bottom-up methods make myopic merge decisions.

25.5.3 Choosing the number of clusters

It is difficult to choose the “right” number of clusters, since a hierarchical clustering algorithm will always create a hierarchy, even if the data is completely random. But, as with choosing K for K-means, there is the hope that there will be a visible “gap” in the lengths of the links in the dendrogram (represent the dissimilarity between merged groups) between natural clusters and unnatural clusters. Of course, on real data, this gap might be hard to detect. In Section 25.5.4, we will present a Bayesian approach to hierarchical clustering that nicely solves this problem.

25.5.4 Bayesian hierarchical clustering

There are several ways to make probabilistic models which produce results similar to hierarchical clustering, e.g., (Williams 2000; Neal 2003b; Castro et al. 2004; Lau and Green 2006). Here we present one particular approach called **Bayesian hierarchical clustering** (Heller and Ghahramani 2005). Algorithmically it is very similar to standard bottom-up agglomerative clustering, and takes comparable time, whereas several of the other techniques referenced above are much slower. However, it uses Bayesian hypothesis tests to decide which clusters to merge (if any), rather than computing the similarity between groups of points in some ad-hoc way. These hypothesis tests are closely related to the calculations required to do inference in a Dirichlet process mixture model, as we will see. Furthermore, the input to the model is a data matrix, not a dissimilarity matrix.

25.5.4.1 The algorithm

Let $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ represent all the data, and let $\mathcal{D}_i$ be the set of datapoints at the leaves of the substree T_i. At each step, we compare two trees T_i and T_j to see if they should be merged into a new tree. Define $\mathcal{D}_{ij}$ as their merged data, and let $M_{ij} = 1$ if they should be merged, and $M_{ij} = 0$ otherwise.

The probability of a merge is given by

$$r_{ij} \triangleq \frac{p(\mathcal{D}_{ij}|M_{ij}=1)p(M_{ij}=1)}{p(\mathcal{D}_{ij}|T_{ij})} \tag{25.55}$$

$$p(\mathcal{D}_{ij}|T_{ij}) = p(\mathcal{D}_{ij}|M_{ij}=1)p(M_{ij}=1) + p(\mathcal{D}_{ij}|M_{ij}=0)p(M_{ij}=0) \tag{25.56}$$

Here $p(M_{ij} = 1)$ is the prior probability of a merge, which can be computed using a bottom-up algorithm described below. We now turn to the likelihood terms. If $M_{ij} = 1$, the data in $\mathcal{D}_{ij}$ is assumed to come from the same model, and hence

$$p(\mathcal{D}_{ij}|M_{ij}=1) = \int \left[\prod_{\mathbf{x}_n \in \mathcal{D}_{ij}} p(\mathbf{x}_n|\boldsymbol{\theta}) \right] p(\boldsymbol{\theta}|\lambda) d\boldsymbol{\theta} \tag{25.57}$$

If $M_{ij} = 0$, the data in $\mathcal{D}_{ij}$ is assumed to have been generated by each tree independently, so

$$p(\mathcal{D}_{ij}|M_{ij}=0) = p(\mathcal{D}_i|T_i)p(\mathcal{D}_j|T_j) \tag{25.58}$$

These two terms will have already been computed by the bottom-up process. Consequently we have all the quantities we need to decide which trees to merge. See Algorithm 9 for the pseudocode, assuming $p(M_{ij})$ is uniform. When finished, we can cut the tree at points where $r_{ij} < 0.5$.

Algorithm 25.3: Bayesian hierarchical clustering

```
1 Initialize $\mathcal{D}_i = \{\mathbf{x}_i\}$, $i = 1 : N$ ;
2 Compute $p(\mathcal{D}_i|T_i)$, $i = 1 : N$ ;
3 repeat
4     for each pair of clusters $i$, $j$ do
5         Compute $p(\mathcal{D}_{ij}|T_{ij})$
6     Find the pair $\mathcal{D}_i$ and $\mathcal{D}_j$ with highest merge probability $r_{ij}$;
7     Merge $\mathcal{D}_k := \mathcal{D}_i \cup \mathcal{D}_j$;
8     Delete $\mathcal{D}_i$, $\mathcal{D}_j$ ;
9 until all clusters merged;
```

25.5.4.2 The connection with Dirichlet process mixture models

In this section, we will establish the connection between BHC and DPMMs. This will in turn give us an algorithm to compute the prior probabilities $p(M_{ij} = 1)$.

Note that the marginal likelihood of a DPMM, summing over all $2^N - 1$ partitions, is given by

$$p(\mathcal{D}_k) = \sum_{v \in \mathcal{V}} p(v) p(\mathcal{D}_v) \tag{25.59}$$

$$p(v) = \frac{\alpha^{m_v} \prod_{l=1}^{m_v} \Gamma(n_l^v)}{\frac{\Gamma(n_k+\alpha)}{\Gamma(\alpha)}} \tag{25.60}$$

$$p(\mathcal{D}_v) = \prod_{l=1}^{m_v} p(\mathcal{D}_l^v) \tag{25.61}$$

where $\mathcal{V}$ is the set of all possible partitions of $\mathcal{D}_k$, $p(v)$ is the probability of partition v, m_v is the number of clusters in partition v, n_l^v is the number of points in cluster l of partition v, $\mathcal{D}_l^v$ are the points in cluster l of partition v, and n_k are the number of points in $\mathcal{D}_k$.

One can show (Heller and Ghahramani 2005) that $p(\mathcal{D}_k|T_k)$ computed by the BHC algorithm is similar to $p(\mathcal{D}_k)$ given above, except for the fact that it only sums over partitions which are consistent with tree T_k. (The number of tree-consistent partitions is exponential in the number of data points for balanced binary trees, but this is obviously a subset of all possible partitions.) In this way, we can use the BHC algorithm to compute a lower bound on the marginal likelihood of the data from a DPMM. Furthermore, we can interpret the algorithm as greedily searching through the exponentially large space of tree-consistent partitions to find the best ones of a given size at each step.

We are now in a position to compute $\pi_k = p(M_k = 1)$, for each node k with children i and j. This is equal to the probability of cluster $\mathcal{D}_k$ coming from the DPMM, relative to all other partitions of $\mathcal{D}_k$ consistent with the current tree. This can be computed as follows: initialize $d_i = \alpha$ and $\pi_i = 1$ for each leaf i; then as we build the tree, for each internal node k, compute $d_k = \alpha\Gamma(n_k) + d_i d_j$, and $\pi_k = \frac{\alpha\Gamma(n_k)}{d_k}$, where i and j are k's left and right children.

Data Set	Single Linkage	Complete Linkage	Average Linkage	BHC
Synthetic	0.599 ± 0.033	0.634 ± 0.024	0.668 ± 0.040	**0.828 ± 0.025**
Newsgroups	0.275 ± 0.001	0.315 ± 0.008	0.282 ± 0.002	**0.465 ± 0.016**
Spambase	0.598 ± 0.017	0.699 ± 0.017	0.668 ± 0.019	**0.728 ± 0.029**
Digits	0.224 ± 0.004	0.299 ± 0.006	0.342 ± 0.005	**0.393 ± 0.015**
Fglass	0.478 ± 0.009	0.476 ± 0.009	**0.491 ± 0.009**	0.467 ± 0.011

Table 25.1 Purity scores for various hierarchical clustering schemes applied to various data sets. The synthetic data has $N = 200, D = 2, C = 4$ and real features. Newsgroups is extracted from the 20 newsgroups dataset ($D = 500, N = 800, C = 4$, binary features). Spambase has $N = 100, C = 2, D = 57$, binary features. Digits is the CEDAR Buffalo digits ($N = 200, C = 10, D = 64$, binarized features). Fglass is forensic glass dataset ($N = 214, C = 6, D = 9$, real features). Source: Table 1 of (Heller and Ghahramani 2005). Used with kind permission of Katherine Heller.

25.5.4.3 Learning the hyper-parameters

The model has two free-parameters: α and λ, where λ are the hyper-parameters for the prior on the parameters $\boldsymbol{\theta}$. In (Heller and Ghahramani 2005), they show how one can back-propagate gradients of the form $\frac{\partial p(\mathcal{D}_k|T_k)}{\partial \lambda}$ through the tree, and thus perform an empirical Bayes estimate of the hyper-parameters.

25.5.4.4 Experimental results

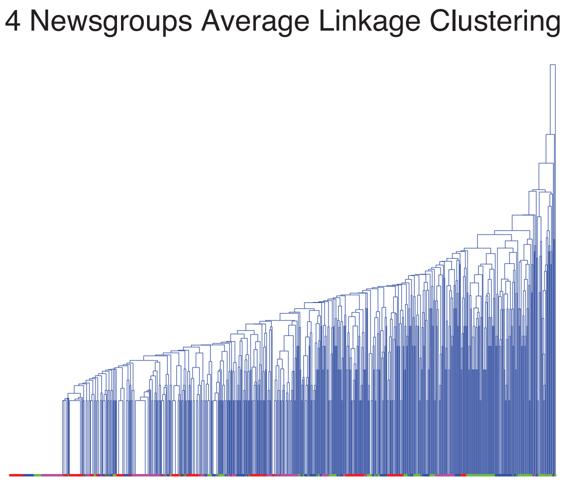
(Heller and Ghahramani 2005) compared BHC with traditional agglomerative clustering algorithms on various data sets in terms of purity scores. The results are shown in Table 25.1. We see that BHC did much better than the other methods on all datasets except the forensic glass one.

Figure 25.16 visualizes the tree structure estimated by BHC and agglomerative hierarchical clustering (AHC) on the newsgroup data (using a beta-Bernoulli model). The BHC tree is clearly superior (look at the colors at the leaves, which represent class labels). Figure 25.17 is a zoom-in on the top few nodes of these two trees. BHC splits off clusters concerning sports from clusters concerning cars and space. AHC keeps sports and cars merged together. Although sports and cars both fall under the same "rec" newsgroup heading (as opposed to space, that comes under the "sci" newsgroup heading), the BHC clustering still seems more reasonable, and this is borne out by the quantitative purity scores.

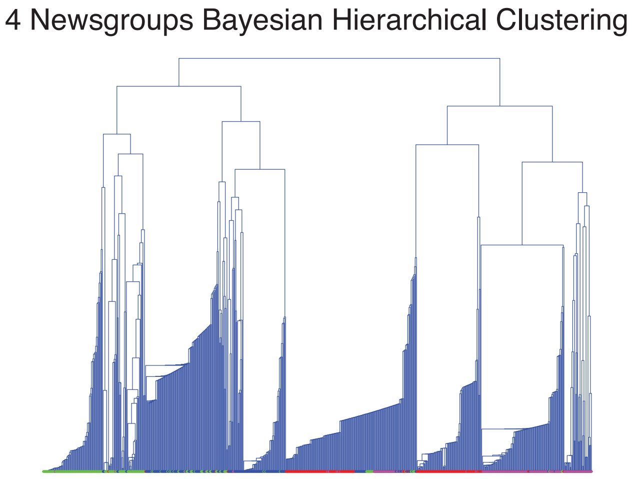
BHC has also been applied to gene expression data, with good results (Savage et al. 2009).

25.6 Clustering datapoints and features

So far, we have been concentrating on clustering datapoints. But each datapoint is often described by multiple features, and we might be interested in clustering them as well. Below we describe some methods for doing this.



(a)



(b)

Figure 25.16 Hierarchical clustering applied to 800 documents from 4 newsgroups (red is rec.autos, blue is rec.sport.baseball, green is rec.sport.hockey, and magenta is sci.space). Top: average linkage hierarchical clustering. Bottom: Bayesian hierarchical clustering. Each of the leaves is labeled with a color, according to which newsgroup that document came from. We see that the Bayesian method results in a clustering that is more consistent with these labels (which were not used during model fitting). Source: Figure 7 of (Heller and Ghahramani 2005). Used with kind permission of Katherine Heller.

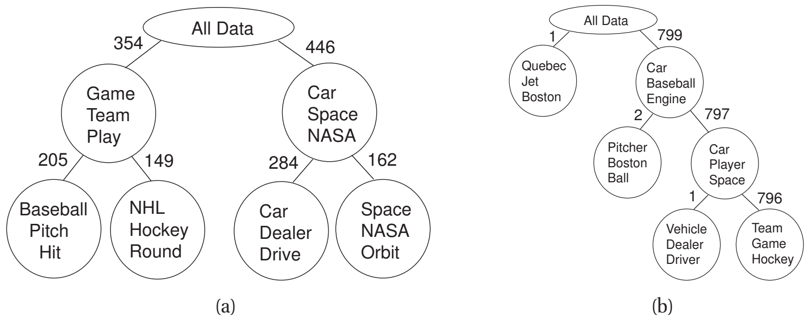


Figure 25.17 Zoom-in on the top nodes in the trees of Figure 25.16. (a) Bayesian method. (b) Average linkage. We show the 3 most probable words per cluster. The number of documents at each cluster is also given. Source: Figure 5 of (Heller and Ghahramani 2005). Used with kind permission of Katherine Heller.

25.6.1 Biclustering

Clustering the rows and columns is known as **biclustering** or **coclustering**. This is widely used in bioinformatics, where the rows often represent genes and the columns represent conditions. It can also be used for collaborative filtering, where the rows represent users and the columns represent movies.

A variety of ad hoc methods for biclustering have been proposed; see (Madeira and Oliveira 2004) for a review. Here we present a simple probabilistic generative model, based on (Kemp et al. 2006) (see also (Sheng et al. 2003) for a related approach). The idea is to associate each row and each column with a latent indicator, $r_i \in \{1, \ldots, K^r\}$, $c_j \in \{1, \ldots, K^c\}$. We then assume the data are iid across samples and across features within each block:

$$p(\mathbf{x}|\mathbf{r}, \mathbf{c}, \boldsymbol{\theta}) = \prod_i \prod_j p(x_{ij}|r_i, c_j, \boldsymbol{\theta}) = p(x_{ij}|\boldsymbol{\theta}_{r_i,c_j}) \tag{25.62}$$

where $\boldsymbol{\theta}_{a,b}$ are the parameters for row cluster a and column cluster b. Rather than using a finite number of clusters for the rows and columns, we can use a Dirchlet process, as in the infinite relational model which we discuss in Section 27.6.1. We can fit this model using e.g., (collapsed) Gibbs sampling.

The behavior of this model is illustrated in Figure 25.18. The data has the form $X(i, j) = 1$ iff animal i has feature j, where $i = 1 : 50$ and $j = 1 : 85$. The animals represent whales, bears, horses, etc. The features represent properties of the habitat (jungle, tree, coastal), or anatomical properties (has teeth, quadrapedal), or behavioral properties (swims, eats meat), etc. The model, using a Bernoulli likelihood, was fit to the data. It discovered 12 animal clusters and 33 feature clusters. For example, it discovered a bicluster that represents the fact that mammals tend to have aquatic features.

25.6.2 Multi-view clustering

The problem with biclustering is that each object (row) can only belong to one cluster. Intuitively, an object can have multiple roles, and can be assigned to different clusters depending on which

O1	killer whale, blue whale, humpback, seal, walrus, dolphin
O2	antelope, horse, giraffe, zebra, deer
O3	monkey, gorilla, chimp
O4	hippo, elephant, rhino
O5	grizzly bear, polar bear
F1	flippers, strain teeth, swims, arctic, coastal, ocean, water
F2	hooves, long neck, horns
F3	hands, bipedal, jungle, tree
F4	bulbous body shape, slow, inactive
F5	meat teeth, eats meat, hunter, fierce
F6	walks, quadrapedal, ground

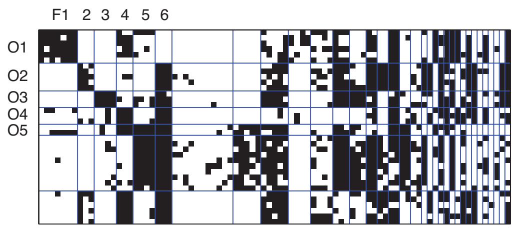


Figure 25.18 Illustration of biclustering . We show 5 of the 12 animal clusters, and 6 of the 33 feature clusters. The original data matrix is shown, partitioned according to the discovered clusters. From Figure 3 of (Kemp et al. 2006). Used with kind permission of Charles Kemp.

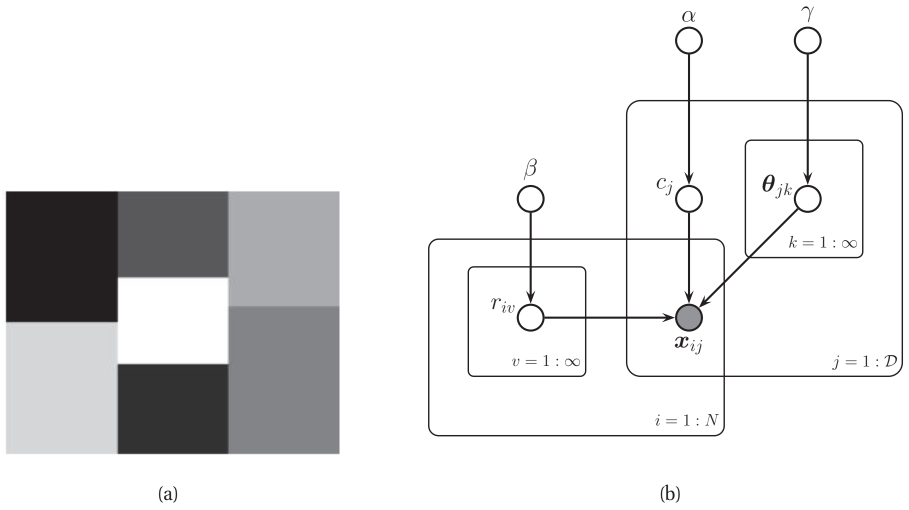


Figure 25.19 (a) Illustration of multi-view clustering. Here we have 3 views (column partitions). In the first view, we have 2 clusters (row partitions). In the second view, we have 3 clusters. In the third view, we have 2 clusters. The number of views and partitions are inferred from data. Rows within each colored block are assumed to generated iid; however, each column can have a different distributional form, which is useful for modeling discrete and continuous data. From Figure 1 of (Guan et al. 2010). Used with kind permission of Jennifer Dy. (b) Corresponding DGM.

subset of features you use. For example, in the animal dataset, we may want to group the animals on the basis of anatomical features (e.g., mammals are warm blooded, reptiles are not), or on the basis of behavioral features (e.g., predators vs prey).

We now present a model that can capture this phenomenon. This model was independently proposed in (Shafto et al. 2006; Mansinghka et al. 2011), who call it **crosscat** (for cross-categorization), and in (Guan et al. 2010; Cui et al. 2010), who call it (non-parametric) **multi-clust**. (See also (Rodriguez and Ghosh 2011) for a very similar model.) The idea is that we partition the columns (features) into V groups or **views**, so $c_j \in \{1, \ldots, V\}$, where $j \in \{1, \ldots, D\}$ indexes features. We will use a Dirichlet process prior for $p(c)$, which allows V to grow automatically. Then for each partition of the columns (i.e., each view), call it v, we partition the rows, again using a DP, as illustrated in Figure 25.19(a). Let $r_{iv} \in \{1, \ldots, K(v)\}$ be the cluster to which the i'th row belongs in view v. Finally, having partitioned the rows and columns, we generate the data: we assume all the rows and columns within a block are iid. We can define the model more precisely as follows:

$$\begin{aligned} p(\mathbf{c}, \mathbf{r}, \mathcal{D}) &= p(\mathbf{c})p(\mathbf{r}|\mathbf{c})p(\mathcal{D}|\mathbf{r}, \mathbf{c}) && (25.63) \\ p(\mathbf{c}) &= \text{DP}(\mathbf{c}|\alpha) && (25.64) \\ p(\mathbf{r}|\mathbf{c}) &= \prod_{v=1}^{V(\mathbf{c})} \text{DP}(\mathbf{r}_v|\beta) && (25.65) \\ p(\mathcal{D}|\mathbf{r}, \mathbf{c}, \boldsymbol{\theta}) &= \prod_{v=1}^{V(\mathbf{c})} \prod_{j:c_j=v} \left[\prod_{k=1}^{K(\mathbf{r}_v)} \int \prod_{i:r_{iv}=k} p(x_{ij}|\boldsymbol{\theta}_{jk}) p(\boldsymbol{\theta}_{jk}) d\boldsymbol{\theta}_{jk} \right] && (25.66) \end{aligned}$$

See Figure 25.19(b) for the DGM.[2]

If the data is binary, and we use a $\text{Beta}(\gamma, \gamma)$ prior for $\boldsymbol{\theta}_{jk}$, the likelihood reduces to

$$p(\mathcal{D}|\mathbf{r}, \mathbf{c}, \gamma) = \prod_{v=1}^{V(\mathbf{c})} \prod_{j:c_j=v} \prod_{k=1}^{K(\mathbf{r}_v)} \frac{\text{Beta}(n_{j,k,v} + \gamma, \overline{n}_{j,k,v} + \gamma)}{\text{Beta}(\gamma, \gamma)} \tag{25.67}$$

where $n_{j,k,v} = \sum_{i:r_{i,v}=k} \mathbb{I}(x_{ij} = 1)$ counts the number of features which are on in the j'th column for view v and for row cluster k. Similarly, $\overline{n}_{j,k,v}$ counts how many features are off. The model is easily extended to other kinds of data, by replacing the beta-Bernoulli with, say, the Gaussian-Gamma-Gaussian model, as discussed in (Guan et al. 2010; Mansinghka et al. 2011).

Approximate MAP estimation can be done using stochastic search (Shafto et al. 2006), and approximate inference can be done using variational Bayes (Guan et al. 2010) or Gibbs sampling (Mansinghka et al. 2011). The hyper-parameter γ for the likelihood can usually be set in a non-informative way, but results are more sensitive to the other two parameters, since α controls the number of column partitions, and β controls the number of row partitions. Hence a more robust technique is to infer the hyper-parameters using MH. This also speeds up convergence (Mansinghka et al. 2011).

Figure 25.20 illustrates the model applied to some binary data containing 22 animals and 106 features. The figures shows the (approximate) MAP partition. The first partition of the columns contains taxonomic features, such as "has bones", "is warm-blooded", "lays eggs", etc. This divides the animals into birds, reptiles/ amphibians, mammals, and invertebrates. The second partition of the columns contains features that are treated as noise, with no apparent structure (except for the single row labeled "frog"). The third partition of the columns contains ecological features like "dangerous", "carnivorous", "lives in water", etc. This divides the animals into prey, land predators, sea predators and air predators. Thus each animal (row) can belong to a different

2. The dependence between $\mathbf{r}$ and $\mathbf{c}$ is not shown, since it is not a dependence between the values of r_{iv} and c_j, but between the cardinality of v and c_j. In other words, the number of row partitions we need to specify (the number of views, indexed by v) depends on the number of column partitions (clusters) that we have.

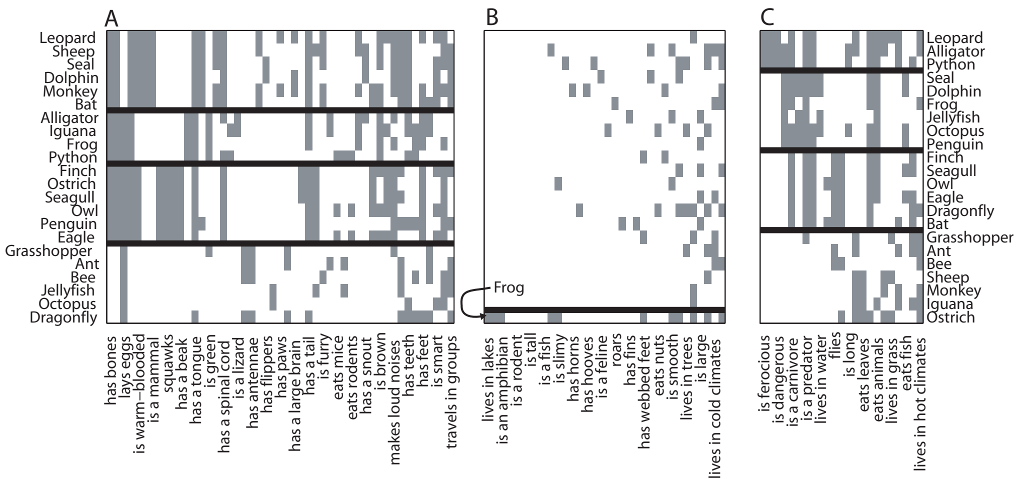


Figure 25.20 MAP estimate produced by the crosscat system when applied to a binary data matrix of animals (rows) by features (columns). See text for details. Source: Figure 7 of (Shafto et al. 2006) . Used with kind permission of Vikash Mansingkha.

cluster depending on what set of features are considered. Uncertainty about the partitions can be handled by sampling.

It is interesting to compare this model to a standard infinite mixture model. While the standard model can represent any density on fixed-sized vectors as $N \to \infty$, it cannot cope with $D \to \infty$, since it has no way to handle irrelevant, noisy or redundant features. By contrast, the crosscat/multi-clust system is robust to irrelevant features: it can just partition them off, and cluster the rows only using the relevant features. Note, however, that it does not need a separate "background" model, since everything is modelled using the same mechanism. This is useful, since one's person's noise is another person's signal. (Indeed, this symmetry may explain why multi-clust outperformed the sparse mixture model approach of (Law et al. 2004) in the experiments reported in (Guan et al. 2010).)

26 *Graphical model structure learning*

26.1 Introduction

We have seen how graphical models can be used to express conditional independence assumptions between variables. In this chapter, we discuss how to learn the structure of the graphical model itself. That is, we want to compute $p(G|\mathcal{D})$, where G is the graph structure, represented as an $V \times V$ adjacency matrix.

As we discussed in Section 1.3.3, there are two main applications of structure learning: knowledge discovery and density estimation. The former just requires a graph topology, whereas the latter requires a fully specified model.

The main obstacle in structure learning is that the number of possible graphs is exponential in the number of nodes: a simple upper bound is $O(2^{V(V-1)/2})$. Thus the full posterior $p(G|\mathcal{D})$ is prohibitively large: even if we could afford to compute it, we could not even store it. So we will seek appropriate summaries of the posterior. These summary statistics depend on our task.

If our goal is knowledge discovery, we may want to compute posterior edge marginals, $p(G_{st} = 1|\mathcal{D})$; we can then plot the corresponding graph, where the thickness of each edge represents our confidence in its presence. By setting a threshold, we can generate a sparse graph, which can be useful for visualization purposes (see Figure 1.11).

If our goal is density estimation, we may want to compute the MAP graph, $\hat{G} \in \text{argmax}_G \, p(G|\mathcal{D})$. In most cases, finding the globally optimal graph will take exponential time, so we will use discrete optimization methods such as heuristic search. However, in the case of trees, we can find the globally optimal graph structure quite efficiently using exact methods, as we discuss in Section 26.3.

If density estimation is our only goal, it is worth considering whether it would be more appropriate to learn a latent variable model, which can capture correlation between the visible variables via a set of latent common causes (see Chapters 12 and 27). Such models are often easier to learn and, perhaps more importantly, they can be applied (for prediction purposes) much more efficiently, since they do not require performing inference in a learned graph with potentially high treewidth. The downside with such models is that the latent factors are often unidentifiable, and hence hard to interpret. Of course, we can combine graphical model structure learning and latent variable learning, as we will show later in this chapter.

In some cases, we don't just want to model the observed correlation between variables; instead, we want to model the *causal* structure behind the data, so we can predict the effects of manipulating variables. This is a much more challenging task, which we briefly discuss in

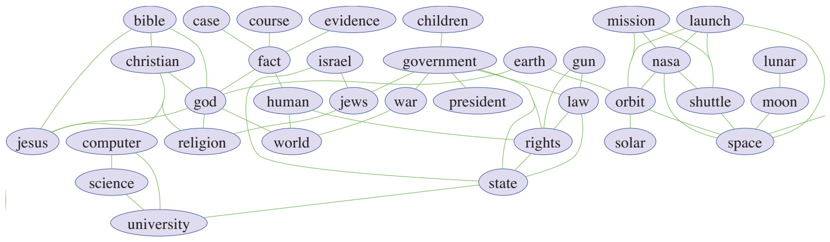


Figure 26.1 Part of a relevance network constructed from the 20-news data shown in Figure 1.2. We show edges whose mutual information is greater than or equal to 20% of the maximum pairwise MI. For clarity, the graph has been cropped, so we only show a subset of the nodes and edges. Figure generated by `relevanceNetworkNewsgroupDemo`.

Section 26.6.

26.2 Structure learning for knowledge discovery

Since computing the MAP graph or the exact posterior edge marginals is in general computationally intractable (Chickering 1996), in this section we discuss some "quick and dirty" methods for learning graph structures which can be used to visualize one's data. The resulting models do not constitute consistent joint probability distributions, so they cannot be used for prediction, and they cannot even be formally evaluated in terms of goodness of fit. Nevertheless, these methods are a useful ad hoc tool to have in one's data visualization toolbox, in view of their speed and simplicity.

26.2.1 Relevance networks

A **relevance network** is a way of visualizing the pairwise mutual information between multiple random variables: we simply choose a threshold and draw an edge from node i to node j if $\mathbb{I}(X_i; X_j)$ is above this threshold. In the Gaussian case, $\mathbb{I}(X_i; X_j) = -\frac{1}{2}\log(1-\rho_{ij}^2)$, where ρ_{ij} is the correlation coefficient (see Exercise 2.13), so we are essentially visualizing $\boldsymbol{\Sigma}$; this is known as the covariance graph (Section 19.4.4.1).

This method is quite popular in systems biology (Margolin et al. 2006), where it is used to visualize the interaction between genes. The trouble with biological examples is that they are hard for non-biologists to understand. So let us instead illustrate the idea using natural language text. Figure 26.1 gives an example, where we visualize the MI between words in the newsgroup dataset from Figure 1.2. The results seem intuitively reasonable.

However, relevance networks suffer from a major problem: the graphs are usually very dense, since most variables are dependent on most other variables, even after thresholding the MIs. For example, suppose X_1 directly influences X_2 which directly influences X_3 (e.g., these form components of a signalling cascade, $X_1 - X_2 - X_3$). Then X_1 has non-zero MI with X_3 (and vice versa), so there will be a $1-3$ edge in the relevance network. Indeed, most pairs will be

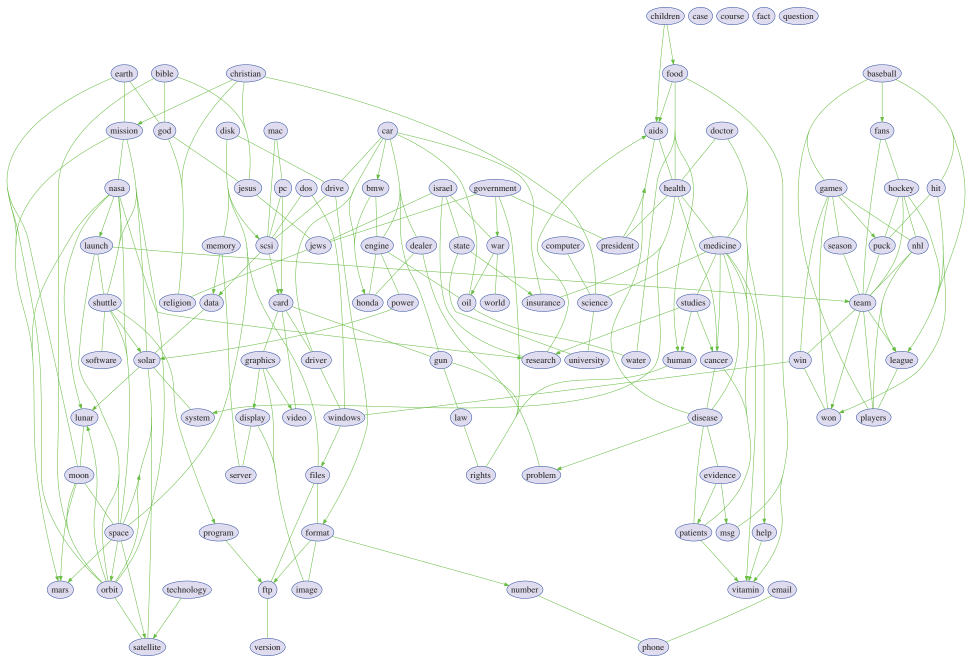


Figure 26.2 A dependency network constructed from the 20-news data. We show all edges with regression weight above 0.5 in the Markov blankets estimated by ℓ_1 penalized logistic regression. Undirected edges represent cases where a directed edge was found in both directions. From Figure 4.9 of (Schmidt 2010). Used with kind permission of Mark Schmidt.

connected.

A better approach is to use graphical models, which represent conditional *independence*, rather than *dependence*. In the above example, X_1 is conditionally independent of X_3 given X_2, so there will not be a $1-3$ edge. Consequently graphical models are usually much sparser than relevance networks, and hence are a more useful way of visualizing interactions between multiple variables.

26.2.2 Dependency networks

A simple and efficient way to learn a graphical model structure is to independently fit D sparse full-conditional distributions $p(x_t|\mathbf{x}_{-t})$; this is called a **dependency network** (Heckerman et al. 2000). The chosen variables constitute the inputs to the node, i.e., its Markov blanket. We can then visualize the resulting sparse graph. The advantage over relevance networks is that redundant variables will not be selected as inputs.

We can use any kind of sparse regression or classification method to fit each CPD. (Heckerman et al. 2000) uses classification/ regression trees, (Meinshausen and Buhlmann 2006) use ℓ_1-regularized linear regression, (Wainwright et al. 2006) use ℓ_1-regularized logistic regression (see `depnetFit` for some code), (Dobra 2009) uses Bayesian variable selection, etc. (Meinshausen and Buhlmann 2006) discuss theoretical conditions under which ℓ_1-regularized linear regression can recover the true graph structure, assuming the data was generated from a sparse Gaussian graphical model.

Figure 26.2 shows a dependency network that was learned from the 20-newsgroup data using ℓ_1 regularized logistic regression, where the penalty parameter λ was chosen by BIC. Many of the words present in these estimated Markov blankets represent fairly natural associations (aids:disease, baseball:fans, bible:god, bmw:car, cancer:patients, etc.). However, some of the estimated statistical dependencies seem less intuitive, such as baseball:windows and bmw:christian. We can gain more insight if we look not only at the sparsity pattern, but also the values of the regression weights. For example, here are the incoming weights for the first 5 words:

- **aids**: children (0.53), disease (0.84), fact (0.47), health (0.77), president (0.50), research (0.53)
- **baseball**: *christian* (-0.98), *drive* (-0.49), games (0.81), *god* (-0.46), *government* (-0.69), hit (0.62), *memory* (-1.29), players (1.16), season (0.31), *software* (-0.68), *windows* (-1.45)
- **bible**: *car* (-0.72), *card* (-0.88), christian (0.49), fact (0.21), god (1.01), jesus (0.68), orbit (0.83), *program* (-0.56), religion (0.24), version (0.49)
- **bmw**: car (0.60), *christian* (-11.54), engine (0.69), *god* (-0.74), *government* (-1.01), *help* (-0.50), *windows* (-1.43)
- **cancer**: disease (0.62), medicine (0.58), patients (0.90), research (0.49), studies (0.70)

Words in italic red have negative weights, which represents a dissociative relationship. For example, the model reflects that baseball:windows is an unlikely combination. It turns out that most of the weights are negative (1173 negative, 286 positive, 8541 zero) in this model.

In addition to visualizing the data, a dependency network can be used for inference. However, the only algorithm we can use is Gibbs sampling, where we repeatedly sample the nodes with missing values from their full conditionals. Unfortunately, a product of full conditionals does not, in general, constitute a representation of any valid joint distribution (Heckerman et al. 2000), so the output of the Gibbs sampler may not be meaningful. Nevertheless, the method can sometimes give reasonable results if there is not much missing data, and it is a useful method for data imputation (Gelman and Raghunathan 2001). In addition, the method can be used as an initialization technique for more complex structure learning methods that we discuss below.

26.3 Learning tree structures

For the rest of this chapter, we focus on learning fully specified joint probability models, which can be used for density estimation, prediction and knowledge discovery.

Since the problem of structure learning for general graphs is NP-hard (Chickering 1996), we start by considering the special case of trees. Trees are special because we can learn their structure efficiently, as we disuscs below, and because, once we have learned the tree, we can use them for efficient exact inference, as discussed in Section 20.2.

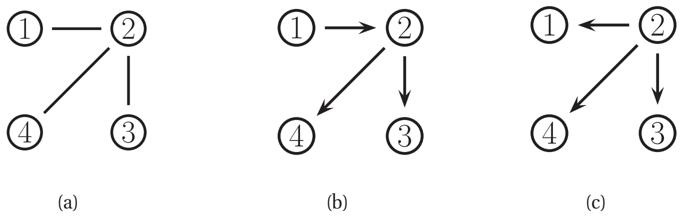


Figure 26.3 An undirected tree and two equivalent directed trees.

26.3.1 Directed or undirected tree?

Before continuing, we need to discuss the issue of whether we should use directed or undirected trees. A directed tree, with a single root node r, defines a joint distribution as follows:

$$p(\mathbf{x}|T) = \prod_{t \in V} p(x_t|x_{\text{pa}(t)}) \tag{26.1}$$

where we define $\text{pa}(r) = \emptyset$. For example, in Figure 26.3(b-c), we have

$$\begin{aligned} p(x_1, x_2, x_3, x_4|T) &= p(x_1)p(x_2|x_1)p(x_3|x_2)p(x_4|x_2) && (26.2) \\ &= p(x_2)p(x_1|x_2)p(x_3|x_2)p(x_4|x_2) && (26.3) \end{aligned}$$

We see that the choice of root does not matter: both of these models are equivalent.

To make the model more symmetric, it is preferable to use an undirected tree. This can be represented as follows:

$$p(\mathbf{x}|T) = \prod_{t \in V} p(x_t) \prod_{(s,t) \in E} \frac{p(x_s, x_t)}{p(x_s)p(x_t)} \tag{26.4}$$

where $p(x_s, x_t)$ is an edge marginal and $p(x_t)$ is a node marginal. For example, in Figure 26.3(a) we have

$$p(x_1, x_2, x_3, x_4|T) = p(x_1)p(x_2)p(x_3)p(x_4) \frac{p(x_1, x_2)p(x_2, x_3)p(x_2, x_4)}{p(x_1)p(x_2)p(x_2)p(x_3)p(x_2)p(x_4)} \tag{26.5}$$

To see the equivalence with the directed representation, let us cancel terms to get

$$\begin{aligned} p(x_1, x_2, x_3, x_4|T) &= p(x_1, x_2) \frac{p(x_2, x_3)}{p(x_2)} \frac{p(x_2, x_4)}{p(x_2)} && (26.6) \\ &= p(x_1)p(x_2|x_1)p(x_3|x_2)p(x_4|x_2) && (26.7) \\ &= p(x_2)p(x_1|x_2)p(x_3|x_2)p(x_4|x_2) && (26.8) \end{aligned}$$

where $p(x_t|x_s) = p(x_s, x_t)/p(x_s)$.

Thus a tree can be represented as either an undirected or directed graph: the number of parameters is the same, and hence the complexity of learning is the same. And of course, inference is the same in both representations, too. The undirected representation, which is symmetric, is useful for structure learning, but the directed representation is more convenient for parameter learning.

26.3.2 Chow-Liu algorithm for finding the ML tree structure

Using Equation 26.4, we can write the log-likelihood for a tree as follows:

$$\begin{aligned}\log p(\mathcal{D}|\boldsymbol{\theta}, T) &= \sum_t \sum_k N_{tk} \log p(x_t = k|\boldsymbol{\theta}) \\ &+ \sum_{s,t} \sum_{j,k} N_{stjk} \log \frac{p(x_s = j, x_t = k|\boldsymbol{\theta})}{p(x_s = j|\boldsymbol{\theta})p(x_t = k|\boldsymbol{\theta})} \end{aligned} \quad (26.9)$$

where N_{stjk} is the number of times node s is in state j and node t is in state k, and N_{tk} is the number of times node t is in state k. We can rewrite these counts in terms of the empirical distribution: $N_{stjk} = N p_{\text{emp}}(x_s = j, x_t = k)$ and $N_{tk} = N p_{\text{emp}}(x_t = k)$. Setting $\boldsymbol{\theta}$ to the MLEs, this becomes

$$\frac{\log p(\mathcal{D}|\boldsymbol{\theta}, T)}{N} = \sum_{t \in \mathcal{V}} \sum_k p_{\text{emp}}(x_t = k) \log p_{\text{emp}}(x_t = k) \quad (26.10)$$

$$+ \sum_{(s,t) \in \mathcal{E}(T)} \mathbb{I}(x_s, x_t|\hat{\boldsymbol{\theta}}_{st}) \quad (26.11)$$

where $\mathbb{I}(x_s, x_t|\hat{\boldsymbol{\theta}}_{st}) \geq 0$ is the mutual information between x_s and x_t given the empirical distribution:

$$\mathbb{I}(x_s, x_t|\hat{\boldsymbol{\theta}}_{st}) = \sum_j \sum_k p_{\text{emp}}(x_s = j, x_t = k) \log \frac{p_{\text{emp}}(x_s = j, x_t = k)}{p_{\text{emp}}(x_s = j) p_{\text{emp}}(x_t = k)} \quad (26.12)$$

Since the first term in Equation 26.11 is independent of the topology T, we can ignore it when learning structure. Thus the tree topology that maximizes the likelihood can be found by computing the maximum weight spanning tree, where the edge weights are the pairwise mutual informations, $\mathbb{I}(y_s, y_t|\hat{\boldsymbol{\theta}}_{st})$. This is called the **Chow-Liu algorithm** (Chow and Liu 1968).

There are several algorithms for finding a max spanning tree (MST). The two best known are Prim's algorithm and Kruskal's algorithm. Both can be implemented to run in $O(E \log V)$ time, where $E = V^2$ is the number of edges and V is the number of nodes. See e.g., (Sedgewick and Wayne 2011, 4.3) for details. Thus the overall running time is $O(NV^2 + V^2 \log V)$, where the first term is the cost of computing the sufficient statistics.

Figure 26.4 gives an example of the method in action, applied to the binary 20 newsgroups data shown in Figure 1.2. The tree has been arbitrarily rooted at the node representing "email". The connections that are learned seem intuitively reasonable.

26.3.3 Finding the MAP forest

Since all trees have the same number of parameters, we can safely used the maximum likelihood score as a model selection criterion without worrying about overfitting. However, sometimes we may want to fit a **forest** rather than a single tree, since inference in a forest is much faster than in a tree (we can run belief propagation in each tree in the forest in parallel). The MLE criterion will never choose to omit an edge. However, if we use the marginal likelihood or a penalized likelihood (such as BIC), the optimal solution may be a forest. Below we give the details for the marginal likelihood case.

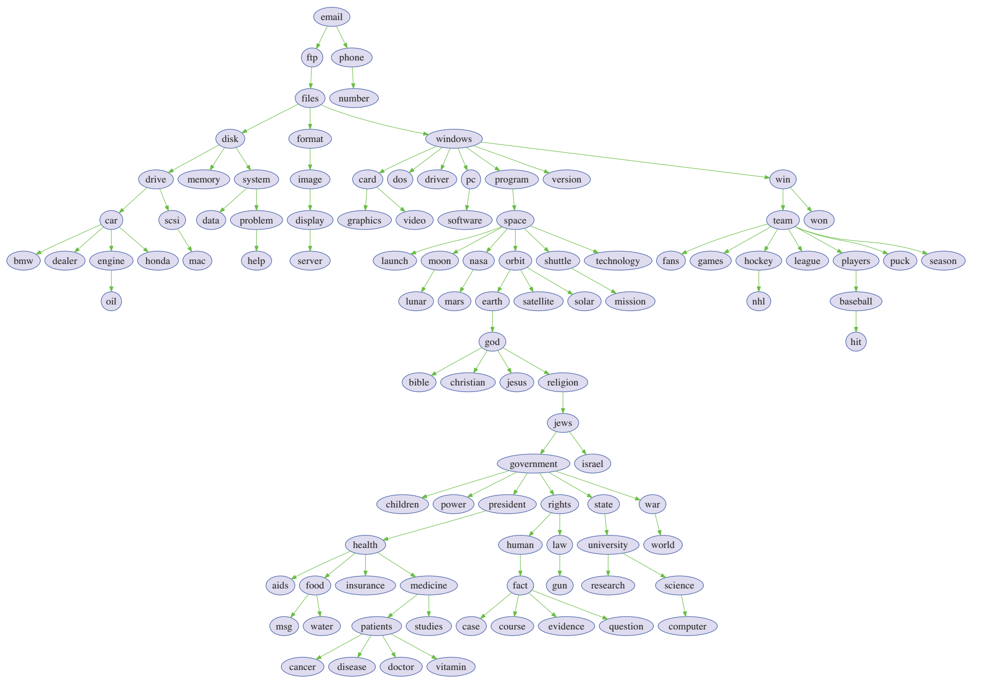


Figure 26.4 The MLE tree on the 20-newsgroup data. From Figure 4.11 of (Schmidt 2010). Used with kind permission of Mark Schmidt. (A topologically equivalent tree can be produced using `chowliuTreeDemo`.)

In Section 26.4.2.2, we explain how to compute the marginal likelihood of any DAG using a Dirichlet prior for the CPTs. The resulting expression can be written as follows:

$$\log p(\mathcal{D}|T) = \sum_{t \in \mathcal{V}} \log \int \prod_{i=1}^{N} p(x_{it}|\mathbf{x}_{i,\text{pa}(t)}|\boldsymbol{\theta}_t) p(\boldsymbol{\theta}_t) d\boldsymbol{\theta}_t = \sum_t \text{score}(\mathbf{N}_{t,\text{pa}(t)}) \tag{26.13}$$

where $\mathbf{N}_{t,\text{pa}(t)}$ are the counts (sufficient statistics) for node t and its parents, and score is defined in Equation 26.28.

Now suppose we only allow DAGs with at most one parent. Following (Heckerman et al. 1995, p227), let us associate a weight with each $s \to t$ edge, $w_{s,t} \triangleq \text{score}(t|s) - \text{score}(t|0)$, where $\text{score}(t|0)$ is the score when t has no parents. Note that the weights might be negative (unlike the MLE case, where edge weights are aways non-negative because they correspond to mutual information). Then we can rewrite the objective as follows:

$$\log p(\mathcal{D}|T) = \sum_t \text{score}(t|\text{pa}(t)) = \sum_t w_{\text{pa}(t),t} + \sum_t \text{score}(t|0) \tag{26.14}$$

The last term is the same for all trees T, so we can ignore it. Thus finding the most probable tree amounts to finding a **maximal branching** in the corresponding weighted directed graph. This can be found using the algorithm in (Gabow et al. 1984).

If the scoring function is prior and likelihood equivalent (these terms are explained in Section 26.4.2.3), we have

$$\text{score}(s|t) + \text{score}(t|0) = \text{score}(t|s) + \text{score}(s|0) \tag{26.15}$$

and hence the weight matrix is symmetric. In this case, the maximal branching is the same as the maximal weight forest. We can apply a slightly modified version of the MST algorithm to find this (Edwards et al. 2010). To see this, let $G = (V, E)$ be a graph with both positive and negative edge weights. Now let G' be a graph obtained by omitting all the negative edges from G. This cannot reduce the total weight, so we can find the maximum weight forest of G by finding the MST for each connected component of G'. We can do this by running Kruskal's algorithm directly on G': there is no need to find the connected components explicitly.

26.3.4 Mixtures of trees

A single tree is rather limited in its expressive power. Later in this chapter we discuss ways to learn more general graphs. However, the resulting graphs can be expensive to do inference in. An interesting alternative is to learn a **mixture of trees** (Meila and Jordan 2000), where each mixture component may have a different tree topology. This is like an unsupervised version of the TAN classifier discussed in Section 10.2.1. We can fit a mixture of trees by using EM: in the E step, we compute the responsibilities of each cluster for each data point, and in the M step, we use a weighted version of the Chow-Liu algorithm. See (Meila and Jordan 2000) for details.

In fact, it is possible to create an "infinite mixture of trees", by integrating out over all possible trees. Remarkably, this can be done in V^3 time using the matrix tree theorem. This allows us to perform exact Bayesian inference of posterior edge marginals etc. However, it is not tractable to use this infinite mixture for inference of hidden nodes. See (Meila and Jaakkola 2006) for details.

26.4 Learning DAG structures

In this section, we discuss how to compute (functions of) $p(G|\mathcal{D})$, where G is constrained to be a DAG. This is often called **Bayesian network structure learning**. In this section, we assume there is no missing data, and that there are no hidden variables. This is called the **complete data assumption**. For simplicity, we will focus on the case where all the variables are categorical and all the CPDs are tables, although the results generalize to real-valued data and other kinds of CPDs, such as linear-Gaussian CPDs.

Our presentation is based in part on (Heckerman et al. 1995), although we will follow the notation of Section 10.4.2. In particular, let $x_{it} \in \{1, \ldots, K_t\}$ be the value of node t in case i, where K_t is the number of states for node t. Let $\theta_{tck} \triangleq p(x_t = k|\mathbf{x}_{\text{pa}(t)} = c)$, for $k = 1 : K_t$, and $c = 1 : C_t$, where C_t is the number of parent combinations (possible conditioning cases). For notational simplicity, we will often assume $K_t = K$, so all nodes have the same number of states. We will also let $d_t = \dim(\text{pa}(t))$ be the degree or fan-in of node t, so that $C_t = K^{d_t}$.

26.4.1 Markov equivalence

In this section, we discuss some fundamental limits to our ability to learn DAG structures from data.

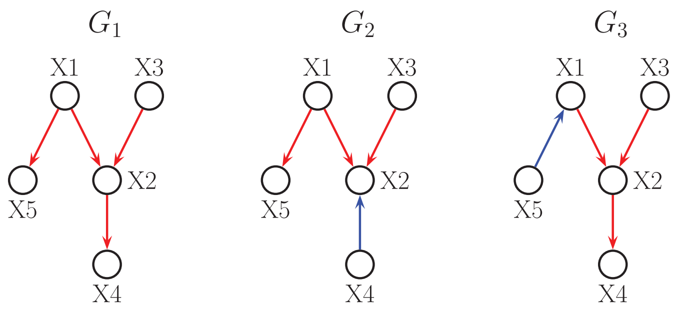


Figure 26.5 Three DAGs. G_1 and G_3 are Markov equivalent, G_2 is not.

Consider the following 3 DGMs: $X \to Y \to Z$, $X \leftarrow Y \leftarrow Z$ and $X \leftarrow Y \to Z$. These all represent the same set of CI statements, namely

$$X \perp Z|Y, \quad X \not\perp Z \tag{26.16}$$

We say these graphs are **Markov equivalent**, since they encode the same set of CI assumptions. That is, they all belong to the same Markov **equivalence class**. However, the v-structure $X \to Y \leftarrow Z$ encodes $X \perp Z$ and $X \not\perp Z|Y$, which represents the opposite set of CI assumptions.

One can prove the following theorem.

Theorem 26.4.1 (Verma and Pearl (Verma and Pearl 1990))**.** *Two structures are Markov equivalent iff they have the same undirected skeleton and the same set of v-structures.*

For example, referring to Figure 26.5, we see that $G_1 \not\equiv G_2$, since reversing the $2 \to 4$ arc creates a new v-structure. However, $G_1 \equiv G_3$, since reversing the $1 \to 5$ arc does not create a new v-structure.

We can represent a Markov equivalence class using a single **partially directed acyclic graph** (PDAG), also called an **essential graph** or **pattern**, in which some edges are directed and some undirected. The undirected edges represent reversible edges; any combination is possible so long as no new v-structures are created. The directed edges are called **compelled edges**, since changing their orientation would change the v-structures and hence change the equivalence class. For example, the PDAG $X - Y - Z$ represents $\{X \to Y \to Z, X \leftarrow Y \leftarrow Z, X \leftarrow Y \to Z\}$ which encodes $X \not\perp Z$ and $X \perp Z|Y$. See Figure 26.6.

The significance of the above theorem is that, when we learn the DAG structure from data, we will not be able to uniquely identify all of the edge directions, even given an infinite amount of data. We say that we can learn DAG structure "up to Markov equivalence". This also cautions us not to read too much into the meaning of particular edge orientations, since we can often change them without changing the model in any observable way.

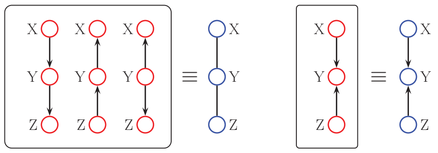


Figure 26.6 PDAG representation of Markov equivalent DAGs.

26.4.2 Exact structural inference

In this section, we discuss how to compute the exact posterior over graphs, $p(G|\mathcal{D})$, ignoring for now the issue of computational tractability.

26.4.2.1 Deriving the likelihood

Assuming there is no missing data, and that all CPDs are tabular, the likelihood can be written as follows:

$$\begin{aligned} p(\mathcal{D}|G,\boldsymbol{\theta}) &= \prod_{i=1}^{N}\prod_{t=1}^{V}\text{Cat}(x_{it}|\mathbf{x}_{i,\text{pa}(t)},\boldsymbol{\theta}_t) && (26.17)\\ &= \prod_{i=1}^{N}\prod_{t=1}^{V}\prod_{c=1}^{C_t}\text{Cat}(x_{it}|\boldsymbol{\theta}_{tc})^{\mathbb{I}(\mathbf{x}_{i,\text{pa}(t)}=c)} && (26.18)\\ &= \prod_{i=1}^{N}\prod_{t=1}^{V}\prod_{c=1}^{C_t}\prod_{k=1}^{K_t}\theta_{tck}^{\mathbb{I}(x_{i,t}=k,\mathbf{x}_{i,\text{pa}(t)}=c)} && (26.19)\\ &= \prod_{t=1}^{V}\prod_{c=1}^{C_t}\prod_{k=1}^{K_t}\theta_{tck}^{N_{tck}} && (26.20)\end{aligned}$$

where N_{tck} is the number of times node t is in state k and its parents are in state c. (Technically these counts depend on the graph structure G, but we drop this from the notation.)

26.4.2.2 Deriving the marginal likelihood

Of course, choosing the graph with the maximum likelihood will always pick a fully connected graph (subject to the acyclicity constraint), since this maximizes the number of parameters. To avoid such overfitting, we will choose the graph with the maximum marginal likelihood, $p(\mathcal{D}|G)$; the magic of the Bayesian Occam's razor will then penalize overly complex graphs.

To compute the marginal likelihood, we need to specify priors on the parameters. We will make two standard assumptions. First, we assume **global prior parameter independence**, which means

$$p(\boldsymbol{\theta}) = \prod_{t=1}^{V} p(\boldsymbol{\theta}_t) \tag{26.21}$$

Second, we assume **local prior parameter independence**, which means

$$p(\boldsymbol{\theta}_t) = \prod_{c=1}^{C_t} p(\boldsymbol{\theta}_{tc}) \tag{26.22}$$

for each t. It turns out that these assumtions imply that the prior for each row of each CPT must be a Dirichlet (Geiger and Heckerman 1997), that is,

$$p(\boldsymbol{\theta}_{tc}) = \mathrm{Dir}(\boldsymbol{\theta}_{tc}|\boldsymbol{\alpha}_{tc}) \tag{26.23}$$

Given these assumptions, and using the results of Section 5.3.2.2, we can write down the marginal likelihood of any DAG as follows:

$$\begin{aligned} p(\mathcal{D}|G) &= \prod_{t=1}^{V}\prod_{c=1}^{C_t} \int \left[\prod_{i:x_{i,\mathrm{pa}(t)}=c} \mathrm{Cat}(x_{it}|\boldsymbol{\theta}_{tc}) \right] \mathrm{Dir}(\boldsymbol{\theta}_{tc}) d\boldsymbol{\theta}_{tc} & (26.24) \\ &= \prod_{t=1}^{V}\prod_{c=1}^{C_t} \frac{B(\mathbf{N}_{tc}+\boldsymbol{\alpha}_{tc})}{B(\boldsymbol{\alpha}_{tc})} & (26.25) \\ &= \prod_{t=1}^{V}\prod_{c=1}^{C_t} \frac{\Gamma(N_{tc})}{\Gamma(N_{tc}+\alpha_{tc})} \prod_{k=1}^{K_t} \frac{\Gamma(N_{tck}+\alpha_{tck}^G)}{\Gamma(\alpha_{ijk}^G)} & (26.26) \\ &= \prod_{t=1}^{V} \mathrm{score}(\mathbf{N}_{t,\mathrm{pa}(t)}) & (26.27) \end{aligned}$$

where $N_{tc} = \sum_k N_{tck}$, $\alpha_{tc} = \sum_k \alpha_{tck}$, $\mathbf{N}_{t,\mathrm{pa}(t)}$ is the vector of counts (sufficient statistics) for node t and its parents, and score() is a local scoring function defined by

$$\mathrm{score}(\mathbf{N}_{t,\mathrm{pa}(t)}) \triangleq \prod_{c=1}^{C_t} \frac{B(\mathbf{N}_{tc}+\boldsymbol{\alpha}_{tc})}{B(\boldsymbol{\alpha}_{tc})} \tag{26.28}$$

We say that the marginal likelihood **decomposes** or factorizes according to the graph structure.

26.4.2.3 Setting the prior

How should we set the hyper-parameters α_{tck}? It is tempting to use a Jeffreys prior of the form $\alpha_{tck} = \frac{1}{2}$ (Equation 5.62). However, it turns out that this violates a property called **likelihood equivalence**, which is sometimes considered desirable. This property says that if G_1 and G_2 are Markov equivalent (Section 26.4.1), they should have the same marginal likelihood, since they are essentially equivalent models. Geiger and Heckerman (1997) proved that, for complete graphs, the only prior that satisfies likelihood equivalence and parameter independence is the Dirichlet prior, where the pseudo counts have the form

$$\alpha_{tck} = \alpha\ p_0(x_t = k, \mathbf{x}_{\mathrm{pa}(t)} = c) \tag{26.29}$$

where $\alpha > 0$ is called the **equivalent sample size**, and p_0 is some prior joint probability distribution. This is called the **BDe** prior, which stands for Bayesian Dirichlet likelihood equivalent.

To derive the hyper-parameters for other graph structures, Geiger and Heckerman (1997) invoked an additional assumption called **parameter modularity**, which says that if node X_t has the same parents in G_1 and G_2, then $p(\boldsymbol{\theta}_t|G_1) = p(\boldsymbol{\theta}_t|G_2)$. With this assumption, we can always derive $\boldsymbol{\alpha}_t$ for a node t in any other graph by marginalizing the pseudo counts in Equation 26.29.

Typically the prior distribution p_0 is assumed to be uniform over all possible joint configurations. In this case, we have

$$\alpha_{tck} = \frac{\alpha}{K_t C_t} \tag{26.30}$$

since $p_0(x_t = k, \mathbf{x}_{\text{pa}(t)} = c) = \frac{1}{K_t C_t}$. Thus if we sum the pseudo counts over all $C_t \times K_t$ entries in the CPT, we get a total equivalent sample size of α. This is called the **BDeu** prior, where the "u" stands for uniform. This is the most widely used prior for learning Bayes net structures. For advice on setting the global tuning parameter α, see (Silander et al. 2007).

26.4.2.4 Simple worked example

We now give a very simple worked example from (Neapolitan 2003, p.438). Suppose we have just 2 binary nodes, and the following 8 data cases:

X_1	X_2
1	1
1	2
1	1
2	2
1	1
2	1
1	1
2	2

Suppose we are interested in two possible graphs: G_1 is $X_1 \rightarrow X_2$ and G_2 is the disconnected graph. The empirical counts for node 1 in G_1 are $\mathbf{N}_1 = (5, 3)$ and for node 2 are

	$X_2 = 1$	$X_2 = 2$
$X_1 = 1$	4	1
$X_1 = 2$	1	2

The BDeu prior for G_1 is $\boldsymbol{\alpha}_1 = (\alpha/2, \alpha/2)$, $\boldsymbol{\alpha}_{2|x_1=1} = (\alpha/4, \alpha/4)$ and $\boldsymbol{\alpha}_{2|x_1=2} = (\alpha/4, \alpha/4)$. For G_2, the prior for $\boldsymbol{\theta}_1$ is the same, and for $\boldsymbol{\theta}_2$ it is $\boldsymbol{\alpha}_{2|x_1=1} = (\alpha/2, \alpha/2)$ and $\boldsymbol{\alpha}_{2|x_1=2} = (\alpha/2, \alpha/2)$. If we set $\alpha = 4$, and use the BDeu prior, we find $p(\mathcal{D}|G_1) = 7.2150 \times 10^{-6}$ and $p(\mathcal{D}|G_2) = 6.7465 \times 10^{-6}$. Hence the posterior probabilites, under a uniform graph prior, are $p(G_1|\mathcal{D}) = 0.51678$ and $p(G_2|\mathcal{D}) = 0.48322$.

26.4.2.5 Example: analysis of the college plans dataset

We now consider a more interesting example from (Heckerman et al. 1997). Consider the data set collected in 1968 by Sewell and Shah which measured 5 variables that might influence the decision of high school students about whether to attend college. Specifically, the variables are as follows:

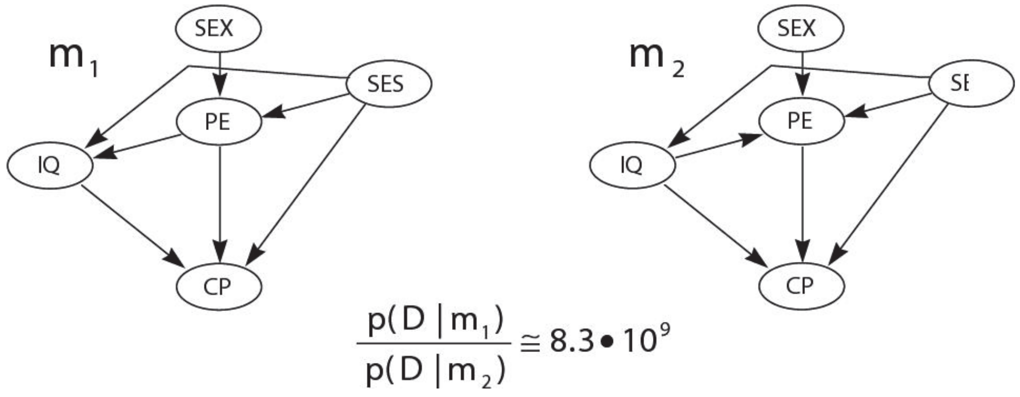


Figure 26.7 The two most probable DAGs learned from the Sewell-Shah data. Source: (Heckerman et al. 1997) . Used with kind permission of David Heckerman

- **Sex** Male or female
- **SES** Socio economic status: low, lower middle, upper middle or high.
- **IQ** Intelligence quotient: discretized into low, lower middle, upper middle or high.
- **PE** Parental encouragment: low or high
- **CP** College plans: yes or no.

These variables were measured for 10,318 Wisconsin high school seniors. There are $2 \times 4 \times 4 \times 2 \times = 128$ possible joint configurations.

Heckerman et al. computed the exact posterior over all 29,281 possible 5 node DAGs, except for ones in which SEX and/or SES have parents, and/or CP have children. (The prior probability of these graphs was set to 0, based on domain knowledge.) They used the BDeu score with $\alpha = 5$, although they said that the results were robust to any α in the range 3 to 40. The top two graphs are shown in Figure 26.7. We see that the most probable one has approximately all of the probability mass, so the posterior is extremely peaked.

It is tempting to interpret this graph in terms of causality (see Section 26.6). In particular, it seems that socio-economic status, IQ and parental encouragment all causally influence the decision about whether to go to college, which makes sense. Also, sex influences college plans only indirectly through parental encouragement, which also makes sense. However, the direct link from socio economic status to IQ seems surprising; this may be due to a hidden common cause. In Section 26.5.1.4 we will re-examine this dataset allowing for the presence of hidden variables.

26.4.2.6 The K2 algorithm

Suppose we know a total ordering of the nodes. Then we can compute the distribution over parents for each node independently, without the risk of introducing any directed cycles: we simply enumerate over all possible subsets of ancestors and compute their marginal likelihoods.[1] If we just return the best set of parents for each node, we get the the **K2 algorithm** (Cooper and Herskovits 1992).

26.4.2.7 Handling non-tabular CPDs

If all CPDs are linear Gaussian, we can replace the Dirichlet-multinomial model with the normal-gamma model, and thus derive a different exact expression for the marginal likelihood. See (Geiger and Heckerman 1994) for the details. In fact, we can easily combine discrete nodes and Gaussian nodes, as long as the discrete nodes always have discrete parents; this is called a **conditional Gaussian** DAG. Again, we can compute the marginal likelihood in closed form. See (Bottcher and Dethlefsen 2003) for the details.

In the general case (i.e., everything except Gaussians and CPTs), we need to approximate the marginal likelihood. The simplest approach is to use the BIC approximation, which has the form

$$\sum_t \log p(\mathcal{D}_t|\hat{\boldsymbol{\theta}}_t) - \frac{K_t C_t}{2} \log N \tag{26.31}$$

26.4.3 Scaling up to larger graphs

The main challenge in computing the posterior over DAGs is that there are so many possible graphs. More precisely, (Robinson 1973) showed that the number of DAGs on D nodes satisfies the following recurrence:

$$f(D) = \sum_{i=1}^{D} (-1)^{i+1} \binom{D}{i} 2^{i(D-i)} f(D-i) \tag{26.32}$$

for $D > 2$. The base case is $f(1) = 1$. Solving this recurrence yields the following sequence: 1, 3, 25, 543, 29281, 3781503, etc.[2] In view of the enormous size of the hypothesis space, we are generally forced to use approximate methods, some of which we review below.

26.4.3.1 Approximating the mode of the posterior

We can use dynamic programming to find the globally optimal MAP DAG (up to Markov equivalence) (Koivisto and Sood 2004; Silander and Myllmaki 2006). Unfortunately this method takes $V 2^V$ time and space, making it intractable beyond about 16 nodes. Indeed, the general problem of finding the globally optimal MAP DAG is provably NP-complete (Chickering 1996),

Consequently, we must settle for finding a locally optimal MAP DAG. The most common method is greedy hill climbing: at each step, the algorithm proposes small changes to the current graph, such as adding, deleting or reversing a single edge; it then moves to the neighboring graph which most increases the posterior. The method stops when it reaches a local maximum. It is important that the method only proposes local changes to the graph,

1. We can make this method more efficient by using ℓ_1-regularization to select the parents (Schmidt et al. 2007). In this case, we need to approximate the marginal likelihood as we discuss below.

2. A longer list of values can be found at `http://www.research.att.com/~njas/sequences/A003024`. Interestingly, the number of DAGs is equal to the number of (0,1) matrices all of whose eigenvalues are positive real numbers (McKay et al. 2004).

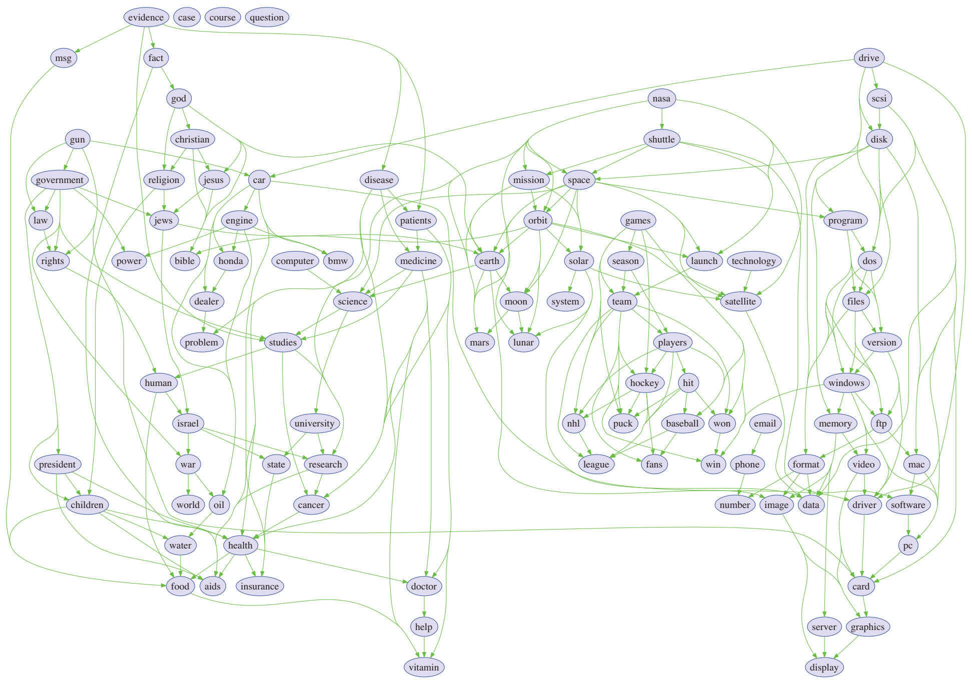


Figure 26.8 A locally optimal DAG learned from the 20-newsgroup data. From Figure 4.10 of (Schmidt 2010). Used with kind permission of Mark Schmidt.

since this enables the change in marginal likelihood (and hence the posterior) to be computed in constant time (assuming we cache the sufficient statistics). This is because all but one or two of the terms in Equation 26.25 will cancel out when computing the log Bayes factor $\delta(G \to G') = \log p(G'|\mathcal{D}) - \log p(G|\mathcal{D})$.

We can initialize the search from the best tree, which can be found using exact methods discussed in Section 26.3. For speed, we can restrict the search so it only adds edges which are part of the Markov blankets estimated from a dependency network (Schmidt 2010). Figure 26.8 gives an example of a DAG learned in this way from the 20-newsgroup data.

We can use techniques such as multiple random restarts to increase the chance of finding a good local maximum. We can also use more sophisticated local search methods, such as genetic algorithms or simulated annealing, for structure learning.

26.4.3.2 Approximating other functions of the posterior

If our goal is knowledge discovery, the MAP DAG can be misleading, for reasons we discussed in Section 5.2.1. A better approach is to compute the probability that each edge is present, $p(G_{st} = 1|\mathcal{D})$, of the probability there is a path from s to t. We can do this exactly using dynamic programming (Koivisto 2006; Parviainen and Koivisto 2011). Unfortunately these methods take $V 2^V$ time in the general case, making them intractable for graphs with more than about 16 nodes.

An approximate method is to sample DAGs from the posterior, and then to compute the fraction of times there is an $s \to t$ edge or path for each (s,t) pair. The standard way to draw samples is to use the Metropolis Hastings algorithm (Section 24.3), where we use the same local proposal as we did in greedy search (Madigan and Raftery 1994).

A faster-mixing method is to use a collapsed MH sampler, as suggested in (Friedman and Koller 2003). This exploits the fact that, if a total ordering of the nodes is known, we can select the parents for each node independently, without worrying about cycles, as discussed in Section 26.4.2.6. By summing over all possible choice of parents, we can marginalize out this part of the problem, and just sample total orders. (Ellis and Wong 2008) also use order-space (collapsed) MCMC, but this time with a parallel tempering MCMC algorithm.

26.5 Learning DAG structure with latent variables

Sometimes the complete data assumption does not hold, either because we have missing data, and/ or because we have hidden variables. In this case, the marginal likelihood is given by

$$p(\mathcal{D}|G) = \int \sum_{\mathbf{h}} p(\mathcal{D}, \mathbf{h}|\boldsymbol{\theta}, G)p(\boldsymbol{\theta}|G)d\boldsymbol{\theta} = \sum_{\mathbf{h}} \int p(\mathcal{D}, \mathbf{h}|\boldsymbol{\theta}, G)p(\boldsymbol{\theta}|G)d\boldsymbol{\theta} \tag{26.33}$$

where $\mathbf{h}$ represents the hidden or missing data.

In general this is intractable to compute. For example, consider a mixture model, where we don't observe the cluster label. In this case, there are K^N possible completions of the data (assuming we have K clusters); we can evaluate the inner integral for each one of these assignments to $\mathbf{h}$, but we cannot afford to evaluate all of the integrals. (Of course, most of these integrals will correspond to hypotheses with little posterior support, such as assigning single data points to isolated clusters, but we don't know ahead of time the relative weight of these assignments.)

In this section, we discuss some ways for learning DAG structure when we have latent variables and/or missing data.

26.5.1 Approximating the marginal likelihood when we have missing data

The simplest approach is to use standard structure learning methods for fully visible DAGs, but to approximate the marginal likelihood. In Section 24.7, we discussed some Monte Carlo methods for approximating the marginal likelihood. However, these are usually too slow to use inside of a search over models. Below we mention some faster deterministic approximations.

26.5.1.1 BIC approximation

A simple approximation is to use the BIC score, which is given by

$$\text{BIC}(G) \triangleq \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \frac{\log N}{2}\dim(G) \tag{26.34}$$

where $\dim(G)$ is the number of degrees of freedom in the model and $\hat{\boldsymbol{\theta}}$ is the MAP or ML estimate. However, the BIC score often severely underestimates the true marginal likelihood (Chickering and Heckerman 1997), resulting in it selecting overly simple models.

26.5.1.2 Cheeseman-Stutz approximation

We now present a better method known as the **Cheeseman-Stutz approximation** (CS) (Cheeseman and Stutz 1996). We first compute a MAP estimate of the parameters $\hat{\boldsymbol{\theta}}$ (e.g., using EM). Denote the expected sufficient statistics of the data by $\overline{\mathcal{D}} = \overline{\mathcal{D}}(\hat{\boldsymbol{\theta}})$; in the case of discrete variables, we just "fill in" the hidden variables with their expectation. We then use the exact marginal likelihood equation on this filled-in data:

$$p(\mathcal{D}|G) \approx p(\overline{\mathcal{D}}|G) = \int p(\overline{\mathcal{D}}|\boldsymbol{\theta}, G)p(\boldsymbol{\theta}|G)d\boldsymbol{\theta} \tag{26.35}$$

However, comparing this to Equation 26.33, we can see that the value will be exponentially smaller, since it does not sum over all values of $\mathbf{h}$. To correct for this, we first write

$$\log p(\mathcal{D}|G) = \log p(\overline{\mathcal{D}}|G) + \log p(\mathcal{D}|G) - \log p(\overline{\mathcal{D}}|G) \tag{26.36}$$

and then we apply a BIC approximation to the last two terms:

$$\begin{aligned}
\log p(\mathcal{D}|G) - \log p(\overline{\mathcal{D}}|G) &\approx \left[\log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \frac{N}{2}\dim(\hat{\boldsymbol{\theta}})\right] && (26.37)\\
&\quad - \left[\log p(\overline{\mathcal{D}}|\hat{\boldsymbol{\theta}}, G) - \frac{N}{2}\dim(\hat{\boldsymbol{\theta}})\right] && (26.38)\\
&= \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \log p(\overline{\mathcal{D}}|\hat{\boldsymbol{\theta}}, G) && (26.39)
\end{aligned}$$

Putting it altogether we get

$$\log p(\mathcal{D}|G) \approx \log p(\overline{\mathcal{D}}|G) + \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \log p(\overline{\mathcal{D}}|\hat{\boldsymbol{\theta}}, G) \tag{26.40}$$

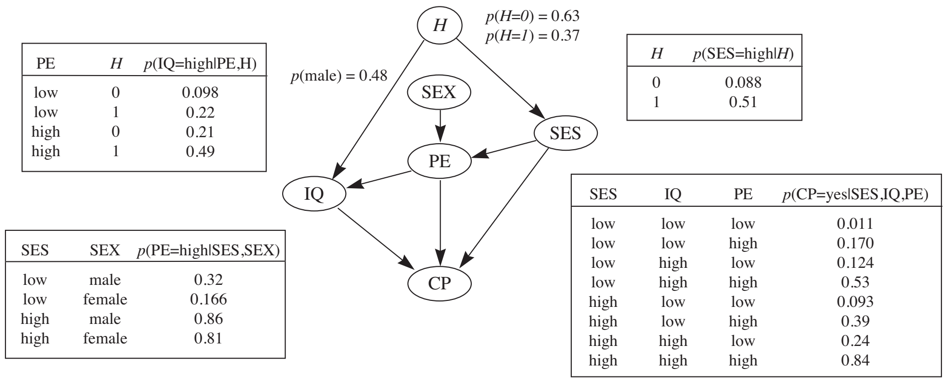
The first term $p(\overline{\mathcal{D}}|G)$ can be computed by plugging in the filled-in data into the exact marginal likelihood. The second term $p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G)$, which involves an exponential sum (thus matching the "dimensionality" of the left hand side) can be computed using an inference algorithm. The final term $p(\overline{\mathcal{D}}|\hat{\boldsymbol{\theta}}, G)$ can be computed by plugging in the filled-in data into the regular likelihood.

26.5.1.3 Variational Bayes EM

An even more accurate approach is to use the variational Bayes EM algorithm. Recall from Section 21.6 that the key idea is to make the following factorization assumption:

$$p(\boldsymbol{\theta}, \mathbf{z}_{1:N}|\mathcal{D}) \approx q(\boldsymbol{\theta})q(\mathbf{z}) = q(\boldsymbol{\theta})\prod_i q(\mathbf{z}_i) \tag{26.41}$$

where $\mathbf{z}_i$ are the hidden variables in case i. In the E step, we update the $q(\mathbf{z}_i)$, and in the M step, we update $q(\boldsymbol{\theta})$. The corresponding variational free energy provides a lower bound on the log marginal likelihood. In (Beal and Ghahramani 2006), it is shown that this bound is a much better approximation to the true log marginal likelihood (as estimated by a slow annealed importance sampling procedure) than either BIC or CS. In fact, one can prove that the variational bound will always be more accurate than CS (which in turn is always more accurate than BIC).



PE	H	p(IQ=high|PE,H)
low	0	0.098
low	1	0.22
high	0	0.21
high	1	0.49

H	p(SES=high|H)
0	0.088
1	0.51

SES	SEX	p(PE=high|SES,SEX)
low	male	0.32
low	female	0.166
high	male	0.86
high	female	0.81

SES	IQ	PE	p(CP=yes|SES,IQ,PE)
low	low	low	0.011
low	low	high	0.170
low	high	low	0.124
low	high	high	0.53
high	low	low	0.093
high	low	high	0.39
high	high	low	0.24
high	high	high	0.84

Figure 26.9 The most probable DAG with a single binary hidden variable learned from the Sewell-Shah data. MAP estimates of the CPT entries are shown for some of the nodes. Source: (Heckerman et al. 1997). Used with kind permission of David Heckerman.

26.5.1.4 Example: college plans revisited

Let us revisit the college plans dataset from Section 26.4.2.5. Recall that if we ignore the possibility of hidden variables there was a direct link from socio economic status to IQ in the MAP DAG. Heckerman et al. decided to see what would happen if they introduced a hidden variable H, which they made a parent of both SES and IQ, representing a hidden common cause. They also considered a variant in which H points to SES, IQ and PE. For both such cases, they considered dropping none, one, or both of the SES-PE and PE-IQ edges. They varied the number of states for the hidden node from 2 to 6. Thus they computed the approximate posterior over $8 \times 5 = 40$ different models, using the CS approximation.

The most probable model which they found is shown in Figure 26.9. This is $2 \cdot 10^{10}$ times more likely than the best model containing no hidden variable. It is also $5 \cdot 10^{9}$ times more likely than the second most probable model with a hidden variable. So again the posterior is very peaked.

These results suggests that there is indeed a hidden common cause underlying both the socio-economic status of the parents and the IQ of the children. By examining the CPT entries, we see that both SES and IQ are more likely to be high when H takes on the value 1. They interpret this to mean that the hidden variable represents "parent quality" (possibly a genetic factor). Note, however, that the arc between H and SES can be reversed without changing the v-structures in the graph, and thus without affecting the likelihood; this underscores the difficulty in interpreting hidden variables.

Interestingly, the hidden variable model has the same conditional independence assumptions amongst the visible variables as the most probable visible variable model. So it is not possible to distinguish between these hypotheses by merely looking at the empirical conditional independencies in the data (which is the basis of the **constraint-based approach** to structure learning (Pearl and Verma 1991; Spirtes et al. 2000)). Instead, by adopting a Bayesian approach, which takes parsimony into account (and not just conditional independence), we can discover the possible existence of hidden factors. This is the basis of much of scientific and everday human reasoning (see e.g. (Griffiths and Tenenbaum 2009) for a discussion).

26.5.2 Structural EM

One way to perform structural inference in the presence of missing data is to use a standard search procedure (deterministic or stochastic), and to use the methods from Section 26.5.1 to estimate the marginal likelihood. However, this approach is very efficient, because the marginal likelihood does not decompose when we have missing data, and nor do its approximations. For example, if we use the CS approximation or the VBEM approximation, we have to perform inference in every neighboring model, just to evaluate the quality of a single move!

(Friedman 1997b; Thiesson et al. 1998) presents a much more efficient approach called the **structural EM** algorithm. The basic idea is this: instead of fitting each candidate neighboring graph and then filling in its data, fill in the data once, and use this filled-in data to evaluate the score of all the neighbors. Although this might be a bad approximation to the marginal likelihood, it can be a good enough approximation of the difference in marginal likelihoods between different models, which is all we need in order to pick the best neighbor.

More precisely, define $\overline{\mathcal{D}}(G_0, \hat{\boldsymbol{\theta}}_0)$ to be the data filled in using model G_0 with MAP parameters $\hat{\boldsymbol{\theta}}_0$. Now define a modified BIC score as follows:

$$\text{score}_{\text{BIC}}(G, \mathcal{D}) \triangleq \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \frac{\log N}{2} \dim(G) + \log p(G) + \log p(\hat{\boldsymbol{\theta}}|G) \tag{26.42}$$

where we have included the log prior for the graph and parameters. One can show (Friedman 1997b) that if we pick a graph G which increases the BIC score relative to G_0 on the expected data, it will also increase the score on the actual data, i.e.,

$$\text{score}_{\text{BIC}}(G, \overline{\mathcal{D}}(G_0, \hat{\boldsymbol{\theta}}_0)) - \text{score}_{\text{BIC}}(G_0, \overline{\mathcal{D}}(G_0, \hat{\boldsymbol{\theta}}_0) \leq \text{score}_{\text{BIC}}(G, \mathcal{D}) - \text{score}_{\text{BIC}}(G_0, \mathcal{D}) \tag{26.43}$$

To convert this into an algorithm, we proceed as follows. First we initialize with some graph G_0 and some set of parameters $\boldsymbol{\theta}_0$. Then we fill-in the data using the current parameters — in practice, this means when we ask for the expected counts for any particular family, we perform inference using our current model. (If we know which counts we will need, we can precompute all of them, which is much faster.) We then evaluate the BIC score of all of our neighbors using the filled-in data, and we pick the best neighbor. We then refit the model parameters, fill-in the data again, and repeat. For increased speed, we may choose to only refit the model every few steps, since small changes to the structure hopefully won't invalidate the parameter estimates and the filled-in data too much.

One interesting application is to learn a phylogenetic tree structure. Here the observed leaves are the DNA or protein sequences of currently alive species, and the goal is to infer the topology of the tree and the values of the missing internal nodes. There are many classical algorithms for this task (see e.g., (Durbin et al. 1998)), but one that uses SEM is discussed in (Friedman et al. 2002).

Another interesting application of this method is to learn sparse mixture models (Barash and Friedman 2002). The idea is that we have one hidden variable C specifying the cluster, and we have to choose whether to add edges $C \to X_t$ for each possible feature X_t. Thus some features will be dependent on the cluster id, and some will be independent. (See also (Law et al. 2004)

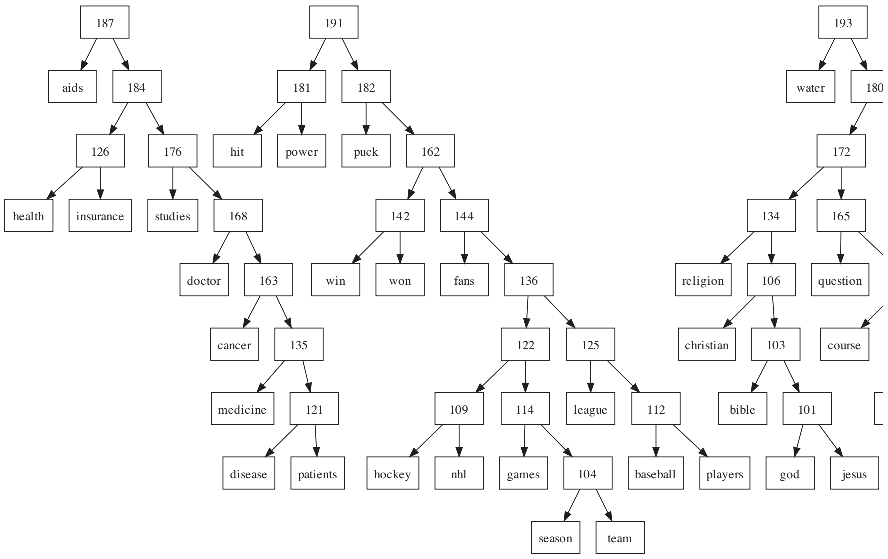


Figure 26.10 Part of a hierarchical latent tree learned from the 20-newsgroup data. From Figure 2 of (Harmeling and Williams 2011). Used with kind permission of Stefan Harmeling.

for a different way to perform this task, using regular EM and a set of bits, one per feature, that are free to change across data cases.)

26.5.3 Discovering hidden variables

In Section 26.5.1.4, we introduced a hidden variable "by hand", and then figured out the local topology by fitting a series of different models and computing the one with the best marginal likelihood. How can we automate this process?

Figure 11.1 provides one useful intuition: if there is a hidden variable in the "true model", then its children are likely to be densely connected. This suggest the following heuristic (Elidan et al. 2000): perform structure learning in the visible domain, and then look for **structural signatures**, such as sets of densely connected nodes (near-cliques); introduce a hidden variable and connect it to all nodes in this near-clique; and then let structural EM sort out the details. Unfortunately, this technique does not work too well, since structure learning algorithms are biased against fitting models with densely connected cliques.

Another useful intuition comes from clustering. In a flat mixture model, also called a **latent class model**, the discrete latent variable provides a compressed representation of its children. Thus we want to create hidden variables with high mutual information with their children.

One way to do this is to create a tree-structured hierarchy of latent variables, each of which only has to explain a small set of children. (Zhang 2004) calls this a **hierarchical latent class model**. They propose a greedy local search algorithm to learn such structures, based on adding or deleting hidden nodes, adding or deleting edges, etc. (Note that learning the optimal latent

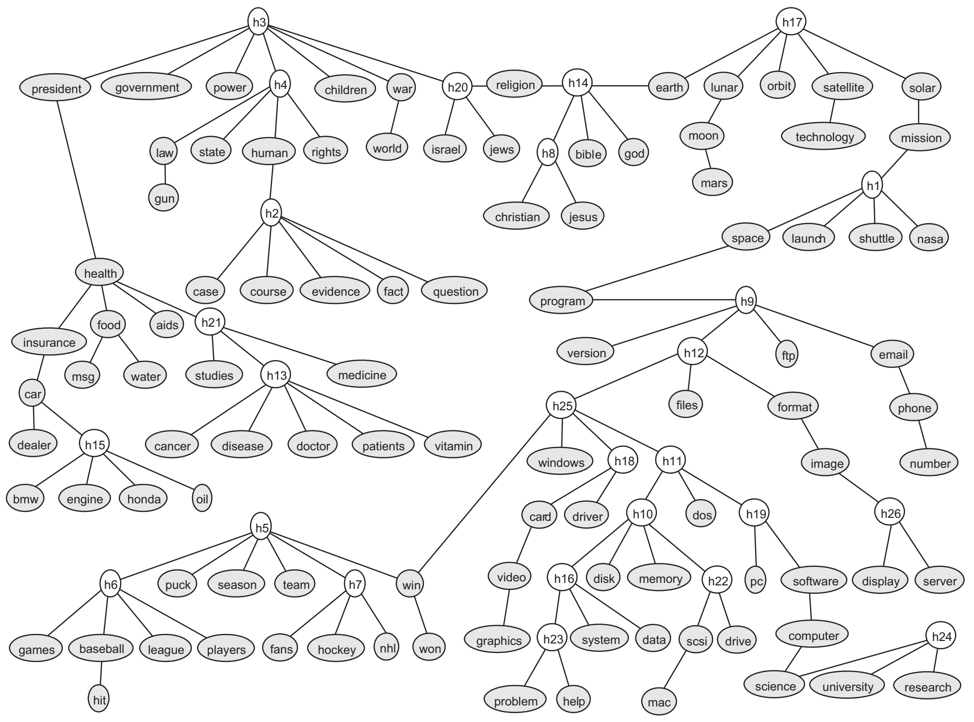


Figure 26.11 A partially latent tree learned from the 20-newsgroup data. Note that some words can have multiple meanings, and get connected to different latent variables, representing different "topics". For example, the word "win" can refer to a sports context (represented by h5) or the Microsoft Windows context (represented by h25). From Figure 12 of (Choi et al. 2011). Used with kind permission of Jin Choi.

tree is NP-hard (Roch 2006).)

Recently (Harmeling and Williams 2011) proposed a faster greedy algorithm for learning such models based on agglomerative hierarchical clustering. Rather than go into details, we just give an example of what this system can learn. Figure 26.10 shows part of a latent forest learned from the 20-newsgroup data. The algorithm imposes the constraint that each latent node has exactly two children, for speed reasons. Nevertheless, we see interpretable clusters arising. For example, Figure 26.10 shows separate clusters concerning medicine, sports and religion. This provides an alternative to LDA and other topic models (Section 4.2.2), with the added advantage that inference in latent trees is exact and takes time linear in the number of nodes.

An alternative approach is proposed in (Choi et al. 2011), in which the observed data is not constrained to be at the leaves. This method starts with the Chow-Liu tree on the observed data, and then adds hidden variables to capture higher-order dependencies between internal nodes. This results in much more compact models, as shown in Figure 26.11. This model also has better predictive accuracy than other approaches, such as mixture models, or trees where all the observed data is forced to be at the leaves. Interestingly, one can show that this method can recover the exact latent tree structure, providing the data is generated from a tree. See

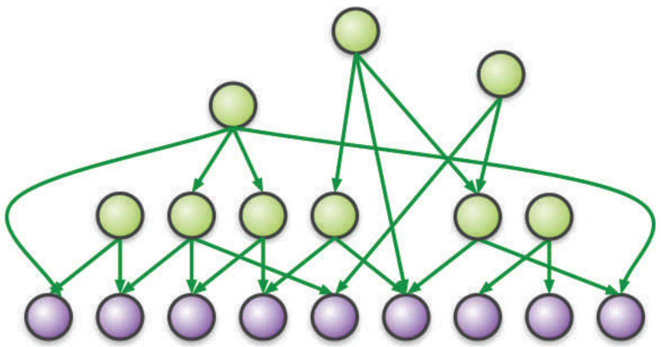


Figure 26.12 Google's rephil model. Leaves represent presence or absence of words. Internal nodes represent clusters of co-occuring words, or "concepts". All nodes are binary, and all CPDs are noisy-OR. The model contains 12 million word nodes, 1 million latent cluster nodes, and 350 million edges. Used with kind permission of Brian Milch.

(Choi et al. 2011) for details. Note, however, that this approach, unlike (Zhang 2004; Harmeling and Williams 2011), requires that the cardinality of all the variables, hidden and observed, be the same. Furthermore, if the observed variables are Gaussian, the hidden variables must be Gaussian also.

26.5.4 Case study: Google's Rephil

In this section, we describe a huge DGM called **Rephil**, which was automatically learned from data.[3] The model is widely used inside Google for various purposes, including their famous AdSense system.[4]

The model structure is shown in Figure 26.12. The leaves are binary nodes, and represent the presence or absence of words or compounds (such as "New York City") in a text document or query. The latent variables are also binary, and represent clusters of co-occuring words. All CPDs are noisy-OR, since some leaf nodes (representing words) can have many parents. This means each edge can be augmented with a hidden variable specifying if the link was activated or not; if the link is not active, then the parent cannot turn the child on. (A very similar model was proposed independently in (Singliar and Hauskrecht 2006).)

Parameter learning is based on EM, where the hidden activation status of each edge needs to be inferred (Meek and Heckerman 1997). Structure learning is based on the old neuroscience

3. The original system, called "Phil", was developed by Georges Harik and Noam Shazeer,. It has been published as US Patent #8024372, "Method and apparatus for learning a probabilistic generative model for text", filed in 2004. Rephil is a more probabilistically sound version of the method, developed by Uri Lerner et al. The summary below is based on notes by Brian Milch (who also works at Google).

4. AdSense is Google's system for matching web pages with content-appropriate ads in an automatic way, by extracting semantic keywords from web pages. These keywords play a role analogous to the words that users type in when searching; this latter form of information is used by Google's AdWords system. The details are secret, but (Levy 2011) gives an overview.

idea that "**nodes that fire together should wire together**". To implement this, we run inference and check for cluster-word and cluster-cluster pairs that frequently turn on together. We then add an edge from parent to child if the link can significantly increase the probability of the child. Links that are not activated very often are pruned out. We initialize with one cluster per "document" (corresponding to a set of semantically related phrases). We then merge clusters A and B if A explains B's top words and vice versa. We can also discard clusters that are used too rarely.

The model was trained on about 100 billion text snippets or search queries; this takes several weeks, even on a parallel distributed computing architecture. The resulting model contains 12 million word nodes and about 1 million latent cluster nodes. There are about 350 million links in the model, including many cluster-cluster dependencies. The longest path in the graph has length 555, so the model is quite deep.

Exact inference in this model is obviously infeasible. However note that most leaves will be off, since most words do not occur in a given query; such leaves can be analytically removed, as shown in Exercise 10.7. We an also prune out unlikely hidden nodes by following the strongest links from the words that are on up to their parents to get a candidate set of concepts. We then perform iterative conditional modes to find a good set of local maxima. At each step of ICM, each node sets its value to its most probable state given the values of its neighbors in its Markov blanket. This continues until it reaches a local maximum. We can repeat this process a few times from random starting configurations. At Google, this can be made to run in 15 milliseconds!

26.5.5 Structural equation models *

A **structural equation model** (Bollen 1989) is a special kind of directed mixed graph (Section 19.4.4.1), possibly cyclic, in which all CPDs are linear Gaussian, and in which all bidirected edges represent correlated Gaussian noise. Such models are also called **path diagrams**. SEMs are widely used, especially in economics and social science. It is common to interpret the edge directions in terms of causality, where directed cycles are interpreted is in terms of **feedback loops** (see e.g., (Pearl 2000, Ch.5)). However, the model is really just a way of specifying a joint Gaussian, as we show below. There is nothing inherently "causal" about it at all. (We discuss causality in Section 26.6.)

We can define an SEM as a series of full conditionals as follows:

$$x_i = \mu_i + \sum_{j \neq i} w_{ij} x_j + \epsilon_i \tag{26.44}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Psi})$. We can rewrite the model in matrix form as follows:

$$\mathbf{x} = \mathbf{W}\mathbf{x} + \boldsymbol{\mu} + \boldsymbol{\epsilon} \Rightarrow \mathbf{x} = (\mathbf{I} - \mathbf{W})^{-1}(\boldsymbol{\epsilon} + \boldsymbol{\mu}) \tag{26.45}$$

Hence the joint distribution is given by $p(\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ where

$$\boldsymbol{\Sigma} = (\mathbf{I} - \mathbf{W})^{-1}\boldsymbol{\Psi}(\mathbf{I} - \mathbf{W})^{-T} \tag{26.46}$$

We draw an arc $X_i \leftarrow X_j$ if $|w_{ij}| > 0$. If $\mathbf{W}$ is lower triangular then the graph is acyclic. If, in addition, $\boldsymbol{\Psi}$ is diagonal, then the model is equivalent to a Gaussian DGM, as discussed in Section 10.2.5; such models are called **recursive**. If $\boldsymbol{\Psi}$ is not diagonal, then we draw a bidirected

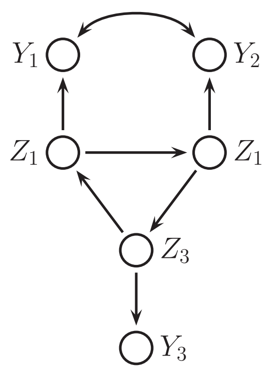


Figure 26.13 A cyclic directed mixed graphical model (non-recursive SEM). Note the $Z_1 \to Z_2 \to Z_3 \to Z_1$ feedback loop.

arc $X_i \leftrightarrow X_j$ for each non-zero off-diagonal term. Such edges represent correlation, possibly due to a hidden common cause.

When using structural equation models, it is common to partition the variables into latent variables, Z_t, and observed or **manifest** variables Y_t. For example, Figure 26.13 illustrates the following model:

$$\begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \end{pmatrix} = \begin{pmatrix} Z_1 \\ Z_2 \\ Z_3 \\ Y_1 \\ Y_2 \\ Y_3 \end{pmatrix} = \begin{pmatrix} 0 & 0 & w_{13} & 0 & 0 & 0 \\ w_{21} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{32} & 0 & 0 & 0 & 0 \\ w_{41} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{52} & 0 & 0 & 0 & 0 \\ 0 & 0 & w_{63} & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} Z_1 \\ Z_2 \\ Z_3 \\ Y_1 \\ Y_2 \\ Y_3 \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \end{pmatrix}, \tag{26.47}$$

where

$$\mathbf{\Psi} = \begin{pmatrix} \Psi_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & \Psi_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & \Psi_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & \Psi_{44} & \Psi_{45} & 0 \\ 0 & 0 & 0 & \Psi_{54} & \Psi_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & \Psi_{66} \end{pmatrix} \tag{26.48}$$

The presence of a feedback loop $Z_1 \to Z_2 \to Z_3$ is evident from the fact that $\mathbf{W}$ is not lower triangular. Also the presence of confounding between Y_1 and Y_2 is evident in the off-diagonal terms in $\mathbf{\Psi}$.

Often we assume there are multiple observations for each latent variable. To ensure identifiability, we can set the mean of the latent variables Z_t to 0, and we can set the regression weights of $Z_t \to Y_t$ to 1. This essentially defines the scale of each latent variable. (In addition to the Z's, there are the extra hidden variables implied by the presence of the bidirected edges.)

The standard practice in the SEM community, as exemplified by the popular commercial software package called **LISREL** (available from `http://www.ssicentral.com/lisrel/`), is to build the structure by hand, to estimate the parameters by maximum likelihood, and then to test if any of the regression weights are significantly different from 0, using standard frequentist methods. However, one can also use Bayesian inference for the parameters (see e.g., (Dunson et al. 2005)). Structure learning in SEMs is rare, but since recursive SEMs are equivalent to Gaussian DAGs, many of the techniques we have been discussing in this section can be applied.

SEMs are closely related to factor analysis (FA) models (Chapter 12). The basic difference is that in an FA model, the latent Gaussian has a low-rank covariance matrix, and the observed noise has a diagonal covariance (hence no bidirected edges). In an SEM, the covariance of the latent Gaussian has a sparse Cholesky decomposition (at least if $\mathbf{W}$ is acyclic), and the observed noise might have a full covariance matrix.

Note that SEMs can be extended in many ways. For example, we can add covariates/ input variables (possibly noisily observed), we can make some of the observations be discrete (e.g., by using probit links), and so on.

26.6 Learning causal DAGs

Causal models are models which can predict the effects of interventions to, or manipulations of, a system. For example, an electronic circuit diagram implicitly provides a compact encoding of what will happen if one removes any given component, or cuts any wire. A causal medical model might predict that if I continue to smoke, I am likely to get lung cancer (and hence if I cease smoking, I am less likely to get lung cancer). Causal claims are inherently stronger, yet more useful, than purely **associative** claims, such as "people who smoke often have lung cancer".

Causal models are often represented by DAGs (Pearl 2000), although this is somewhat controversial (Dawid 2010). We explain this causal interpretation of DAGs below. We then show how to use a DAG to do causal reasoning. Finally, we briefly discuss how to learn the structure of causal DAGs. A more detailed description of this topic can be found in (Pearl 2000) and (Koller and Friedman 2009, Ch.21).

26.6.1 Causal interpretation of DAGs

In this section, we define a directed edge $A \rightarrow B$ in a DAG to mean that "A directly causes B", so if we manipulate A, then B will change. This is known as the **causal Markov assumption**. (Of course, we have not defined the word "causes", and we cannot do that by appealing to a DAG, lest we end up with a cyclic definition; see (Dawid 2010) for further disussion of this point.)

We will also assume that all relevant variables are included in the model, i.e., there are no unknown **confounders**, reflecting hidden common causes. This is called the **causal sufficiency** assumption. (If there are known to be confounders, they should be added to the model, although one can sometimes use mixed directed graphs (Section 26.5.5) as a way to avoid having to model confounders explicitly.)

Assuming we are willing to make the causal Markov and causal sufficiency assumptions, we can use DAGs to answer causal questions. The key abstraction is that of a **perfect intervention**; this represents the act of setting a variable to some known value, say setting X_i to x_i. A real world example of such a perfect intervention is a gene knockout experiment, in which a gene is "silenced". We need some notational convention to distinguish this from observing that X_i

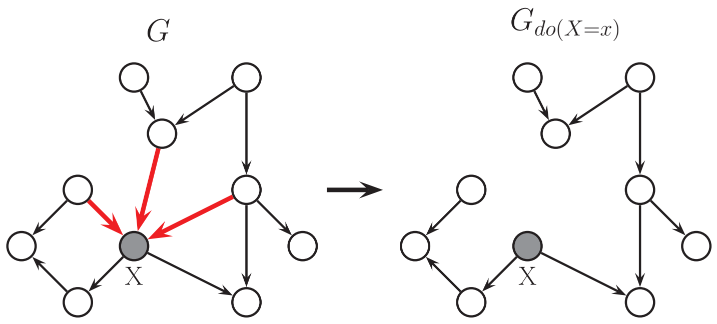


Figure 26.14 Surgical intervention on X. Based on (Pe'er 2005).

happens to have value x_i. We use Pearl's **do calculus** notation (as in the verb "to do") and write $\text{do}(X_i = x_i)$ to denote the event that we set X_i to x_i. A causal model can be used to make inferences of the form $p(\mathbf{x}|\text{do}(X_i = x_i))$, which is different from making inferences of the form $p(\mathbf{x}|X_i = x_i)$.

To understand the difference between conditioning on interventions and conditioning on observations (i.e., the difference between doing and seeing), consider a 2 node DGM $S \rightarrow Y$, in which $S = 1$ if you smoke and $S = 0$ otherwise, and $Y = 1$ if you have yellow-stained fingers, and $Y = 0$ otherwise. If I observe you have yellow fingers, I am licensed to infer that you are probably a smoker (since nicotine causes yellow stains):

$$p(S = 1|Y = 1) > p(S = 1) \tag{26.49}$$

However, if I intervene and *paint* your fingers yellow, I am no longer licensed to infer this, since I have disrupted the normal causal mechanism. Thus

$$p(S = 1|\text{do}(Y = 1)) = p(S = 1) \tag{26.50}$$

One way to model perfect interventions is to use **graph surgery**: represent the joint distribution by a DGM, and then cut the arcs coming into any nodes that were set by intervention. See Figure 26.14 for an example. This prevents any information flow from the nodes that were intervened on from being sent back up to their parents. Having perform this surgery, we can then perform probabilistic inference in the resulting "mutilated" graph in the usual way to reason about the effects of interventions. We state this formally as follows.

Theorem 26.6.1 (Manipulation theorem (Pearl 2000; Spirtes et al. 2000))**.** . *To compute* $p(X_i|\text{do}(X_j))$ *for sets of nodes* i, j, *we can perform surgical intervention on the* X_j *nodes and then use standard probabilistic inference in the mutilated graph.*

We can generalize the notion of a perfect intervention by adding interventions as explicit action nodes to the graph. The result is like an influence diagram, except there are no utility nodes (Lauritzen 2000; Dawid 2002). This has been called the **augmented DAG** (Pearl 2000). We

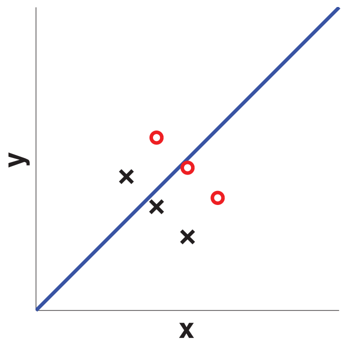


Figure 26.15 Illustration of Simpson's paradox. Figure generated by `simpsonsParadoxGraph`.

can then define the CPD $p(X_i|\text{do}(X_i))$ to be anything we want. We can also allow an action to affect multiple nodes. This is called a **fat hand** intervention, a reference to someone trying to change a single component of some system (e.g., an electronic circuit), but accidently touching multiple components and thereby causing various side effects (see (Eaton and Murphy 2007) for a way to model this using augmented DAGs).

26.6.2 Using causal DAGs to resolve Simpson's paradox

In this section, we assume we know the causal DAG. We can then do causal reasoning by applying d-separation to the mutilated graph. In this section, we give an example of this, and show how causal reasoning can help resolve a famous paradox, known as **Simpon's paradox**.

Simpson's paradox says that any statistical relationship between two variables can be reversed by including additional factors in the analysis. For example, suppose some cause C (say, taking a drug) makes some effect E (say getting better) more likely

$$P(E|C) \quad > \quad P(E|\neg C)$$

and yet, when we condition on the gender of the patient, we find that taking the drug makes the effect less likely in both females (F) and males ($\neg F$):

$$\begin{aligned} P(E|C,F) \quad &< \quad P(E|\neg C,F) \\ P(E|C,\neg F) \quad &< \quad P(E|\neg C,\neg F) \end{aligned}$$

This seems impossible, but by the rules of probability, this is perfectly possible, because the event space where we condition on $(\neg C, F)$ or $(\neg C, \neg F)$ can be completely different to the event space when we just condition on $\neg C$. The table of numbers below shows a concrete example (from (Pearl 2000, p175)):

	Combined				Male				Female			
	E	$\neg E$	Total	Rate	E	$\neg E$	Total	Rate	E	$\neg E$	Total	Rate
C	20	20	40	50%	18	12	30	60%	2	8	10	20%
$\neg C$	16	24	40	40%	7	3	10	70%	9	21	30	30%
Total	36	44	80		25	15	40		11	29	40	

From this table of numbers, we see that

$$p(E|C) = 20/40 = 0.5 \quad > \quad p(E|\neg C) = 16/40 = 0.4 \tag{26.51}$$

$$p(E|C, F) = 2/10 = 0.2 \quad < \quad p(E|\neg C, F) = 9/30 = 0.3 \tag{26.52}$$

$$p(E|C, \neg F) = 18/30 = 0.6 \quad < \quad p(E|\neg, \neg F) = 7/10 = 0.7 \tag{26.53}$$

A visual representation of the paradox is given in in Figure 26.15. The line which goes up and to the right shows that the effect (y-axis) increases as the cause (x-axis) increases. However, the dots represent the data for females, and the crosses represent the data for males. Within each subgroup, we see that the effect decreases as we increase the cause.

It is clear that the effect is real, but it is still very counter-intuitive. The reason the paradox arises is that we are interpreting the statements causally, but we are not using proper causal reasoning when performing our calculations. The statement that the drug C causes recovery E is

$$P(E|\text{do}(C)) \quad > \quad P(E|\text{do}(\neg C)) \tag{26.54}$$

whereas the data merely tell us

$$P(E|C) \quad > \quad P(E|\neg C) \tag{26.55}$$

This is not a contradiction. Observing C is positive evidence for E, since more males than females take the drug, and the male recovery rate is higher (regardless of the drug). Thus Equation 26.55 does not imply Equation 26.54.

Nevertheless, we are left with a practical question: should we use the drug or not? It seems like if we don't know the patient's gender, we should use the drug, but as soon as we discover if they are male or female, we should stop using it. Obviously this conclusion is ridiculous.

To answer the question, we need to make our assumptions more explicit. Suppose reality can be modeled by the causal DAG in Figure 26.16(a). To compute the causal effect of C on E, we need to **adjust for** (i.e., condition on) the **confounding variable** F. This is necessary because there is a **backdoor path** from C to E via F, so we need to check the $C \rightarrow E$ relationship for each value of F separately, to make sure the relationship between C and E is not affected by any value of F.

Suppose that for each value of F, taking the drug is harmful, that is,

$$p(E|\text{do}(C), F) \quad < \quad p(E|\text{do}(\neg C), F) \tag{26.56}$$

$$p(E|\text{do}(C), \neg F) \quad < \quad p(E|\text{do}(\neg C), \neg F) \tag{26.57}$$

Then we can show that taking the drug is harmful overall:

$$p(E|\text{do}(C)) < p(E|\text{do}(\neg C)) \tag{26.58}$$

The proof is as follows (Pearl 2000, p181). First, from our assumptions in Figure 26.16(a), we see that drugs have no effect on gender

$$p(F|\text{do}(C)) = p(F|\text{do}(\neg C)) = p(F) \tag{26.59}$$

Now using the law of total probability,

$$p(E|\text{do}(C)) \quad = \quad p(E|\text{do}(C), F)p(F|\text{do}(C)) + p(E|\text{do}(C), \neg F)p(\neg F|\text{do}(C)) \tag{26.60}$$

$$= \quad p(E|\text{do}(C), F)p(F) + p(E|\text{do}(C), \neg F)p(\neg F) \tag{26.61}$$

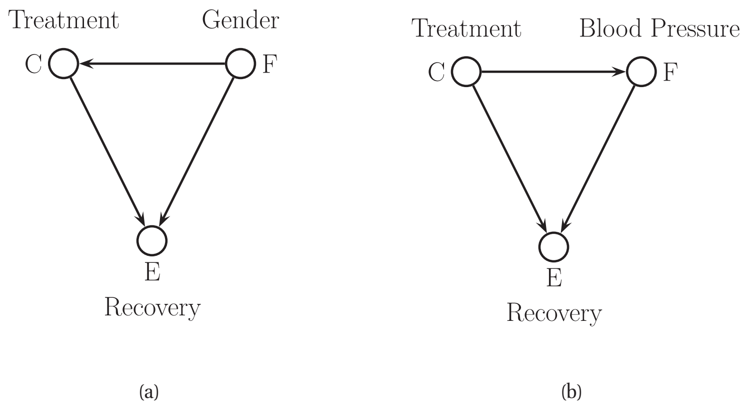


Figure 26.16 Two different models uses to illustrate Simpson's paradox. (a) F is gender and is a confounder for C and E. (b) F is blood pressure and is caused by C.

Similarly,

$$p(E|\text{do}(\neg C)) \quad = \quad p(E|\text{do}(\neg C), F)p(F) + p(E|\text{do}(\neg C), \neg F)p(\neg F) \tag{26.62}$$

Since every term in Equation 26.61 is less than the corresponding term in Equation 26.62, we conclude that

$$p(E|\text{do}(C)) < p(E|\text{do}(\neg C)) \tag{26.63}$$

So if the model in Figure 26.16(a) is correct, we should not administer the drug, since it reduces the probability of the effect.

Now consider a different version of this example. Suppose we keep the data the same but interpret F as something that is affected by C, such as blood pressure. See Figure 26.16(b). In this case, we can no longer assume

$$p(F|\text{do}(C)) = p(F|\text{do}(\neg C)) = p(F) \tag{26.64}$$

and the above proof breaks down. So $p(E|\text{do}(C)) - p(E|\text{do}(\neg C))$ may be positive or negaitve.

In the true model is Figure 26.16(b), then we should not condition on F when assessing the effect of C on E, since there is no backdoor path in this case, because of the v-structure at F. That is, conditioning on F might block one of the causal pathways. In other words, by comparing patients with the same post-treatment blood pressure (value of F), we may mask the effect of one of the two pathways by which the drug operates to bring about recovery.

Thus we see that different causal assumptions lead to different causal conclusions, and hence different courses of action. This raises the question on whether we can learn the causal model from data. We discuss this issue below.

26.6.3 Learning causal DAG structures

In this section, we discuss some ways to learn causal DAG structures.

26.6.3.1 Learning from observational data

In Section 26.4, we discussed various methods for learning DAG structures from observational data. It is natural to ask whether these methods can recover the "true" DAG structure that was used to generate the data. Clearly, even if we have infinite data, an optimal method can only identify the DAG up to Markov equivalence (Section 26.4.1). That is, it can identify the PDAG (partially directed acylic graph), but not the complete DAG structure, because all DAGs which are Markov equivalent have the same likelihood.

There are several algorithms (e.g., the **greedy equivalence search** method of (Chickering 2002)) that are consistent estimators of PDAG structure, in the sense that they identify the true Markov equivalence class as the sample size goes to infinity, assuming we observe all the variables. However, we also have to assume that the generating distribution p is **faithful** to the generating DAG G. This means that all the conditional indepence (CI) properties of p are exactly captured by the graphical structure, so $I(p) = I(G)$; this means there cannot be any CI properties in p that are due to particular settings of the parameters (such as zeros in a regression matrix) that are not graphically explicit. For this reason, a faithful distribution is also called a **stable** distribution.

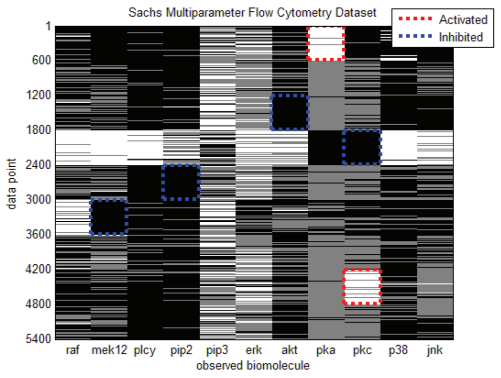
Suppose the assumptions hold and we learn a PDAG. What can we do with it? Instead of recovering the full graph, we can focus on the causal analog of edge marginals, by computing the magnitude of the causal effect of one node on another (say A on B). If we know the DAG, we can do this using techniques described in (Pearl 2000). If the DAG is unknown, we can compute a lower bound on the effect as follows (Maathuis et al. 2009): learn an equivalence class (PDAG) from data; enumerate all the DAGs in the equivalence class; apply Pearl's do-calculus to compute the magnitude of the causal effect of A on B in each DAG; finally, take the minimum of these effects as the lower bound. It is usually computationally infeasible to compute all DAGs in the equivalence class, but fortunately one only needs to be able to identify the local neighborhood of A and B, which can be esimated more efficiently, as described in (Maathuis et al. 2009). This technique is called **IDA**, which is short for "intervention-calculus when the DAG is absent".

In (Maathuis et al. 2010), this technique was applied to some yeast gene expression data. Gene knockout data was used to estimate the "ground truth" effect of each 234 single-gene deletions on the remaining 5,361 genes. Then the algorithm was applied to 63 unperturbed (wild-type) samples, and was used to rank order the likely targets of each of the 234 genes. The method had a precision of 66% when the recall was set to 10%; while low, this is substantially more than rival variable-selection methods, such as lasso and elastic net, which were only slightly above chance.

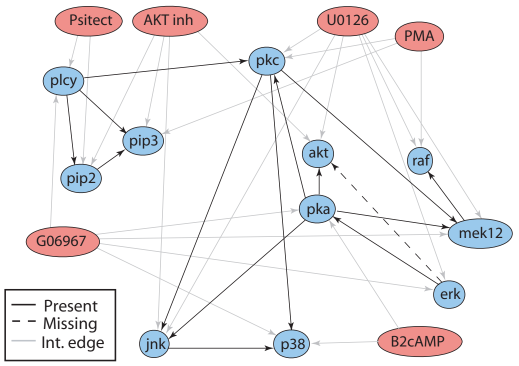
26.6.3.2 Learning from interventional data

If we want to distinguish between DAGs within the equivalence class, we need to use **interventional data**, where certain variables have been set, and the consequences have been measured. An example of this is the dataset in Figure 26.17(a), where proteins in a signalling pathway were perturbed, and their phosphorylation status was measured using a technique called flow cytometry (Sachs et al. 2005).

It is straightforward to modify the standard Bayesian scoring criteria, such as the marginal likelihood or BIC score, to handle learning from mixed observational and experimental data: we



(a)



(b)

Figure 26.17 (a) A design matrix consisting of 5400 data points (rows) measuring the status (using flow cytometry) of 11 proteins (columns) under different experimental conditions. The data has been discretized into 3 states: low (black), medium (grey) and high (white). Some proteins were explicitly controlled using activating or inhibiting chemicals. (b) A directed graphical model representing dependencies between various proteins (blue circles) and various experimental interventions (pink ovals), which was inferred from this data. We plot all edges for which $p(G_{st} = 1|\mathcal{D}) > 0.5$. Dotted edges are believed to exist in nature but were not discovered by the algorithm (1 false negative). Solid edges are true positives. The light colored edges represent the effects of intervention. Source: Figure 6d of (Eaton and Murphy 2007) . This figure can be reproduced using the code at `http://www.cs.ubc.ca/~murphyk/Software/BDAGL/index.html`.

just compute the sufficient statistics for a CPD's parameter by skipping over the cases where that node was set by intervention (Cooper and Yoo 1999). For example, when using tabular CPDs, we modify the counts as follows:

$$N_{tck} \triangleq \sum_{i:x_{it} \text{ not set}} \mathbb{I}(x_{i,t} = k, \mathbf{x}_{i,\text{pa}(t)} = c) \tag{26.65}$$

The justification for this is that in cases where node t is set by force, it is not sampled from its usual mechanism, so such cases should be ignored when inferring the parameter $\boldsymbol{\theta}_t$. The modified scoring criterion can be combined with any of the standard structure learning algorithms. (He and Geng 2009) discusses some methods for choosing which interventions to perform, so as to reduce the posterior uncertainty as quickly as possible (a form of active learning).

The preceeding method assumes the interventions are perfect. In reality, experimenters can rarely control the state of individual molecules. Instead, they inject various stimulant or inhibitor chemicals which are designed to target specific molecules, but which may have side effects. We can model this quite simply by adding the intervention nodes to the DAG, and then learning a larger augmented DAG structure, with the constraint that there are no edges between the intervention nodes, and no edges from the "regular" nodes back to the intervention nodes.

Figure 26.17(b) shows the augmented DAG that was learned from the interventional flow cytometry data depicted in Figure 26.17(a). In particular, we plot the median graph, which includes all edges for which $p(G_{ij} = 1|\mathcal{D}) > 0.5$. These were computed using the exact algorithm of (Koivisto 2006). It turns out that, in this example, the median model has exactly the same structure as the optimal MAP model, $\text{argmax}_G \, p(G|\mathcal{D})$, which was computed using the algorithm of (Koivisto and Sood 2004; Silander and Myllmaki 2006).

26.7 Learning undirected Gaussian graphical models

Learning the structured of undirected graphical models is easier than learning DAG structure because we don't need to worry about acyclicity. On the other hand, it is harder than learning DAG structure since the likelihood does not decompose (see Section 19.5). This precludes the kind of local search methods (both greedy search and MCMC sampling) we used to learn DAG structures, because the cost of evaluating each neighboring graph is too high, since we have to refit each model from scratch (there is no way to incrementally update the score of a model).

In this section, we discuss several solutions to this problem, in the context of **Gaussian random fields** or undirected Gaussian graphical models (GGM)s. We consider structure learning for discrete undirected models in Section 26.8.

26.7.1 MLE for a GGM

Before discussing structure learning, we need to discuss parameter estimation. The task of computing the MLE for a (non-decomposable) GGM is called **covariance selection** (Dempster 1972).

From Equation 4.19, the log likelihood can be written as

$$\ell(\boldsymbol{\Omega}) = \log \det \boldsymbol{\Omega} - \text{tr}(\mathbf{S}\boldsymbol{\Omega}) \tag{26.66}$$

where $\mathbf{\Omega} = \mathbf{\Sigma}^{-1}$ is the precision matrix, and $\mathbf{S} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x}_i - \overline{\mathbf{x}})^T$ is the empirical covariance matrix. (For notational simplicity, we assume we have already estimated $\hat{\boldsymbol{\mu}} = \overline{\mathbf{x}}$.) One can show that the gradient of this is given by

$$\nabla \ell(\mathbf{\Omega}) = \mathbf{\Omega}^{-1} - \mathbf{S} \tag{26.67}$$

However, we have to enforce the constraints that $\Omega_{st} = 0$ if $G_{st} = 0$ (structural zeros), and that $\mathbf{\Omega}$ is positive definite. The former constraint is easy to enforce, but the latter is somewhat challenging (albeit still a convex constraint). One approach is to add a penalty term to the objective if $\mathbf{\Omega}$ leaves the positive definite cone; this is the approach used in `ggmFitMinfunc` (see also (Dahl et al. 2008)). Another approach is to use a coordinate descent method, described in (Hastie et al. 2009, p633), and implemented in `ggmFitHtf`. Yet another approach is to use iterative proportional fitting, described in Section 19.5.7. However, IPF requires identifying the cliques of the graph, which is NP-hard in general.

Interestingly, one can show that the MLE must satisfy the following property: $\Sigma_{st} = S_{st}$ if $G_{st} = 1$ or $s = t$, i.e., the covariance of a pair that are connected by an edge must match the empirical covariance. In addition, we have $\Omega_{st} = 0$ if $G_{st} = 0$, by definition of a GGM, i.e., the precision of a pair that are not connected must be 0. We say that $\mathbf{\Sigma}$ is a positive definite **matrix completion** of $\mathbf{S}$, since it retains as many of the entries in $\mathbf{S}$ as possible, corresponding to the edges in the graph, subject to the required sparsity pattern on $\mathbf{\Sigma}^{-1}$, corresponding to the absent edges; the remaining entries in $\mathbf{\Sigma}$ are filled in so as to maximize the likelihood.

Let us consider a worked example from (Hastie et al. 2009, p652). We will use the following adjacency matrix, representing the cyclic structure, $X_1 - X_2 - X_3 - X_4 - X_1$, and the following empirical covariance matrix:

$$\mathbf{G} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \quad \mathbf{S} = \begin{pmatrix} 10 & 1 & 5 & 4 \\ 1 & 10 & 2 & 6 \\ 5 & 2 & 10 & 3 \\ 4 & 6 & 3 & 10 \end{pmatrix} \tag{26.68}$$

The MLE is given by

$$\mathbf{\Sigma} = \begin{pmatrix} 10.00 & 1.00 & \mathbf{1.31} & 4.00 \\ 1.00 & 10.00 & 2.00 & \mathbf{0.87} \\ \mathbf{1.31} & 2.00 & 10.00 & 3.00 \\ 4.00 & \mathbf{0.87} & 3.00 & 10.00 \end{pmatrix}, \quad \mathbf{\Omega} = \begin{pmatrix} 0.12 & -0.01 & \mathbf{0} & -0.05 \\ -0.01 & 0.11 & -0.02 & \mathbf{0} \\ \mathbf{0} & -0.02 & 0.11 & -0.03 \\ -0.05 & \mathbf{0} & -0.03 & 0.13 \end{pmatrix} \tag{26.69}$$

(See `ggmFitDemo` for the code to reproduce these numbers.) The constrained elements in $\mathbf{\Omega}$, and the free elements in $\mathbf{\Sigma}$, both of which correspond to absent edges, have been highlighted.

26.7.2 Graphical lasso

We now discuss one way to learn a sparse GRF structure, which exploits the fact that there is a 1:1 correspondence between zeros in the precision matrix and absent edges in the graph. This suggests that we can learn a sparse graph structure by using an objective that encourages zeros in the precision matrix. By analogy to lasso (see Section 13.3), one can define the following ℓ_1 penalized NLL:

$$J(\mathbf{\Omega}) = -\log\det\mathbf{\Omega} + \mathrm{tr}(\mathbf{S}\mathbf{\Omega}) + \lambda||\mathbf{\Omega}||_1 \tag{26.70}$$

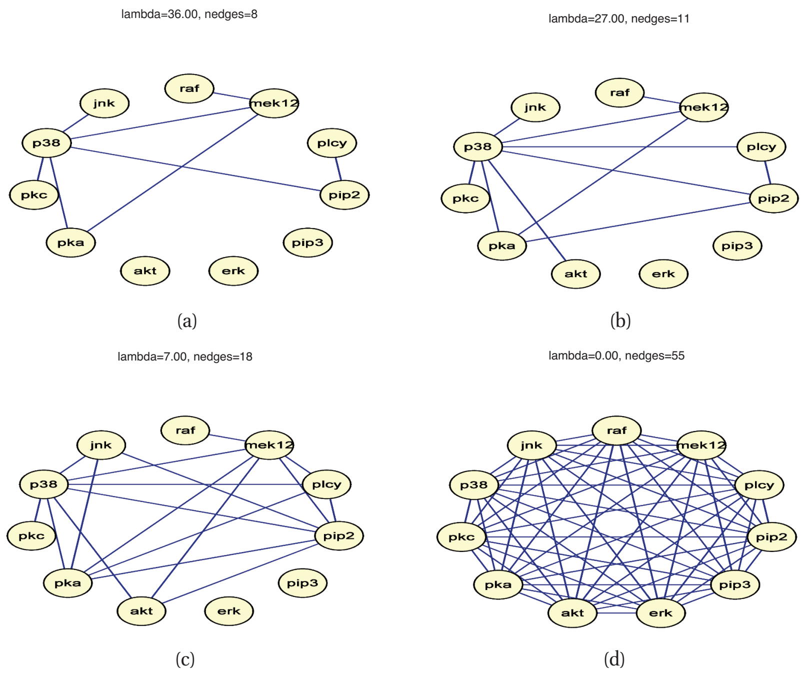


Figure 26.18 Sparse GGMs learned using graphical lasso applied to the flow cytometry data. (a) $\lambda = 36$. (b) $\lambda = 27$. (c) $\lambda = 7$. (d) $\lambda = 0$. Figure generated by `ggmLassoDemo`.

where $||\mathbf{\Omega}||_1 = \sum_{j,k} |\omega_{jk}|$ is the 1-norm of the matrix. This is called the **graphical lasso** or **Glasso**.

Although the objective is convex, it is non-smooth (because of the non-differentiable ℓ_1 penalty) and is constrained (because $\mathbf{\Omega}$ must be a positive definite matrix). Several algorithms have been proposed for optimizing this objective (Yuan and Lin 2007; Banerjee et al. 2008; Duchi et al. 2008), although arguably the simplest is the one in (Friedman et al. 2008), which uses a coordinate descent algorithm similar to the shooting algorithm for lasso. See `ggmLassoHtf` for an implementation. (See also (Mazumder and Hastie 2012) for a more recent version of this algorithm.)

As an example, let us apply the method to the flow cytometry dataset from (Sachs et al. 2005). A discretized version of the data is shown in Figure 26.17(a). Here we use the original continuous data. However, we are ignoring the fact that the data was sampled under intervention. In Figure 26.18, we illustrate the graph structures that are learned as we sweep λ from 0 to a large value. These represent a range of plausible hypotheses about the connectivity of these proteins.

It is worth comparing this with the DAG that was learned in Figure 26.17(b). The DAG has the advantage that it can easily model the interventional nature of the data, but the disadvantage that it cannot model the feedback loops that are known to exist in this biological pathway (see the discussion in (Schmidt and Murphy 2009)). Note that the fact that we show many UGMs and only one DAG is incidental: we could easily use BIC to pick the "best" UGM, and conversely, we could easily display several DAG structures, sampled from the posterior.

26.7.3 Bayesian inference for GGM structure *

Although the graphical lasso is reasonably fast, it only gives a point estimate of the structure. Furthermore, it is not model-selection consistent (Meinshausen 2005), meaning it cannot recover the true graph even as $N \to \infty$. It would be preferable to integrate out the parameters, and perform posterior inference in the space of graphs, i.e., to compute $p(G|\mathcal{D})$. We can then extract summaries of the posterior, such as posterior edge marginals, $p(G_{ij} = 1|\mathcal{D})$, just as we did for DAGs. In this section, we discuss how to do this.

Note that the situation is analogous to Chapter 13, where we discussed variable selection. In Section 13.2, we discussed Bayesian variable selection, where we integrated out the regression weights and computed $p(\boldsymbol{\gamma}|\mathcal{D})$ and the marginal inclusion probabilities $p(\gamma_j = 1|\mathcal{D})$. Then in Section 13.3, we discussed methods based on ℓ_1 regularization. Here we have the same dichotomy, but we are presenting them in the opposite order.

If the graph is decomposable, and if we use conjugate priors, we can compute the marginal likelihood in closed form (Dawid and Lauritzen 1993). Furthermore, we can efficiently identify the decomposable neighbors of a graph (Thomas and Green 2009), i.e., the set of legal edge additions and removals. This means that we can perform relatively efficient stochastic local search to approximate the posterior (see e.g. (Giudici and Green 1999; Armstrong et al. 2008; Scott and Carvalho 2008)).

However, the restriction to decomposable graphs is rather limiting if one's goal is knowledge discovery, since the number of decomposable graphs is much less than the number of general undirected graphs.[5]

A few authors have looked at Bayesian inference for GGM structure in the non-decomposable case (e.g., (Dellaportas et al. 2003; Wong et al. 2003; Jones et al. 2005)), but such methods cannot scale to large models because they use an expensive Monte Carlo approximation to the marginal likelihood (Atay-Kayis and Massam 2005). (Lenkoski and Dobra 2008) suggested using a Laplace approxmation. This requires computing the MAP estimate of the parameters for $\boldsymbol{\Omega}$ under a G-Wishart prior (Roverato 2002). In (Lenkoski and Dobra 2008), they used the iterative proportional scaling algorithm (Speed and Kiiveri 1986; Hara and Takimura 2008) to find the mode. However, this is very slow, since it requires knowing the maximal cliques of the graph, which is NP-hard in general.

In (Moghaddam et al. 2009), a much faster method is proposed. In particular, they modify the gradient-based methods from Section 26.7.1 to find the MAP estimate; these algorithms do not need to know the cliques of the graph. A further speedup is obtained by just using a diagonal Laplace approximation, which is more accurate than BIC, but has essentially the same cost. This, plus the lack of restriction to decomposable graphs, enables fairly fast stochastic search methods to be used to approximate $p(G|\mathcal{D})$ and its mode. This approach significantly outperfomed graphical lasso, both in terms of predictive accuracy and structural recovery, for a comparable computational cost.

5. The number of decomposable graphs on V nodes, for $V = 2, \ldots, 8$, is as follows ((Armstrong 2005, p158)): 2; 8; 61; 822; 18,154; 61,7675; 30,888,596. If we divide these numbers by the number of undirected graphs, which is $2^{V(V-1)/2}$, we find the ratios are: 1, 1, 0.95, 0.8, 0.55, 0.29, 0.12. So we see that decomposable graphs form a vanishing fraction of the total hypothesis space.

26.7.4 Handling non-Gaussian data using copulas *

The graphical lasso and variants is inhertently limited to data that is jointly Gaussian, which is a rather severe restriction. Fortunately the method can be generalized to handle non-Gaussian, but still continuous, data in a fairly simple fashion. The basic idea is to estimate a set of D univariate monotonic transformations f_j, one per variable j, such that the resulting transformed data is jointly Gaussian. If this is possible, we say the data belongs to the nonparametric Normal distribution, or **nonparanormal** distribution (Liu et al. 2009). This is equivalent to the family of **Gaussian copulas** (Klaassen and Wellner 1997). Details on how to estimate the f_j transformations from the empirical cdf's of each variable can be found in (Liu et al. 2009). After transforming the data, we can compute the correlation matrix and then apply glasso in the usual way. One can show, under various assumptions, that this is a consistent estimator of the graph structure, representing the CI assumptions of the original distribution(Liu et al. 2009).

26.8 Learning undirected discrete graphical models

The problem of learning the structure for UGMs with discrete variables is harder than the Gaussian case, because computing the partition function $Z(\boldsymbol{\theta})$, which is needed for parameter estimation, has complexity comparable to computing the permanent of a matrix, which in general is intractable (Jerrum et al. 2004). By contrast, in the Gaussian case, computing Z only requires computing a matrix determinant, which is at most $O(V^3)$.

Since stochastic local search is not tractable for general discrete UGMs, below we mention some possible alternative approaches that have been tried.

26.8.1 Graphical lasso for MRFs/CRFs

It is possible to extend the graphical lasso idea to the discrete MRF and CRF case. However, now there is a set of parameters associated with each edge in the graph, so we have to use the graph analog of group lasso (see Section 13.5.1). For example, consider a pairwise CRF with ternary nodes, and node and edge potentials given by

$$\psi_t(y_t, \mathbf{x}) = \begin{pmatrix} \mathbf{v}_{t1}^T\mathbf{x} \\ \mathbf{v}_{t2}^T\mathbf{x} \\ \mathbf{v}_{t3}^T\mathbf{x} \end{pmatrix}, \ \psi_{st}(y_s, y_t, \mathbf{x}) = \begin{pmatrix} \mathbf{w}_{t11}^T\mathbf{x} & \mathbf{w}_{st12}^T\mathbf{x} & \mathbf{w}_{st13}^T\mathbf{x} \\ \mathbf{w}_{st21}^T\mathbf{x} & \mathbf{w}_{st22}^T\mathbf{x} & \mathbf{w}_{st23}^T\mathbf{x} \\ \mathbf{w}_{st31}^T\mathbf{x} & \mathbf{w}_{st32}^T\mathbf{x} & \mathbf{w}_{st33}^T\mathbf{x} \end{pmatrix} \tag{26.71}$$

where we assume $\mathbf{x}$ begins with a constant 1 term, to account for the offset. (If $\mathbf{x}$ only contains 1, the CRF reduces to an MRF.) Note that we may choose to set some of the $\mathbf{v}_{tk}$ and $\mathbf{w}_{stjk}$ weights to 0, to ensure identifiability, although this can also be taken care of by the prior, as shown in Exercise 8.5.

To learn sparse structure, we can minimize the following objective:

$$\begin{aligned} J &= -\sum_{i=1}^{N}\left[\sum_{t}\log\psi_t(y_{it}, \mathbf{x}_i, \mathbf{v}_t) + \sum_{s=1}^{V}\sum_{t=s+1}^{V}\log\psi_{st}(y_{is}, y_{it}, \mathbf{x}_i, \mathbf{w}_{st})\right] \\ &\quad +\lambda_1\sum_{s=1}^{V}\sum_{t=s+1}^{V}||\mathbf{w}_{st}||_p + \lambda_2\sum_{t=1}^{V}||\mathbf{v}_t||_2^2 \end{aligned} \tag{26.72}$$

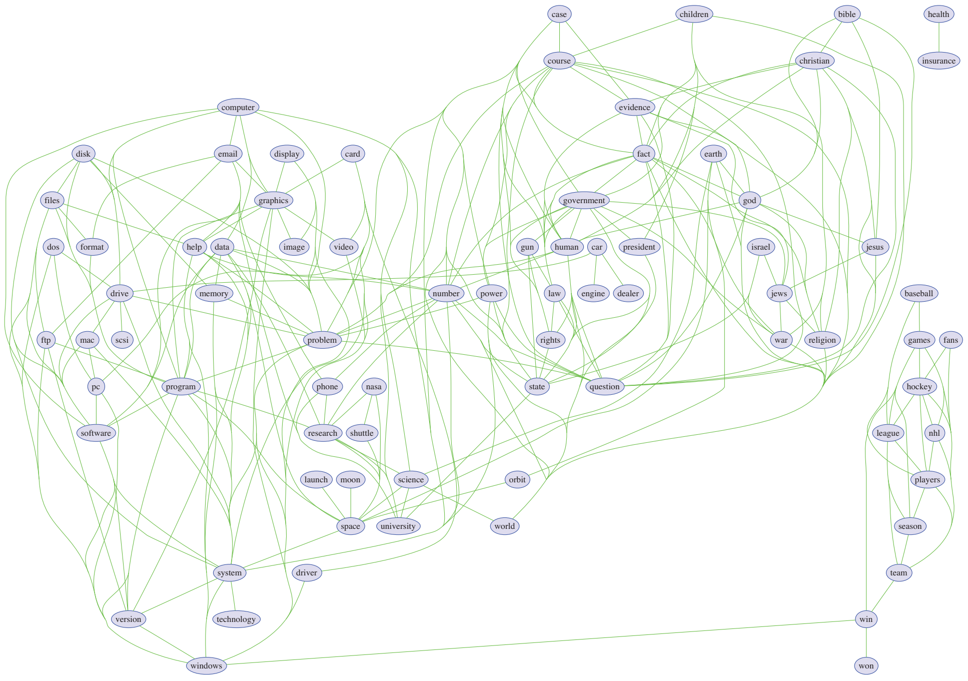
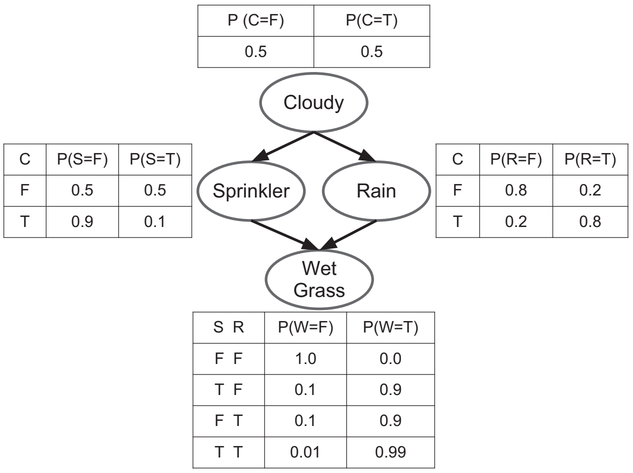


Figure 26.19 An MRF estimated from the 20-newsgroup data using group ℓ_1 regularization with $\lambda = 256$. Isolated nodes are not plotted. From Figure 5.9 of (Schmidt 2010). Used with kind permission of Mark Schmidt.

where $||\mathbf{w}_{st}||_p$ is the p-norm; common choices are $p = 2$ or $p = \infty$, as explained in Section 13.5.1. This method of CRF structure learning was first suggested in (Schmidt et al. 2008). (The use of ℓ_1 regularization for learning the structure of binary MRFs was proposed in (Lee et al. 2006).)

Although this objective is convex, it can be costly to evaluate, since we need to perform inference to compute its gradient, as explained in Section 19.6.3 (this is true also for MRFs). We should therefore use an optimizer that does not make too many calls to the objective function or its gradient, such as the projected quasi-Newton method in (Schmidt et al. 2009). In addition, we can use approximate inference, such as convex belief propagation (Section 22.4.2), to compute an approximate objective and gradient more quickly. Another approach is to apply the group lasso penalty to the pseudo-likelihood discussed in Section 19.5.4. This is much faster, since inference is no longer required (Hoefling and Tibshirani 2009). Figure 26.19 shows the result of applying this procedure to the 20-newsgroup data, where y_{it} indicates the presence of word t in document i, and $\mathbf{x}_i = 1$ (so the model is an MRF).



P (C=F)	P(C=T)
0.5	0.5

C	P(S=F)	P(S=T)
F	0.5	0.5
T	0.9	0.1

C	P(R=F)	P(R=T)
F	0.8	0.2
T	0.2	0.8

S R	P(W=F)	P(W=T)
F F	1.0	0.0
T F	0.1	0.9
F T	0.1	0.9
T T	0.01	0.99

Figure 26.20 Water sprinkler DGM with corresponding binary CPTs. T and F stand for true and false.

26.8.2 Thin junction trees

So far, we have been concerned with learning "sparse" graphs, but these do not necessarily have low treewidth. For example, a $D \times D$ grid is sparse, but has treewidth $O(D)$. This means that the models we learn may be intractable to use for inference purposes, which defeats one of the two main reasons to learn graph structure in the first place (the other reason being "knowledge discovery"). There have been various attempts to learn graphical models with bounded treewidth (e.g., (Bach and Jordan 2001; Srebro 2001; Elidan and Gould 2008; Shahaf et al. 2009)), also known as **thin junction trees**, but the exact problem in general is hard.

An alternative approach is to learn a model with low **circuit complexity** (Gogate et al. 2010; Poon and Domingos 2011). Such models may have high treewidth, but they exploit context-specific independence and determinism to enable fast exact inference (see e.g., (Darwiche 2009)).

Exercises

Exercise 26.1 Causal reasoning in the sprinkler network

Consider the causal network in Figure 26.20. Let T represent true and F represent false.

a. Suppose I perform a perfect intervention and make the grass wet. What is the probability the sprinkler is on, $p(S = T|\text{do}(W = T))$?
b. Suppose I perform a perfect intervention and make the grass dry. What is the probability the sprinkler is on, $p(S = T|\text{do}(W = F))$?
c. Suppose I perform a perfect intervention and make the clouds "turn on" (e.g., by seeding them). What is the probability the sprinkler is on, $p(S = T|\text{do}(C = T))$?

27 *Latent variable models for discrete data*

27.1 Introduction

In this chapter, we are concerned with latent variable models for discrete data, such as bit vectors, sequences of categorical variables, count vectors, graph structures, relational data, etc. These models can be used to analyze voting records, text and document collections, low-intensity images, movie ratings, etc. However, we will mostly focus on text analysis, and this will be reflected in our terminology.

Since we will be dealing with so many different kinds of data, we need some precise notation to keep things clear. When modeling variable-length sequences of categorical variables (i.e., symbols or **tokens**), such as words in a document, we will let $y_{il} \in \{1, \ldots, V\}$ represent the identity of the l'th word in document i, where V is the number of possible words in the vocabulary. We assume $l = 1 : L_i$, where L_i is the (known) length of document i, and $i = 1 : N$, where N is the number of documents.

We will often ignore the word order, resulting in a **bag of words**. This can be reduced to a fixed length vector of counts (a histogram). We will use $n_{iv} \in \{0, 1, \ldots, L_i\}$ to denote the number of times word v occurs in document i, for $v = 1 : V$. Note that the $N \times V$ count matrix $\mathbf{N}$ is often large but sparse, since we typically have many documents, but most words do not occur in any given document.

In some cases, we might have multiple different bags of words, e.g., bags of text words and bags of visual words. These correspond to different "channels" or types of features. We will denote these by y_{irl}, for $r = 1 : R$ (the number of responses) and $l = 1 : L_{ir}$. If $L_{ir} = 1$, it means we have a single token (a bag of length 1); in this case, we just write $y_{ir} \in \{1, \ldots, V_r\}$ for brevity. If every channel is just a single token, we write the fixed-size response vector as $\mathbf{y}_{i,1:R}$; in this case, the $N \times R$ design matrix $\mathbf{Y}$ will not be sparse. For example, in social science surveys, y_{ir} could be the response of person i to the r'th multi-choice question.

Out goal is to build joint probability models of $p(\mathbf{y}_i)$ or $p(\mathbf{n}_i)$ using latent variables to capture the correlations. We will then try to interpret the latent variables, which provide a compressed representation of the data. We provide an overview of some approaches in Section 27.2, before going into more detail in later sections.

Towards the end of the chapter, we will consider modeling graphs and relations, which can also be represented as sparse discrete matrices. For example, we might want to model the graph of which papers mycite which other papers. We will denote these relations by $\mathbf{R}$, reserving the symbol $\mathbf{Y}$ for any categorical data (e.g., text) associated with the nodes.

27.2 Distributed state LVMs for discrete data

In this section, we summarize a variety of possible approaches for constructing models of the form $p(\mathbf{y}_{i,1:L_i})$, for bags of tokens; $p(\mathbf{y}_{1:R})$, for vectors of tokens; and $p(\mathbf{n}_i)$, for vectors of integer counts.

27.2.1 Mixture models

The simplest approach is to use a finite mixture model (Chapter 11). This associates a single discrete latent variable, $q_i \in \{1, \ldots, K\}$, with every document, where K is the number of clusters. We will use a discrete prior, $q_i \sim \text{Cat}(\boldsymbol{\pi})$. For variable length documents, we can define $p(y_{il}|q_i = k) = b_{kv}$, where b_{kv} is the probability that cluster k generates word v. The value of q_i is known as a **topic**, and the vector $\mathbf{b}_k$ is the k'th topic's word distribution. That is, the likelihood has the form

$$p(\mathbf{y}_{i,1:L_i}|q_i = k) = \prod_{l=1}^{L_i} \text{Cat}(y_{il}|\mathbf{b}_k) \tag{27.1}$$

The induced distribution on the visible data is given by

$$p(\mathbf{y}_{i,1:L_i}) = \sum_k \pi_k \left[\prod_{l=1}^{L_i} \text{Cat}(y_{il}|\mathbf{b}_k) \right] \tag{27.2}$$

The "generative story" which this encodes is as follows: for document i, pick a topic q_i from $\boldsymbol{\pi}$, call it k, and then for each word $l = 1 : L_i$, pick a word from $\mathbf{b}_k$. We will consider more sophisticated generative models later in this chapter.

If we have a fixed set of categorical observations, we can use a different topic matrix for each output variable:

$$p(\mathbf{y}_{i,1:R}|q_i = k) = \prod_{r=1}^{R} \text{Cat}(y_{il}|\mathbf{b}_k^{(r)}) \tag{27.3}$$

This is an unsupervised analog of naive Bayes classification.

We can also model count vectors. If the sum $L_i = \sum_v n_{iv}$ is known, we can use a multinomial:

$$p(\mathbf{n}_i|L_i, q_i = k) = \text{Mu}(\mathbf{n}_i|L_i, \mathbf{b}_k) \tag{27.4}$$

If the sum is unknown, we can use a Poisson class-conditional density to give

$$p(\mathbf{n}_i|q_i = k) = \prod_{v=1}^{V} \text{Poi}(n_{iv}|\lambda_{vk}) \tag{27.5}$$

In this case, $L_i|q_i = k \sim \text{Poi}(\sum_v \lambda_{vk})$.

27.2.2 Exponential family PCA

Unfortunately, finite mixture models are very limited in their expressive power. A more flexible model is to use a vector of real-valued continuous latent variables, similar to the factor analysis (FA) and PCA models in Chapter 12. In PCA, we use a Gaussian prior of the form $p(\mathbf{z}_i) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\mathbf{z}_i \in \mathbb{R}^K$, and a Gaussian likelihood of the form $p(\mathbf{y}_i|\mathbf{z}_i) = \mathcal{N}(\mathbf{W}\mathbf{z}_i, \sigma^2\mathbf{I})$. This method can certainly be applied to discrete or count data. Indeed, the method known as **latent semantic analysis** (**LSA**) or **latent semantic indexing** (**LSI**) (Deerwester et al. 1990; Dumais and Landauer 1997) is exactly equivalent to applying PCA to a term by document count matrix.

A better method for modeling categorical data is to use a multinoulli or multinomial distribution. We just have to change the likelihood to

$$p(\mathbf{y}_{i,1:L_i}|\mathbf{z}_i) = \prod_{l=1}^{L_i} \text{Cat}(y_{il}|\mathcal{S}(\mathbf{W}\mathbf{z}_i)) \tag{27.6}$$

where $\mathbf{W} \in \mathbb{R}^{V \times K}$ is a weight matrix and $\mathcal{S}$ is the softmax function. If we have a fixed number of categorical responses, we can use

$$p(\mathbf{y}_{1:R}|\mathbf{z}_i) = \prod_{r=1}^{R} \text{Cat}(y_{ir}|\mathcal{S}(\mathbf{W}_r\mathbf{z}_i)) \tag{27.7}$$

where $\mathbf{W}_r \in \mathbb{R}^{V \times K}$ is the weight matrix for the r'th response variable. This model is called **categorical PCA**, and is illustrated in Figure 27.1(a); see Section 12.4 for further discussion. If we have counts, we can use a multinomial model

$$p(\mathbf{n}_i|L_i, \mathbf{z}_i) = \text{Mu}(\mathbf{n}_i|L_i, \mathcal{S}(\mathbf{W}\mathbf{z}_i)) \tag{27.8}$$

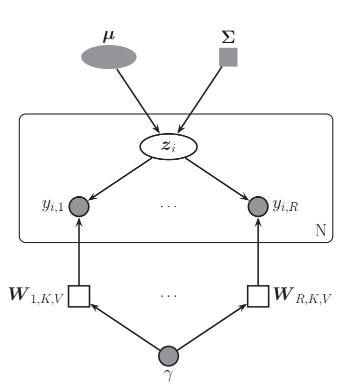
or a Poisson model

$$p(\mathbf{n}_i|\mathbf{z}_i) = \prod_{v=1}^{V} \text{Poi}(n_{iv}|\exp(\mathbf{w}_{v,:}^T\mathbf{z}_i)) \tag{27.9}$$

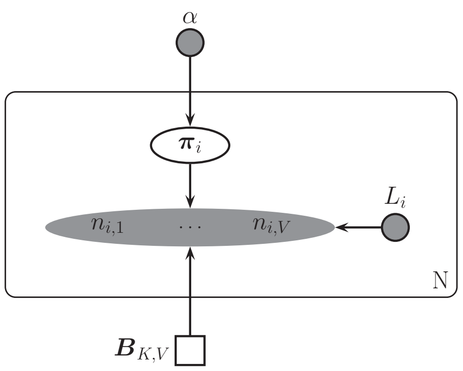
All of these models are examples of **exponential family PCA** or **ePCA** (Collins et al. 2002; Mohamed et al. 2008), which is an unsupervised analog of GLMs. The corresponding induced distribution on the visible variables has the form

$$p(\mathbf{y}_{i,1:L_i}) = \int \left[\prod_{l=1}^{L_i} p(y_{il}|\mathbf{z}_i, \mathbf{W})\right] \mathcal{N}(\mathbf{z}_i|\boldsymbol{\mu}, \boldsymbol{\Sigma}) d\mathbf{z}_i \tag{27.10}$$

Fitting this model is tricky, due to the lack of conjugacy. (Collins et al. 2002) proposed a coordinate ascent method that alternates between estimating the $\mathbf{z}_i$ and $\mathbf{W}$. This can be regarded as a degenerate version of EM, that computes a point estimate of $\mathbf{z}_i$ in the E step. The problem with the degenerate approach is that it is very prone to overfitting, since the number of latent variables is proportional to the number of datacases (Welling et al. 2008). A true EM algorithm would marginalize out the latent variables $\mathbf{z}_i$. A way to do this for categorical PCA, using variational EM, is discussed in Section 12.4. For more general models, one can use MCMC (Mohamed et al. 2008).



(a)



(b)

Figure 27.1 Two LVMs for discrete data. Circles are scalar nodes, ellipses are vector nodes, squares are matrix nodes. (a) Categorical PCA. (b) Multinomial PCA.

27.2.3 LDA and mPCA

In ePCA, the quantity $\mathbf{W}\mathbf{z}_i$ represents the natural parameters of the exponential family. Sometimes it is more convenient to use the dual parameters. For example, for the multinomial, the dual parameter is the probability vector, whereas the natural parameter is the vector of log odds.

If we want to use the dual parameters, we need to constrain the latent variables so they live in the appropriate parameter space. In the case of categorical data, we will need to ensure the latent vector lives in S_K, the K-dimensional probability simplex. To avoid confusion with ePCA, we will denote such a latent vector by $\boldsymbol{\pi}_i$. In this case, the natural prior for the latent variables is the Dirichlet, $\boldsymbol{\pi}_i \sim \text{Dir}(\boldsymbol{\alpha})$. Typically we set $\boldsymbol{\alpha} = \alpha \mathbf{1}_K$. If we set $\alpha \ll 1$, we encourage $\boldsymbol{\pi}_i$ to be sparse, as shown in Figure 2.14.

When we have a count vector whose total sum is known, the likelihood is given by

$$p(\mathbf{n}_i|L_i, \boldsymbol{\pi}_i) = \text{Mu}(\mathbf{n}_i|L_i, \mathbf{B}\boldsymbol{\pi}_i) \tag{27.11}$$

This model is called **multinomial PCA** or **mPCA** (Buntine 2002; Buntine and Jakulin 2004, 2006). See Figure 27.1(b). Since we are assuming $n_{iv} = \sum_k b_{vk}\pi_{iv}$, this can be seen as a form of matrix factorization for the count matrix. Note that we use $b_{v,k}$ to denote the parameter vector, rather than $w_{v,k}$, since we impose the constraints that $0 \leq b_{v,k} \leq 1$ and $\sum_v b_{v,k} = 1$. The corresponding marginal distribution has the form

$$p(\mathbf{n}_i|L_i) = \int \text{Mu}(\mathbf{n}_i|L_i, \mathbf{B}\boldsymbol{\pi}_i)\text{Dir}(\boldsymbol{\pi}_i|\boldsymbol{\alpha})d\boldsymbol{\pi}_i \tag{27.12}$$

Unfortunately, this integral cannot be computed analytically.

If we have a variable length sequence (of known length), we can use

$$p(\mathbf{y}_{i,1:L_i}|\boldsymbol{\pi}_i) = \prod_{l=1}^{L_i} \text{Cat}(y_{il}|\mathbf{B}\boldsymbol{\pi}_i) \tag{27.13}$$

This is called **latent Dirichlet allocation** or **LDA** (Blei et al. 2003), and will be described in much greater detail below. LDA can be thought of as a probabilistic extension of LSA, where the latent quantities π_{ik} are non-negative and sum to one. By contrast, in LSA, z_{ik} can be negative which makes interpetation difficult.

A predecessor to LDA, known as **probabilistic latent semantic indexing** or **PLSI** (Hofmann 1999), uses the same model but computes a point estimate of $\boldsymbol{\pi}_i$ for each document (similar to ePCA), rather than integrating it out. Thus in PLSI, there is no prior for $\boldsymbol{\pi}_i$.

We can modify LDA to handle a fixed number of different categorical responses as follows:

$$p(\mathbf{y}_{i,1:R}|\boldsymbol{\pi}_i) = \prod_{r=1}^{R} \mathrm{Cat}(y_{il}|\mathbf{B}^{(r)}\boldsymbol{\pi}_i) \tag{27.14}$$

This has been called the **user rating profile** (URP) model (Marlin 2003), and the **simplex factor model** (Bhattacharya and Dunson 2011).

27.2.4 GaP model and non-negative matrix factorization

Now consider modeling count vectors where we do not constrain the sum to be observed. In this case, the latent variables just need to be non-negative, so we will denote them by $\mathbf{z}_i^+$. This can be ensured by using a prior of the form

$$p(\mathbf{z}_i^+) = \prod_{k=1}^{K} \mathrm{Ga}(z_{ik}^+|\alpha_k, \beta_k) \tag{27.15}$$

The likelihood is given by

$$p(\mathbf{n}_i|\mathbf{z}_i^+) = \prod_{v=1}^{V} \mathrm{Poi}(n_{iv}|\mathbf{b}_{v,:}^T\mathbf{z}_i^+) \tag{27.16}$$

This is called the **GaP** (Gamma-Poisson) model (Canny 2004). See Figure 27.2(a).

In (Buntine and Jakulin 2006), it is shown that the GaP model, when conditioned on a fixed L_i, reduces to the mPCA model. This follows since a set of Poisson random variables, when conditioned on their sum, becomes a multinomial distribution (see e.g., (Ross 1989)).

If we set $\alpha_k = \beta_k = 0$ in the GaP model, we recover a method known as **non-negative matrix factorization** or **NMF** (Lee and Seung 2001), as shown in (Buntine and Jakulin 2006). NMF is not a probabilistic generative model, since it does not specify a proper prior for $\mathbf{z}_i^+$. Furthermore, the algorithm proposed in (Lee and Seung 2001) is another degenerate EM algorithm, so suffers from overfitting. Some procedures to fit the GaP model, which overcome these problems, are given in (Buntine and Jakulin 2006).

To encourage $\mathbf{z}_i^+$ to be sparse, we can modify the prior to be a spike-and-Gamma type prior as follows:

$$p(z_{ik}^+) = \rho_k \mathbb{I}(z_{ik}^+ = 0) + (1-\rho_k)\mathrm{Ga}(z_{ik}^+|\alpha_k, \beta_k) \tag{27.17}$$

where ρ_k is the probability of the spike at 0. This is called the conditional Gamma Poisson model (Buntine and Jakulin 2006). It is simple to modify Gibbs sampling to handle this kind of prior, although we will not go into detail here.

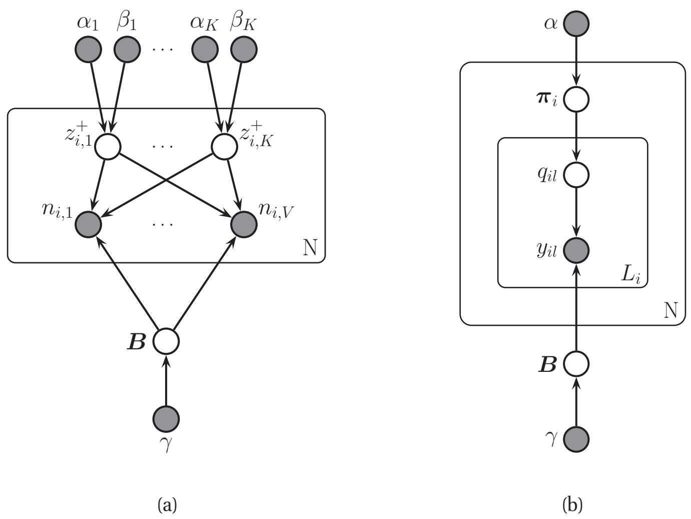


Figure 27.2 (a) Gaussian-Poisson (GAP) model. (b) Latent Dirichlet allocation (LDA) model.

27.3 Latent Dirichlet allocation (LDA)

In this section, we explain the **latent Dirichlet allocation** or **LDA** (Blei et al. 2003) model in detail.

27.3.1 Basics

In a mixture of multinoullis, every document is assigned to a single topic, $q_i \in \{1, \ldots, K\}$, drawn from a global distribution $\boldsymbol{\pi}$. In LDA, every word is assigned to its own topic, $q_{il} \in \{1, \ldots, K\}$, drawn from a document-specific distribution $\boldsymbol{\pi}_i$. Since a document belongs to a distribution over topics, rather than a single topic, the model is called an **admixture mixture** or **mixed membership model** (Erosheva et al. 2004). This model has many other applications beyond text analysis, e.g., genetics (Pritchard et al. 2000), health science (Erosheva et al. 2007), social network analysis (Airoldi et al. 2008), etc.

Adding conjugate priors to the parameters, the full model is as follows:[1]

$$\boldsymbol{\pi}_i|\alpha \sim \text{Dir}(\alpha \mathbf{1}_K) \quad (27.18)$$

$$q_{il}|\boldsymbol{\pi}_i \sim \text{Cat}(\boldsymbol{\pi}_i) \quad (27.19)$$

$$\mathbf{b}_k|\gamma \sim \text{Dir}(\gamma \mathbf{1}_V) \quad (27.20)$$

$$y_{il}|q_{il} = k, \mathbf{B} \sim \text{Cat}(\mathbf{b}_k) \quad (27.21)$$

This is illustrated in Figure 27.2(b). We can marginalize out the q_i variables, thereby creating a

1. Our notation is similar to the one we use elsewhere in this book, but is different from that used by most LDA papers. They typically use w_{nd} for the identity of word n in document d, z_{nd} to represent the discrete indicator, $\boldsymbol{\theta}_d$ as the continuous latent vector for document d, and $\boldsymbol{\beta}_k$ as the k'th topic vector.

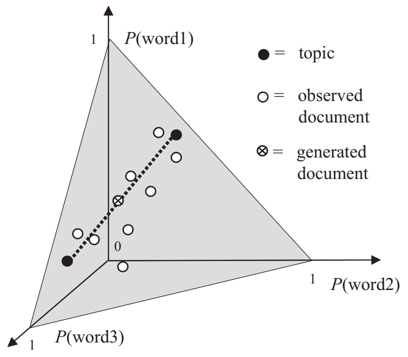


Figure 27.3 Geometric interpretation of LDA. We have $K = 2$ topics and $V = 3$ words. Each document (white dots), and each topic (black dots), is a point in the 3d simplex. Source: Figure 5 of (Steyvers and Griffiths 2007). Used with kind permission of Tom Griffiths.

direct arc from $\boldsymbol{\pi}_i$ to y_{il}, with the following CPD:

$$p(y_{il} = v|\boldsymbol{\pi}_i) = \sum_k p(y_{il} = v|q_{il} = k)p(q_{il} = k) = \sum_k \pi_{ik} b_{kv} \tag{27.22}$$

As we mentioned in the introduction, this is very similar to the multinomial PCA model proposed in (Buntine 2002), which in turn is closely related to categorical PCA, GaP, NMF, etc.

LDA has an interesting geometric interpretation. Each vector $\mathbf{b}_k$ defines a distribution over V words; each k is known as a **topic**. Each document vector $\boldsymbol{\pi}_i$ defines a distribution over K topics. So we model each document as an admixture over topics. Equivalently, we can think of LDA as a form of dimensionality reduction (assuming $K < V$, as is usually the case), where we project a point in the V-dimensional simplex (a normalized document count vector $\mathbf{x}_i$) onto the K-dimensional simplex. This is illustrated in Figure 27.3, where we have $V = 3$ words and $K = 2$ topics. The observed documents (which live in the 3d simplex) are approximated as living on a 2d simplex spanned by the 2 topic vectors, each of which lives in the 3d simplex.

One advantage of using the simplex as our latent space rather than Euclidean space is that the simplex can handle ambiguity. This is importance since in natural language, words can often have multiple meanings, a phenomomen known as **polysemy**. For example, “play” might refer to a verb (e.g., “to play ball” or “to play the coronet”), or to a noun (e.g., “Shakespeare’s play”). In LDA, we can have multiple topics, each of which can generate the word “play”, as shown in Figure 27.4, reflecting this ambiguity.

Given word l in document i, we can compute $p(q_{il} = k|\mathbf{y}_i, \boldsymbol{\theta})$, and thus infer its most likely topic. By looking at the word in isolation, it might be hard to know what sense of the word is meant, but we can disambiguate this by looking at other words in the document. In particular, given $\mathbf{x}_i$, we can infer the topic distribution $\boldsymbol{\pi}_i$ for the document; this acts as a prior for disambiguating q_{il}. This is illustrated in Figure 27.5, where we show three documents from the TASA corpus.[2] In the first document, there are a variety of music related words, which suggest

2. The TASA corpus is a collection of 37,000 high-school level English documents, comprising over 10 million words,

Topic 77

word	prob.
MUSIC	.090
DANCE	.034
SONG	.033
PLAY	.030
SING	.026
SINGING	.026
BAND	.026
PLAYED	.023
SANG	.022
SONGS	.021
DANCING	.020
PIANO	.017
PLAYING	.016
RHYTHM	.015
ALBERT	.013
MUSICAL	.013

Topic 82

word	prob.
LITERATURE	.031
POEM	.028
POETRY	.027
POET	.020
PLAYS	.019
POEMS	.019
PLAY	.015
LITERARY	.013
WRITERS	.013
DRAMA	.012
WROTE	.012
POETS	.011
WRITER	.011
SHAKESPEARE	.010
WRITTEN	.009
STAGE	.009

Topic 166

word	prob.
PLAY	.136
BALL	.129
GAME	.065
PLAYING	.042
HIT	.032
PLAYED	.031
BASEBALL	.027
GAMES	.025
BAT	.019
RUN	.019
THROW	.016
BALLS	.015
TENNIS	.011
HOME	.010
CATCH	.010
FIELD	.010

Figure 27.4 Three topics related to the word *play*. Source: Figure 9 of (Steyvers and Griffiths 2007). Used with kind permission of Tom Griffiths.

Document #29795

Bix beiderbecke, at age^{060} fifteen^{207}, sat^{174} on the slope^{071} of a bluff^{055} overlooking^{027} the mississippi^{137} river^{137}. He was listening^{077} to music^{077} coming^{009} from a passing^{043} riverboat. The music^{077} had already captured^{006} his heart^{157} as well as his ear^{119}. It was jazz^{077}. Bix beiderbecke had already had music^{077} lessons^{077}. He showed^{002} promise^{134} on the piano^{077}, and his parents^{035} hoped^{268} he might consider^{118} becoming a concert^{077} pianist^{077}. But bix was interested^{268} in another kind^{050} of music^{077}. He wanted^{268} to $\boxed{\text{play}^{077}}$ the cornet. And he wanted^{268} to $\boxed{\text{play}^{077}}$ jazz^{077}...

Document #1883

There is a simple^{050} reason^{106} why there are so few periods^{078} of really great theater^{082} in our whole western^{046} world. Too many things^{300} have to come right at the very same time. The dramatists must have the right actors^{082}, the actors^{082} must have the right playhouses, the playhouses must have the right audiences^{082}. We must remember^{288} that plays^{082} exist^{143} to be performed^{077}, not merely^{050} to be read^{254}. (even when you read^{254} a $\boxed{\text{play}^{082}}$ to yourself, try^{288} to perform^{062} it, to put^{174} it on a stage^{078}, as you go along.) as soon^{028} as a $\boxed{\text{play}^{082}}$ has to be performed^{082}, then some kind^{126} of theatrical^{082}...

Document #21359

Jim^{296} has a game^{166} book^{254}. Jim^{296} reads^{254} the book^{254}. Jim^{296} sees^{081} a game^{166} for one. Jim^{296} plays^{166} the game^{166}. Jim^{296} likes^{081} the game^{166} for one. The game^{166} book^{254} helps^{081} jim^{296}. Don^{180} comes^{040} into the house^{038}. Don^{180} and jim^{296} read^{254} the game^{166} book^{254}. The boys^{020} see a game^{166} for two. The two boys^{020} $\boxed{\text{play}^{166}}$ the game^{166}. The boys^{020} $\boxed{\text{play}^{166}}$ the game^{166} for two. The boys^{020} like the game^{166}. Meg^{282} comes^{040} into the house^{282}. Meg^{282} and don^{180} and jim^{296} read^{254} the book^{254}. They see a game^{166} for three. Meg^{282} and don^{180} and jim^{296} $\boxed{\text{play}^{166}}$ the game^{166}. They $\boxed{\text{play}^{166}}$...

Figure 27.5 Three documents from the TASA corpus containing different senses of the word *play*. Grayed out words were ignored by the model, because they correspond to uninteresting stop words (such as "and", "the", etc.) or very low frequency words. Source: Figure 10 of (Steyvers and Griffiths 2007). Used with kind permission of Tom Griffiths.

$\boldsymbol{\pi}_i$ will put most of its mass on the music topic (number 77); this in turn makes the music interpretation of "play" the most likely, as shown by the superscript. The second document interprets play in the theatrical sense, and the third in the sports sense. Note that is crucial that $\boldsymbol{\pi}_i$ be a latent variable, so information can flow between the q_{il}'s, thus enabling local disambiguation to use the full set of words.

27.3.2 Unsupervised discovery of topics

One of the main purposes of LDA is discover topics in a large collection or **corpus** of documents (see Figure 27.12 for an example). Unfortunately, since the model is unidentifiable, the interpertation of the topics can be difficult (Chang et al. 2009).. One approach, known as labeled LDA (Ramage et al. 2009), exploits the existence of tags on documents as a way to ensure identifiability. In particular, it forces the topics to correspond to the tags, and then it learns a distribution over words for each tag. This can make the results easier to interpret.

27.3.3 Quantitatively evaluating LDA as a language model

In order to evaluate LDA quantitatively, we can treat it as a **language model**, i.e., a probability distribution over sequences of words. Of course, it is not a very good language model, since it ignores word order and just looks at single words (unigrams), but it is interesting to compare LDA to other unigram-based models, such as mixtures of multinoullis, and pLSI. Such simple language models are sometimes useful for information retrieval purposes. The standard way to measure the quality of a language model is to use perplexity, which we now define below.

27.3.3.1 Perplexity

The **perplexity** of language model q given a stochastic process[3] p is defined as

$$\text{perplexity}(p, q) \triangleq 2^{H(p,q)} \tag{27.23}$$

where $H(p, q)$ is the cross-entropy of the two stochastic processes, defined as

$$H(p, q) \triangleq \lim_{N\to\infty} -\frac{1}{N} \sum_{\mathbf{y}_{1:N}} p(\mathbf{y}_{1:N}) \log q(\mathbf{y}_{1:N}) \tag{27.24}$$

The cross entropy (and hence perplexity) is minimized if $q = p$; in this case, the model can predict as well as the "true" distribution.

We can approximate the stochastic process by using a single long test sequence (composed of multiple documents and multiple sentences, complete with end-of-sentence markers), call it $\mathbf{y}^*_{1:N}$. (This approximation becomes more and more accurate as the sequence gets longer, provided the process is stationary and ergodic (Cover and Thomas 2006).) Define the empirical distribution (an approximation to the stochastic process) as

$$p_{\text{emp}}(\mathbf{y}_{1:N}) = \delta_{\mathbf{y}^*_{1:N}}(\mathbf{y}_{1:N}) \tag{27.25}$$

collated by a company formerly known as Touchstone Applied Science Associates, but now known as Questar Assessment Inc `www.questarai.com`.

3. A stochastic process is one which can define a joint distribution over an arbitrary number of random variables. We can think of natural language as a stochastic process, since it can generate an infinite stream of words.

In this case, the cross-entropy becomes

$$H(p_{\text{emp}}, q) = -\frac{1}{N} \log q(\mathbf{y}^*_{1:N}) \tag{27.26}$$

and the perplexity becomes

$$\text{perplexity}(p_{\text{emp}}, q) = 2^{H(p_{\text{emp}},q)} = q(\mathbf{y}^*_{1:N})^{-1/N} = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{q(y^*_i|\mathbf{y}^*_{1:i-1})}} \tag{27.27}$$

We see that this is the geometric mean of the inverse predictive probabilities, which is the usual definition of perplexity (Jurafsky and Martin 2008, p96).

In the case of unigram models, the cross entropy term is given by

$$H = -\frac{1}{N} \sum_{i=1}^{N} \frac{1}{L_i} \sum_{l=1}^{L_i} \log q(y_{il}) \tag{27.28}$$

where N is the number of documents and L_i is the number of words in document i. Hence the perplexity of model q is given by

$$\text{perplexity}(p_{\text{emp}}, p) = \exp\left(-\frac{1}{N} \sum_{i=1}^{N} \frac{1}{L_i} \sum_{l=1}^{L_i} \log q(y_{il})\right) \tag{27.29}$$

Intuitively, perplexity mesures the weighted average **branching factor** of the model's predictive distribution. Suppose the model predicts that each symbol (letter, word, whatever) is equally likely, so $p(y_i|\mathbf{y}_{1:i-1}) = 1/K$. Then the perplexity is $((1/K)^N)^{-1/N} = K$. If some symbols are more likely than others, and the model correctly reflects this, its perplexity will be lower than K. Of course, $H(p, p) = H(p) \leq H(p, q)$, so we can never reduce the perplexity below the entropy of the underlying stochastic process.

27.3.3.2 Perplexity of LDA

The key quantity is $p(v)$, the predictive distribution of the model over possible words. (It is implicitly conditioned on the training set.) For LDA, this can be approximated by plugging in $\mathbf{B}$ (e.g., the posterior mean estimate) and approximately integrating out $\mathbf{q}$ using mean field inference (see (Wallach et al. 2009) for a more accurate way to approximate the predictive likelihood).

In Figure 27.6, we compare LDA to several other simple unigram models, namely MAP estimation of a multinoulli, MAP estimation of a mixture of multinoullis, and pLSI. (When performing MAP estimation, the same Dirichlet prior on $\mathbf{B}$ was used as in the LDA model.) The metric is perplexity, as in Equation 27.29, and the data is a subset of the TREC AP corpus containing 16,333 newswire articles with 23,075 unique terms. We see that LDA significantly outperforms these other methods.

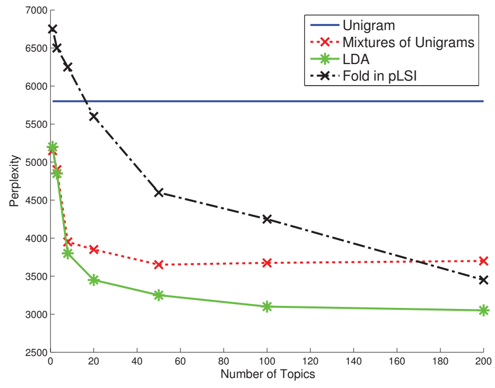


Figure 27.6 Perplexity vs number of topics on the TREC AP corpus for various language models. Based on Figure 9 of (Blei et al. 2003). Figure generated by `bleiLDAperplexityPlot`.

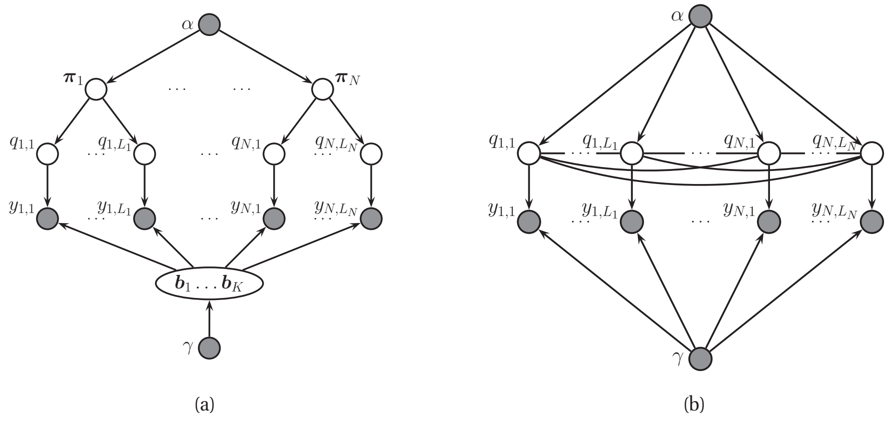


Figure 27.7 (a) LDA unrolled for N documents. (b) Collapsed LDA, where we integrate out the $\boldsymbol{\pi}_i$ and the $\mathbf{b}_k$.

27.3.4 Fitting using (collapsed) Gibbs sampling

It is straightforward to derive a Gibbs sampling algorithm for LDA. The full conditionals are as follows:

$$p(q_{il} = k|\cdot) \propto \exp[\log \pi_{ik} + \log b_{k,x_{il}}] \tag{27.30}$$

$$p(\boldsymbol{\pi}_i|\cdot) = \text{Dir}(\{\alpha_k + \sum_l \mathbb{I}(z_{il} = k)\}) \tag{27.31}$$

$$p(\mathbf{b}_k|\cdot) = \text{Dir}(\{\gamma_v + \sum_i \sum_l \mathbb{I}(x_{il} = v, z_{il} = k)\}) \tag{27.32}$$

However, one can get better performance by analytically integrating out the $\boldsymbol{\pi}_i$'s and the $\mathbf{b}_k$'s, both of which have a Dirichlet distribution, and just sampling the discrete q_{il}'s. This approach was first suggested in (Griffiths and Steyvers 2004), and is an example of **collapsed Gibbs sampling**. Figure 27.7(b) shows that now all the q_{il} variables are fully correlated. However, we can sample them one at a time, as we explain below.

First, we need some notation. Let $c_{ivk} = \sum_{l=1}^{L_i} \mathbb{I}(q_{il} = k, y_{il} = v)$ be the number of times word v is assigned to topic k in document i. Let $c_{ik} = \sum_v c_{ivk}$ be the number of times any word from document i has been assigned to topic k. Let $c_{vk} = \sum_i c_{ivk}$ be the number of times word v has been assigned to topic k in any document. Let $n_{iv} = \sum_k c_{ivk}$ be the number of times word v occurs in document i; this is observed. Let $c_k = \sum_v c_{vk}$ be the number of words assigned to topic k. Finally, let $L_i = \sum_k c_{ik}$ be the number of words in document i; this is observed.

We can now derive the marginal prior. By applying Equation 5.24, one can show that

$$p(\mathbf{q}|\alpha) = \prod_i \int \left[\prod_{l=1}^{L_i} \text{Cat}(q_{il}|\boldsymbol{\pi}_i) \right] \text{Dir}(\boldsymbol{\pi}_i|\boldsymbol{\alpha}\mathbf{1}_K) d\boldsymbol{\pi}_i \tag{27.33}$$

$$= \left(\frac{\Gamma(K\alpha)}{\Gamma(\alpha)^K} \right)^N \prod_{i=1}^N \frac{\prod_{k=1}^K \Gamma(c_{ik} + \alpha)}{\Gamma(L_i + K\alpha)} \tag{27.34}$$

By similar reasoning, one can show

$$p(\mathbf{y}|\mathbf{q},\gamma) = \prod_k \int \left[\prod_{il:q_{il}=k} \text{Cat}(y_{il}|\mathbf{b}_k) \right] \text{Dir}(\mathbf{b}_k|\gamma\mathbf{1}_V) d\mathbf{b}_k \tag{27.35}$$

$$= \left(\frac{\Gamma(V\beta)}{\Gamma(\beta)^V} \right)^K \prod_{k=1}^K \frac{\prod_{v=1}^V \Gamma(c_{vk} + \beta)}{\Gamma(c_k + V\beta)} \tag{27.36}$$

From the above equations, and using the fact that $\Gamma(x+1)/\Gamma(x) = x$, we can derive the full conditional for $p(q_{il}|\mathbf{q}_{-i,l})$. Define c_{ivk}^- to be the same as c_{ivk} except it is compute by summing over all locations in document i except for q_{il}. Also, let $y_{il} = v$. Then

$$p(q_{i,l} = k|\mathbf{q}_{-i,l}, \mathbf{y}, \alpha, \gamma) \propto \frac{c_{v,k}^- + \gamma}{c_k^- + V\gamma} \frac{c_{i,k}^- + \alpha}{L_i + K\alpha} \tag{27.37}$$

We see that a word in a document is assigned to a topic based both on how often that word is generated by the topic (first term), and also on how often that topic is used in that document (second term).

Given Equation 27.37, we can implement the collapsed Gibbs sampler as follows. We randomly assign a topic to each word, $q_{il} \in \{1, \ldots, K\}$. We can then sample a new topic as follows: for a given word in the corpus, decrement the relevant counts, based on the topic assigned to the current word; draw a new topic from Equation 27.37, update the count matrices; and repeat. This algorithm can be made efficient since the count matrices are very sparse.

27.3.5 Example

This process is illustrated in Figure 27.8 on a small example with two topics, and five words. The left part of the figure illustrates 16 documents that were sampled from the LDA model using

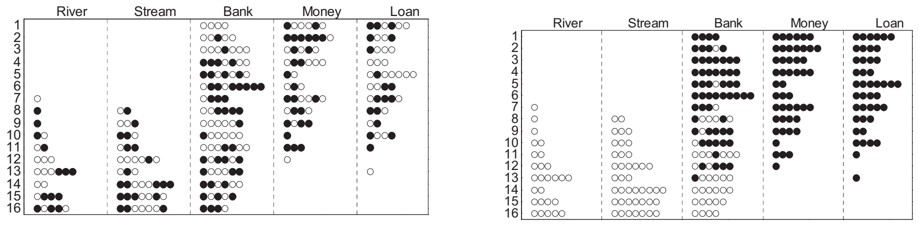


Figure 27.8 Illustration of (collapsed) Gibbs sampling applied to a small LDA example. There are $N = 16$ documents, each containing a variable number of words drawn from a vocabulary of $V = 5$ words, There are two topics. A white dot means word the word is assigned to topic 1, a black dot means the word is assigned to topic 2. (a) The initial random assignment of states. (b) A sample from the posterior after 64 steps of Gibbs sampling. Source: Figure 7 of (Steyvers and Griffiths 2007). Used with kind permission of Tom Griffiths.

$p(\text{money}|k = 1) = p(\text{loan}|k = 1) = p(\text{bank}|k = 1) = 1/3$ and $p(\text{river}|k = 2) = p(\text{stream}|k = 2) = p(\text{bank}|k = 2) = 1/3$. For example, we see that the first document contains the word "bank" 4 times (indicated by the four dots in row 1 of the "bank" column), as well as various other financial terms. The right part of the figure shows the state of the Gibbs sampler after 64 iterations. The "correct" topic has been assigned to each token in most cases. For example, in document 1, we see that the word "bank" has been correctly assigned to the financial topic, based on the presence of the words "money" and "loan". The posterior mean estimate of the parameters is given by $\hat{p}(\text{money}|k = 1) = 0.32$, $\hat{p}(\text{loan}|k = 1) = 0.29$, $\hat{p}(\text{bank}|k = 1) = 0.39$, $\hat{p}(\text{river}|k = 2) = 0.25$, $\hat{p}(\text{stream}|k = 2) = 0.4$, and $\hat{p}(\text{bank}|k = 2) = 0.35$, which is impressively accurate, given that there are only 16 training examples.

27.3.6 Fitting using batch variational inference

A faster alternative to MCMC is to use variational EM. (We cannot use exact EM since exact inference of $\boldsymbol{\pi}_i$ and $\mathbf{q}_i$ is intractable.) We give the details below.

27.3.6.1 Sequence version

Following (Blei et al. 2003), we will use a fully factorized (mean field) approximation of the form

$$q(\boldsymbol{\pi}_i, \mathbf{q}_i) = \text{Dir}(\boldsymbol{\pi}_i|\tilde{\boldsymbol{\pi}}_i) \prod_l \text{Cat}(q_{il}|\tilde{\mathbf{q}}_{il}) \tag{27.38}$$

We will follow the usual mean field recipe. For $q(q_{il})$, we use Bayes' rule, but where we need to take expectations over the prior:

$$\tilde{q}_{ilk} \quad \propto \quad b_{y_{i,l},k} \exp(\mathbb{E}[\log \pi_{ik}]) \tag{27.39}$$

where

$$\mathbb{E}[\log \pi_{ik}] = \psi_k(\tilde{\boldsymbol{\pi}}_{i.}) \triangleq \Psi(\tilde{\pi}_{ik}) - \Psi(\sum_{k'} \tilde{\pi}_{ik'}) \tag{27.40}$$

where Ψ is the digamma function. The update for $q(\boldsymbol{\pi}_i)$ is obtained by adding up the expected counts:

$$\tilde{\pi}_{ik} = \alpha_k + \sum_l \tilde{q}_{ilk} \tag{27.41}$$

The M step is obtained by adding up the expected counts and normalizing:

$$\hat{b}_{vk} \propto \gamma_v + \sum_{i=1}^{N} \sum_{l=1}^{L_i} \tilde{q}_{ilk} \mathbb{I}(y_{il} = v) \tag{27.42}$$

27.3.6.2 Count version

Note that the E step takes $O((\sum_i L_i)VK)$ space to store the $\tilde{q}_{ilk}$. It is much more space efficient to perform inference in the mPCA version of the model, which works with counts; these only take $O(NVK)$ space, which is a big savings if documents are long. (By contrast, the collapsed Gibbs sampler must work explicitly with the q_{il} variables.)

We will focus on approximating $p(\boldsymbol{\pi}_i, \mathbf{c}_i | \mathbf{n}_i, L_i)$, where we write $\mathbf{c}_i$ as shorthand for $\mathbf{c}_{i..}$. We will again use a fully factorized (mean field) approximation of the form

$$q(\boldsymbol{\pi}_i, \mathbf{c}_i) = \text{Dir}(\boldsymbol{\pi}_i | \tilde{\boldsymbol{\pi}}_i) \prod_v \text{Mu}(\mathbf{c}_{iv.} | n_{iv}, \tilde{\mathbf{c}}_{iv.}) \tag{27.43}$$

The new E step becomes

$$\tilde{\pi}_{ik} = \alpha_k + \sum_v n_{iv} \tilde{c}_{ivk} \tag{27.44}$$

$$\tilde{c}_{ivk} \propto b_{vk} \exp(\mathbb{E}[\log \pi_{ik}]) \tag{27.45}$$

The new M step becomes

$$\hat{b}_{vk} \propto \gamma_v + \sum_i n_{iv} \tilde{c}_{ivk} \tag{27.46}$$

27.3.6.3 VB version

We now modify the algorithm to use VB instead of EM, so that we infer the parameters as well as the latent variables. There are two advantages to this. First, by setting $\gamma \ll 1$, VB will encourage $\mathbf{B}$ to be sparse (as in Section 21.6.1.6). Second, we will be able to generalize this to the online learning setting, as we discuss below.

Our new posterior approximation becomes

$$q(\boldsymbol{\pi}_i, \mathbf{c}_i, \mathbf{B}) = \text{Dir}(\boldsymbol{\pi}_i | \tilde{\boldsymbol{\pi}}_i) \prod_v \text{Mu}(\mathbf{c}_{iv.} | n_{iv}, \tilde{\mathbf{c}}_{iv.}) \prod_k \text{Dir}(\mathbf{b}_{.k} | \tilde{\mathbf{b}}_{.k}) \tag{27.47}$$

The update for $\tilde{c}_{ivk}$ changes, to the following:

$$\tilde{c}_{ivk} \propto \exp\left(\mathbb{E}[\log b_{vk}] + \mathbb{E}[\log \pi_{ik}]\right) \tag{27.48}$$

Algorithm 27.1: Batch VB for LDA

```
1  Input: $n_{iv}$, $K$, $\alpha_k$, $\gamma_v$;
2  Estimate $\tilde{b}_{vk}$ using EM for multinomial mixtures;
3  Initialize counts $n_{iv}$;
4  while not converged do
5      // E step ;
6      $s_{vk} = 0$ // expected sufficient statistics;
7      for each document $i = 1 : N$ do
8          $(\tilde{\boldsymbol{\pi}}_i, \tilde{\mathbf{c}}_i) = \text{Estep}(\mathbf{n}_i, \tilde{\mathbf{B}}, \boldsymbol{\alpha})$;
9          $s_{vk} += n_{iv}\tilde{c}_{ivk}$;
10     // M step ;
11     for each topic $k = 1 : K$ do
12         $\tilde{b}_{vk} = \gamma_v + s_{vk}$;
13 function $(\tilde{\boldsymbol{\pi}}_i, \tilde{\mathbf{c}}_i) = \text{Estep}(\mathbf{n}_i, \tilde{\mathbf{B}}, \boldsymbol{\alpha})$;
14 Initialize $\tilde{\pi}_{ik} = \alpha_k$;
15 repeat
16     $\tilde{\pi}_{i\cdot}^{old} = \tilde{\pi}_{i\cdot}$, $\tilde{\pi}_{ik} = \alpha_k$;
17     for each word $v = 1 : V$ do
18         for each topic $k = 1 : K$ do
19             $\tilde{c}_{ivk} = \exp\left(\psi_k(\tilde{\mathbf{b}}_{v\cdot}) + \psi_k(\tilde{\boldsymbol{\pi}}_{i\cdot}^{old})\right)$;
20         $\tilde{\mathbf{c}}_{iv\cdot} = \text{normalize}(\tilde{\mathbf{c}}_{iv\cdot})$;
21         $\tilde{\pi}_{ik} += n_{iv}\tilde{c}_{ivk}$
22 until $\frac{1}{K}\sum_k |\tilde{\pi}_{ik} - \tilde{\pi}_{ik}^{old}| <$ thresh;
```

Also, the M step becomes

$$\tilde{b}_{vk} = \gamma_v + \sum_i \tilde{c}_{ivk} \tag{27.49}$$

No normalization is required, since we are just updating the pseudcounts. The overall algorithm is summarized in Algorithm 22.

27.3.7 Fitting using online variational inference

In the bathc version, the E step clearly takes $O(NKVT)$ time, where T is the number of mean field updates (typically $T \sim 5$). This can be slow if we have many documents. This can be reduced by using stochastic gradient descent (Section 8.5.2) to perform online variational inference, as we now explain.

We can derive an online version, following (Hoffman et al. 2010). We perform an E step in the usual way. We then compute the variational parameters for $\mathbf{B}$ treating the expected sufficient statistics from the single data case as if the whole data set had those statistics. Finally, we make

Algorithm 27.2: Online variational Bayes for LDA

```
1 Input: $n_{iv}$, $K$, $\alpha_k$, $\gamma_v$, $\tau_0$, $\kappa$;
2 Initialize $\tilde{b}_{vk}$ randomly;
3 for $t = 1 : \infty$ do
4     Set step size $\rho_t = (\tau_0 + t)^{-\kappa}$;
5     Pick document $i = i(t)$; ;
6     $(\tilde{\boldsymbol{\pi}}_i, \tilde{\mathbf{c}}_i) = \text{Estep}(\mathbf{n}_i, \tilde{\mathbf{B}}, \boldsymbol{\alpha})$;
7     $\tilde{b}_{vk}^{new} = \gamma_v + N n_{iv} \tilde{c}_{ivk}$;
8     $\tilde{b}_{vk} = (1 - \rho_t)\tilde{b}_{vk} + \rho_t \tilde{b}_{vk}^{new}$;
```

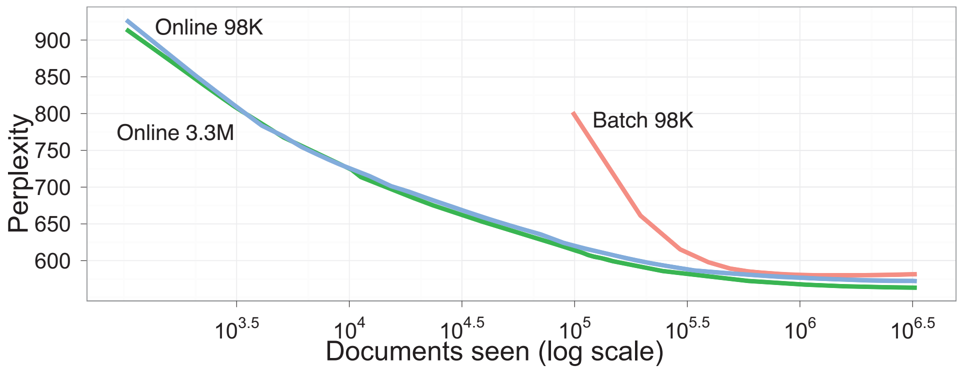


Figure 27.9 Test perplexity vs number of training documents for batch and online VB-LDA. From Figure 1 of (Hoffman et al. 2010). Used with kind permission of David Blei.

a partial update for the variational parameters for $\mathbf{B}$, putting weight ρ_t on the new estimate and weight $1 - \rho_t$ on the old estimate. The step size ρ_t decays over time, as in Equation 8.83. The overall algorithm is summarized in Algorithm 3. In practice, we should use mini-batches, as explained in Section 8.5.2.3. In (Hoffman et al. 2010), they used a batch of size 256–4096.

Figure 27.9 plots the perplexity on a test set of size 1000 vs number of analyzed documents (E steps), where the data is drawn from (English) Wikipedia. The figure shows that online variational inference is much faster than offline inference, yet produces similar results.

27.3.8 Determining the number of topics

Choosing K, the number of topics, is a standard model selection problem. Here are some approaches that have been taken:

- Use annealed importance sampling (Section 24.6.2) to approximate the evidence (Wallach et al. 2009).
- Cross validation, using the log likelihood on a test set.

- Use the variational lower bound as a proxy for $\log p(\mathcal{D}|K)$.
- Use non-parametric Bayesian methods (Teh et al. 2006).

27.4 Extensions of LDA

Many extensions of LDA have been proposed since the first paper came out in 2003. We briefly discuss a few of these below.

27.4.1 Correlated topic model

One weakness of LDA is that it cannot capture correlation between topics. For example, if a document has the "business" topic, it is reasonable to expect the "finance" topic to co-occcur. The source of the problem is the use of a Dirichlet prior for $\boldsymbol{\pi}_i$. The problem with the Dirichelt it that it is characterized by just a mean vector and a strength parameter, but its covariance is fixed ($\Sigma_{ij} = -\alpha_i \alpha_j$), rather than being a free parameter.

One way around this is to replace the Dirichlet prior with the logistic normal distribution, as in categorical PCA (Section 27.2.2). The model becomes

$$\begin{aligned}
\mathbf{b}_k|\gamma &\sim \text{Dir}(\gamma \mathbf{1}_V) && (27.50)\\
\mathbf{z}_i &\sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) && (27.51)\\
\boldsymbol{\pi}_i|\mathbf{z}_i &= \mathcal{S}(\mathbf{z}_i) && (27.52)\\
q_{il}|\boldsymbol{\pi}_i &\sim \text{Cat}(\boldsymbol{\pi}_i) && (27.53)\\
y_{il}|q_{il} = k, \mathbf{B} &\sim \text{Cat}(\mathbf{b}_k) && (27.54)
\end{aligned}$$

This is known as the **correlated topic model** (Blei and Lafferty 2007). This is very similar to categorical PCA, but slightly different. To see the difference, let us marginalize out the q_{il} and $\boldsymbol{\pi}_i$. Then in the CTM we have

$$y_{il} \sim \text{Cat}(\mathbf{B}\mathcal{S}(\mathbf{z}_i)) \tag{27.55}$$

where $\mathbf{B}$ is a stochastic matrix. By contrast, in catPCA we have

$$y_{il} \sim \text{Cat}(\mathcal{S}(\mathbf{W}\mathbf{z}_i)) \tag{27.56}$$

where $\mathbf{W}$ is an unconstrained matrix.

Fitting this model is tricky, since the prior for $\boldsymbol{\pi}_i$ is no longer conjugate to the multinomial likelihood for q_{il}. However, we can use any of the variational methods in Section 21.8.1.1, where we discussed Bayesian multiclass logistic regression. In the CTM case, things are even harder since the categorical response variables $\mathbf{q}_i$ are hidden, but we can handle this by using an additional mean field approximation. See (Blei and Lafferty 2007) for details.

Having fit the model, one can then convert $\hat{\boldsymbol{\Sigma}}$ to a sparse precision matrix $\hat{\boldsymbol{\Sigma}}^{-1}$ by pruning low-strength edges, to get a sparse Gaussian graphical model. This allows you to visualize the correlation between topics. Figure 27.10 shows the result of applying this procedure to articles from *Science* magazine, from 1990-1999. (This corpus contains 16,351 documents, and 5.7M words (19,088 of them unique), after stop-word and low-frequency removal.) Nodes represent topics, with the top 5 words per topic listed inside. The font size reflects the overall prevalence of the topic in the corpus. Edges represent significant elements of the precision matrix.

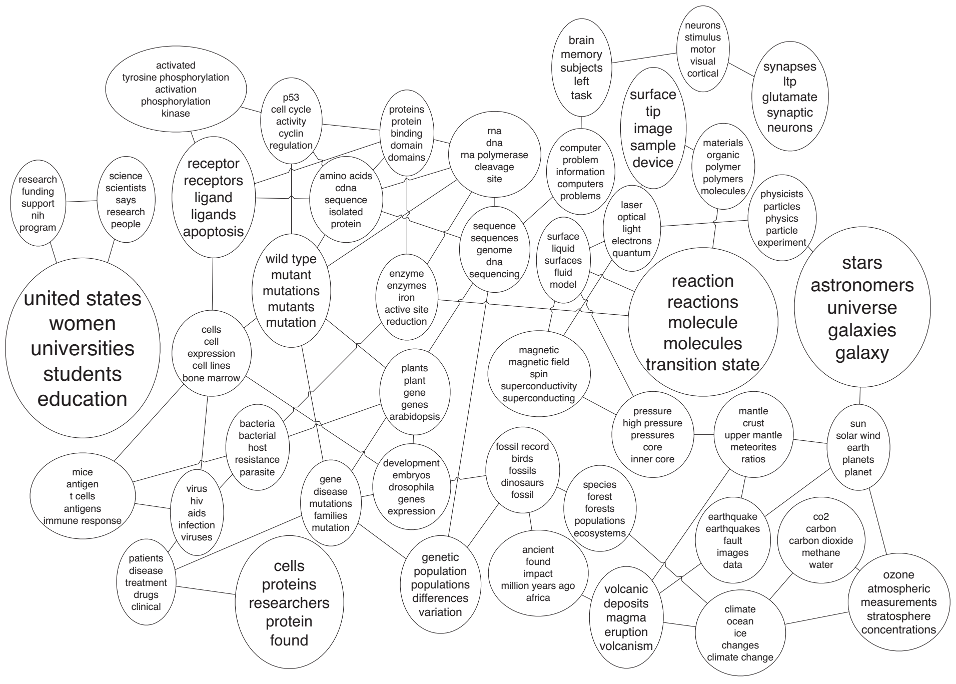

activated tyrosine phosphorylation activation phosphorylation kinase
p53 cell cycle activity cyclin regulation
proteins protein binding domain domains
rna dna rna polymerase cleavage site
brain memory subjects left task
neurons stimulus motor visual cortical
synapses ltp glutamate synaptic neurons
surface tip image sample device
computer problem information computers problems
materials organic polymer polymers molecules
research funding support nih program
science scientists says research people
receptor receptors ligand ligands apoptosis
amino acids cdna sequence isolated protein
sequence sequences genome dna sequencing
surface liquid surfaces fluid model
laser optical light electrons quantum
physicists particles physics particle experiment
united states women universities students education
wild type mutant mutations mutants mutation
enzyme enzymes iron active site reduction
reaction reactions molecule molecules transition state
stars astronomers universe galaxies galaxy
cells cell expression cell lines bone marrow
plants plant gene genes arabidopsis
magnetic magnetic field spin superconductivity superconducting
bacteria bacterial host resistance parasite
pressure high pressure pressures core inner core
mantle crust upper mantle meteorites ratios
sun solar wind earth planets planet
mice antigen t cells antigens immune response
virus hiv aids infection viruses
gene disease mutations families mutation
development embryos drosophila genes expression
fossil record birds fossils dinosaurs fossil
species forest forests populations ecosystems
earthquake earthquakes fault images data
co2 carbon carbon dioxide methane water
patients disease treatment drugs clinical
cells proteins researchers protein found
genetic population populations differences variation
ancient found impact million years ago africa
volcanic deposits magma eruption volcanism
climate ocean ice changes climate change
ozone atmospheric measurements stratosphere concentrations

Figure 27.10 Output of the correlated topic model (with $K = 50$ topics) when applied to articles from *Science*. Nodes represent topics, with the 5 most probable phrases from each topic shown inside. Font size reflects overall prevalence of the topic. See `http://www.cs.cmu.edu/~lemur/science/` for an interactive version of this model with 100 topics. Source: Figure 2 of (Blei and Lafferty 2007). Used with kind permission of David Blei.

27.4.2 Dynamic topic model

In LDA, the topics (distributions over words) are assumed to be static. In some cases, it makes sense to allow these distributions to evolve smoothly over time. For example, an article might use the topic "neuroscience", but if it was written in the 1900s, it is more likely to use words like "nerve", whereas if it was written in the 2000s, it is more likely to use words like "calcium receptor" (this reflects the general trend of neuroscience towards molecular biology).

One way to model this is use a dynamic logistic normal model, as illustrated in Figure 27.11. In particular, we assume the topic distributions evolve according to a Gaussian random walk, and then we map these Gaussian vectors to probabilities via the softmax function:

$$\mathbf{b}_{t,k}|\mathbf{b}_{t-1,k} \sim \mathcal{N}(\mathbf{b}_{t-1,k}, \sigma^2 \mathbf{1}_V) \tag{27.57}$$

$$\boldsymbol{\pi}_i^t \sim \text{Dir}(\alpha \mathbf{1}_K) \tag{27.58}$$

$$q_{il}^t|\boldsymbol{\pi}_i^t \sim \text{Cat}(\boldsymbol{\pi}_i^t) \tag{27.59}$$

$$y_{il}^t|q_{il}^t = k, \mathbf{B}^t \sim \text{Cat}(\mathcal{S}(\mathbf{b}_k^t)) \tag{27.60}$$

This is known as a **dynamic topic model** (Blei and Lafferty 2006b).

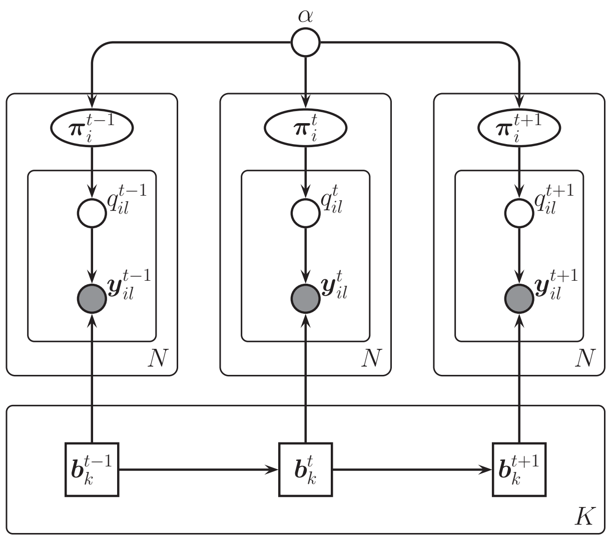


Figure 27.11 The dynamic topic model.

One can perform approximate infernece in this model using a structured mean field method (Section 21.4), that exploits the Kalman smoothing algorithm (Section 18.3.1) to perform exact inference on the linear-Gaussian chain between the $\mathbf{b}_{t,k}$ nodes (see (Blei and Lafferty 2006b) for details).

Figure 27.12 illustrates a typical output of the system when applied to 100 years of articles from *Science*. On the top, we visualize the top 10 words from a specific topic (which seems to be related to neuroscience) after 10 year intervals. On the bottom left, we plot the probability of some specific words belonging to this topic. On the bottom right, we list the titles of some articles that contained this topic.

One interesting application of this model is to perform temporally-corrected document retrieval. That is, suppose we look for documents about the inheritance of disease. Modern articles will use words like “DNA”, but older articles (before the discovery of DNA) may use other terms such as “heritable unit”. But both articles are likely to use the same topics. Similar ideas can be used to perform cross-language information retrieval, see e.g., (Cimiano et al. 2009).

27.4.3 LDA-HMM

The LDA model assumes words are exchangeable, which is clearly not true. A simple way to model sequential dependence between words is to use a hidden Markov model or HMM. The trouble with HMMs is that they can only model short-range dependencies, so they cannot capture the overall gist of a document. Hence they can generate syntactically correct sentences (see e.g., Table 17.1). but not semantically plausible ones.

It is possible to combine LDA with HMM to create a model called **LDA-HMM** (Griffiths et al.

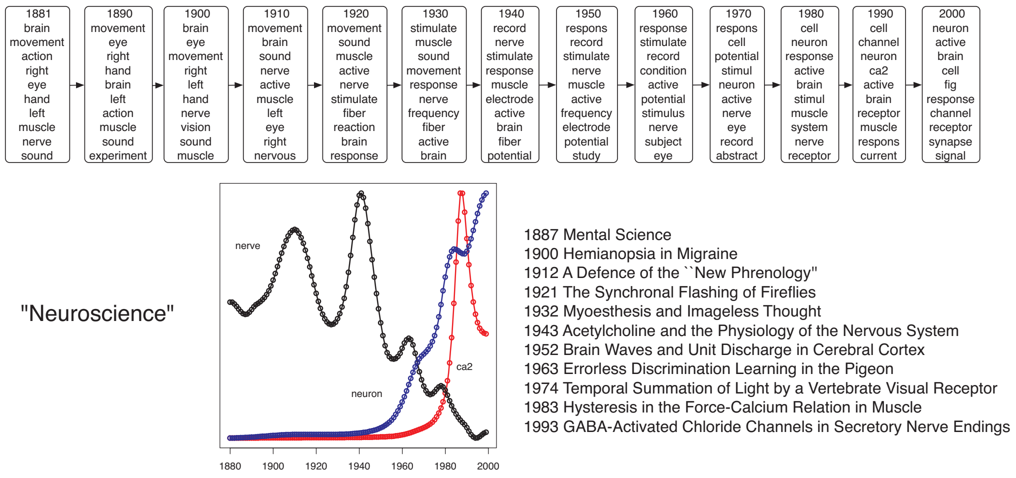


Figure 27.12 Part of the output of the dynamic topic model when applied to articles from *Science*. We show the top 10 words for the neuroscience topic over time. We also show the probability of three words within this topic over time, and some articles that contained this topic. Source: Figure 4 of (Blei and Lafferty 2006b). Used with kind permission of David Blei.

2004). This model uses the HMM states to model function or syntactic words, such as "and" or "however", and uses the LDA to model content or semantic words, which are harder to predict. There is a distinguished HMM state which specifies when the LDA model should be used to generate the word; the rest of the time, the HMM generates the word.

More formally, for each document i, the model defines an HMM with states $z_{il} \in \{0, \ldots, C\}$. In addition, each document has an LDA model associated with it. If $z_{il} = 0$, we generate word y_{il} from the semantic LDA model, with topic specified by q_{il}; otherwise we generate word y_{il} from the syntactic HMM model. The DGM is shown in Figure 27.13. The CPDs are as follows:

$$p(\boldsymbol{\pi}_i) = \text{Dir}(\boldsymbol{\pi}_i|\alpha \mathbf{1}_K) \tag{27.61}$$

$$p(q_{il} = k|\boldsymbol{\pi}_i) = \pi_{ik} \tag{27.62}$$

$$p(z_{il} = c'|z_{i,l-1} = c) = A^{HMM}(c, c') \tag{27.63}$$

$$p(y_{il} = v|q_{il} = k, z_{il} = c) = \begin{cases} B^{LDA}(k, v) & \text{if } c = 0 \\ B^{HMM}(c, v) & \text{if } c > 0 \end{cases} \tag{27.64}$$

where $\mathbf{B}^{LDA}$ is the usual topic-word matrix, $\mathbf{B}^{HMM}$ is the state-word HMM emission matrix and $\mathbf{A}^{HMM}$ is the state-state HMM transition matrix.

Inference in this model can be done with collapsed Gibbs sampling, analytically integrating out all the continuous quantities. See (Griffiths et al. 2004) for the details.

The results of applying this model (with $K = 200$ LDA topics and $C = 20$ HMM states) to the combined Brown and TASA corpora[4] are shown in Table 27.1. We see that the HMM generally is

4. The Brown corpus consists of 500 documents and 1,137,466 word tokens, with part-of-speech tags for each token.

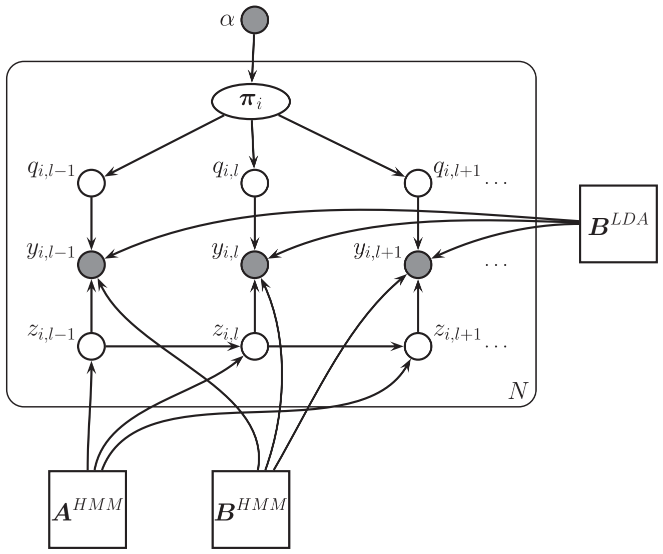


Figure 27.13 LDA-HMM model.

1. In contrast to this approach, we study here how the overall network activity can $\boxed{\text{control}}$ single cell parameters such as input resistance, as well as time and space constants, parameters that are crucial for excitability and spariotemporal (sic) integration.

   The integrated architecture in this paper combines feed forward $\boxed{\text{control}}$ and error feedback adaptive $\boxed{\text{control}}$ using neural networks.

2. In other words, for our proof of convergence, we require the softassign algorithm to $\boxed{\text{return}}$ a doubly stochastic matrix as *sinkhorn theorem guarantees that it will instead of a matrix which is merely close to being doubly stochastic based on some reasonable metric.

   The aim is to construct a portfolio with a maximal expected $\boxed{\text{return}}$ for a given risk level and time horizon while simultaneously obeying *institutional or *legally required constraints.

3. The left $\boxed{\text{graph}}$ is the standard experiment the right from a training with # samples.

   The $\boxed{\text{graph}}$ G is called the *guest $\boxed{\text{graph,}}$ and H is called the host $\boxed{\text{graph.}}$

Figure 27.14 Function and content words in the NIPS corpus, as distinguished by the LDA-HMM model. Graylevel indicates posterior probability of assignment to LDA component, with black being highest. The boxed word appears as a function word in one sentence, and as a content word in another sentence. Asterisked words had low frequency, and were treated as a single word type by the model. Source: Figure 4 of (Griffiths et al. 2004). Used with kind permission of Tom Griffiths.

the	the	the	the	the	a	the	the	the
blood	,	,	of	a	the	,	,	,
,	and	and	,	of	of	of	a	a
of	of	of	to	,	,	a	of	in
body	a	in	in	in	in	and	and	game
heart	in	land	and	to	water	in	drink	ball
and	trees	to	classes	picture	is	story	alcohol	and
in	tree	farmers	government	film	and	is	to	team
to	with	for	a	image	matter	to	bottle	to
is	on	farm	state	lens	are	as	in	play
blood	forest	farmers	government	light	water	story	drugs	ball
heart	trees	land	state	eye	matter	stories	drug	game
pressure	forests	crops	federal	lens	molecules	poem	alcohol	team
body	land	farm	public	image	liquid	characters	people	*
lungs	soil	food	local	mirror	particles	poetry	drinking	baseball
oxygen	areas	people	act	eyes	gas	character	person	players
vessels	park	farming	states	glass	solid	author	effects	football
arteries	wildlife	wheat	national	object	substance	poems	marijuana	player
*	area	farms	laws	objects	temperature	life	body	field
breathing	rain	corn	department	lenses	changes	poet	use	basketball
the	in	he	*	be	said	can	time	,
a	for	it	new	have	made	would	way	;
his	to	you	other	see	used	will	years	(
this	on	they	first	make	came	could	day	:
their	with	i	same	do	went	may	part	)
these	at	she	great	know	found	had	number	
your	by	we	good	get	called	must	kind	
her	from	there	small	go		do	place	
my	as	this	little	take		have		
some	into	who	old	find		did		

Table 27.1 Upper row: Topics extracted by the LDA model when trained on the combined Brown and TASA corpora. Middle row: topics extracted by LDA part of LDA-HMM model. Bottom row: topics extracted by HMM part of LDA-HMM model. Each column represents a single topic/class, and words appear in order of probability in that topic/class. Since some classes give almost all probability to only a few words, a list is terminated when the words account for 90% of the probability mass. Source: Figure 2 of (Griffiths et al. 2004). Used with kind permission of Tom Griffiths.

responsible for syntactic words, and the LDA for semantics words. If we did not have the HMM, the LDA topics would get "polluted" by function words (see top of figure), which is why such words are normally removed during preprocessing.

The model can also help disambiguate when the same word is being used syntactically or semantically. Figure 27.14 shows some examples when the model was applied to the NIPS corpus.[5] We see that the roles of words are distinguished, e.g., "we require the algorithm to *return* a matrix" (verb) vs "the maximal expected *return*" (noun). In principle, a part of speech tagger could disambiguate these two uses, but note that (1) the LDA-HMM method is fully unsupervised (no POS tags were used), and (2) sometimes a word can have the same POS tag, but different senses, e.g., "the left graph" (a synactic role) vs "the graph G" (a semantic role).

The topic of probabilistic models for syntax and semantics is a vast one, which we do not

The TASA corpus is an untagged collection of educational materials consisting of 37,651 documents and 12,190,931 word tokens. Words appearing in fewer than 5 documents were replaced with an asterisk, but punctuation was included. The combined vocabulary was of size 37,202 unique words.

5. NIPS stands for "Neural Information Processing Systems". It is one of the top machine learning conferences. The NIPS corpus volumes 1–12 contains 1713 documents.

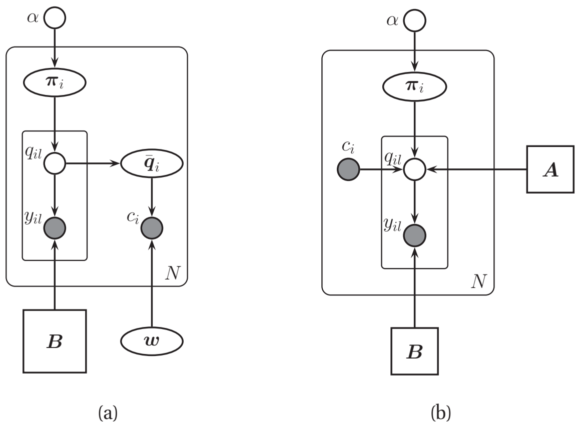


Figure 27.15 (a) Supervised LDA. (b) Discriminative LDA.

have space to delve into any more. See e.g., (Jurafsky and Martin 2008) for further information.

27.4.4 Supervised LDA

In this section, we discuss extensions of LDA to handle side information of various kinds beyond just words.

27.4.4.1 Generative supervised LDA

Suppose we have a variable length sequence of words $y_{il} \in \{1, \ldots, V\}$ as usual, but we also have a class label $c_i \in \{1, \ldots, C\}$. How can we predict c_i from $\mathbf{y}_i$? There are many possible approaches, but most are direct mappings from the words to the class. In some cases, such as **sentiment analysis**, we can get better performance by first performing inference, to try to disambiguate the meaning of words. For example, suppose the goal is to determine if a document is a favorable review of a movie or not. If we encounter the phrase "Brad Pitt was excellent until the middle of the movie", the word "excellent" may lead us to think the review is positive, but clearly the overall sentiment is negative.

One way to tackle such problems is to build a joint model of the form $p(c_i, \mathbf{y}_i|\boldsymbol{\theta})$. (Blei and McAuliffe 2010) proposes an approach, called **supervised LDA**, where the class label c_i is generated from the topics as follows:

$$p(c_i|\overline{\mathbf{q}}_i) = \text{Ber}(\text{sigm}(\mathbf{w}^T\overline{\mathbf{q}}_i)) \tag{27.65}$$

Here $\overline{\mathbf{q}}_i$ is the empirical topic distribution for document i:

$$\overline{q}_{ik} \triangleq \frac{1}{L_i}\sum_{i=1}^{L_i} q_{ilk} \tag{27.66}$$

See Figure 27.15(a) for an illustration.

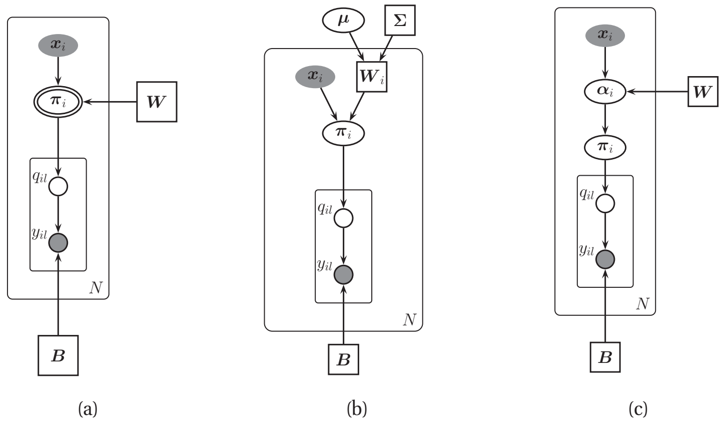


Figure 27.16 Discriminative variants of LDA. (a) Mixture of experts aka MR-LDA. The double ring denotes a node that $\boldsymbol{\pi}_i$ a deterministic function of its parents. (b) Mixture of experts with random effects. (c) DMR-LDA.

We can fit this model using Monte Carlo EM: run the collapsed Gibbs sampler in the E step, to compute $\mathbb{E}[\overline{q}_{ik}]$, and then use this as the input feature to a standard logistic regression package.

27.4.4.2 Discriminative supervised LDA

An alternative approach, known as **discriminative LDA** (Lacoste-Julien et al. 2009), is shown in Figure 27.15(b). This is a discriminative model of the form $p(\mathbf{y}_i|c_i, \boldsymbol{\theta})$. The only change from regular LDA is that the topic prior becomes input dependent, as follows:

$$p(q_{il}|\boldsymbol{\pi}_i, c_i = c, \boldsymbol{\theta}) = \text{Cat}(\mathbf{A}_c \boldsymbol{\pi}) \tag{27.67}$$

where $\mathbf{A}_c$ is a $K \times K$ stochastic matrix.

So far, we have assumed the "side information" is a single categorical variable c_i. Often we have high dimensional covariates $\mathbf{x}_i \in \mathbb{R}^D$. For example, consider the task of **image tagging**. The idea is that y_{il} represent correlated tags or labels, which we want to predict given $\mathbf{x}_i$. We now discuss several attempts to extend LDA so that it can generate tags given the inputs.

The simplest approach is to use a mixture of experts (Section 11.2.4) with multiple outputs. This is just like LDA except we replace the Dirichlet prior on $\boldsymbol{\pi}_i$ with a deterministic function of the input:

$$\boldsymbol{\pi}_i = \mathcal{S}(\mathbf{W}\mathbf{x}_i) \tag{27.68}$$

In (Law et al. 2010), this is called **multinomial regression LDA**. See Figure 27.16(a). Eliminating the deterministic $\boldsymbol{\pi}_i$ we have

$$p(q_{il}|\mathbf{x}_i, \mathbf{W}) = \text{Cat}(\mathcal{S}(\mathbf{W}\mathbf{x}_i)) \tag{27.69}$$

We can fit this with EM in the usual way. However, (Law et al. 2010) suggest an alternative. First fit an unsupervised LDA model based only on $\mathbf{y}_i$; then treat the inferred $\boldsymbol{\pi}_i$ as data, and fit a multinomial logistic regression model mapping $\mathbf{x}_i$ to $\boldsymbol{\pi}_i$. Although this is fast, fitting LDA in an unsupervised fashion does not necessarily result in a discriminative set of latent variables, as discussed in (Blei and McAuliffe 2010).

There is a more subtle problem with this model. Since $\boldsymbol{\pi}_i$ is a deterministic function of the inputs, it is effectively observed, rendering the q_{il} (and hence the tags y_{il}) independent. In other words,

$$p(\mathbf{y}_i|\mathbf{x}_i, \boldsymbol{\theta}) = \prod_{l=1}^{L_i} p(y_{il}|\mathbf{x}_i, \boldsymbol{\theta}) = \prod_{l=1}^{L_i} \sum_k p(y_{il}|q_{il} = k, \mathbf{B}) p(q_{il} = k|\mathbf{x}_i, \mathbf{W}) \tag{27.70}$$

This means that if we observe the value of one tag, it will have no influence on any of the others. This may explain why the results in (Law et al. 2010) only show negligible improvement over predicting each tag independently.

One way to induce correlations is to make $\mathbf{W}$ a random variable. The resulting model is shown in Figure 27.16(b). We call this a **random effects mixture of experts**. We typically assume a Gaussian prior on $\mathbf{W}_i$. If $\mathbf{x}_i = 1$, then $p(q_{il}|\mathbf{x}_i, \mathbf{w}_i) = \text{Cat}(\mathcal{S}(\mathbf{w}_i))$, so we recover the correlated topic model. It is possible to extend this model by adding Markovian dynamics to the q_{il} variables. This is called a **conditional topic random field** (Zhu and Xing 2010).

A closely related approach, known as **Dirichlet multinomial regression LDA** (Mimno and McCallum 2008), is shown in Figure 27.16(c). This is identical to standard LDA except we make $\boldsymbol{\alpha}$ a function of the input

$$\boldsymbol{\alpha}_i = \exp(\mathbf{W}\mathbf{x}_i) \tag{27.71}$$

where $\mathbf{W}$ is a $K \times D$ matrix. Eliminating the deterministic $\boldsymbol{\alpha}_i$ we have

$$\boldsymbol{\pi}_i \sim \text{Dir}(\exp(\mathbf{W}\mathbf{x}_i)) \tag{27.72}$$

Unlike (Law et al. 2010), this model allows information to flow between tags via the latent $\boldsymbol{\pi}_i$.

A variant of this model, where $\mathbf{x}_i$ corresponds to a bag of discrete labels and $\boldsymbol{\pi}_i \sim \text{Dir}(\boldsymbol{\alpha} \odot \mathbf{x}_i)$, is known as **labeled LDA** (Ramage et al. 2009). In this case, the labels $\mathbf{x}_i$ are in 1:1 correspondence with the latent topics, which makes the resulting topics much more interpretable. An extension, known as **partially labeled LDA** (Ramage et al. 2011), allows each label to have multiple latent sub-topics; this model includes LDA, labeled LDA and a multinomial mixture model as special cases.

27.4.4.3 Discriminative categorical PCA

An alternative to using LDA is to expand the categorical PCA model with inputs, as shown in Figure 27.17(a). Since the latent space is now real-valued, we can use simple linear regression for the input-hidden mapping. For the hidden-output mapping, we use traditional catPCA:

$$p(\mathbf{z}_i|\mathbf{x}_i, \mathbf{V}) = \mathcal{N}(\mathbf{V}\mathbf{x}_i, \boldsymbol{\Sigma}) \tag{27.73}$$

$$p(\mathbf{y}_i|\mathbf{z}_i, \mathbf{W}) = \prod_l \text{Cat}(y_{il}|\mathcal{S}(\mathbf{W}\mathbf{z}_i)) \tag{27.74}$$

This model is essentially a probabilistic neural network with one hidden layer, as shown in Figure 27.17(b), but with exchangeable output (e.g., to handle variable numbers of tags). The

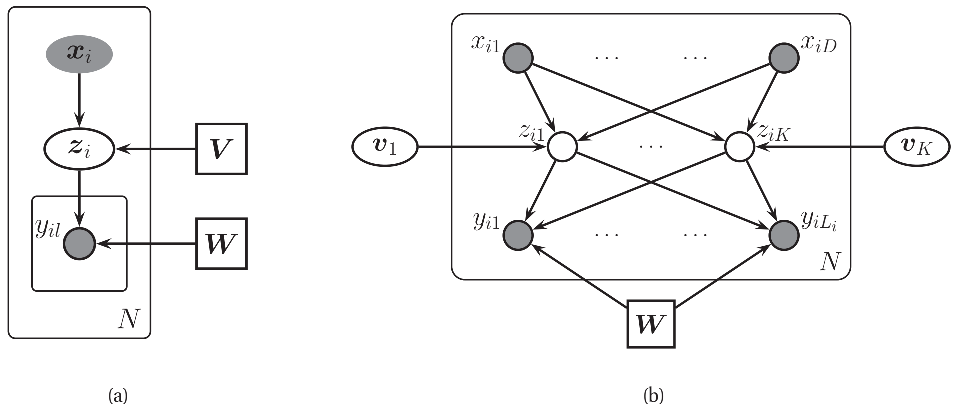


Figure 27.17 (a) Categorical PCA with inputs and exchangeable outputs. (b) Same as (a), but with the vector nodes expanded out.

key difference from a neural net is that information can flow between the y_{il}'s via the latent **bottleneck layer** $\mathbf{z}_i$. This should work better than a conventional neural net when the output labels are highly correlated, even after conditioning on the features; this problem frequently arises in multi label classification. Note that we could allow a direct $\mathbf{x}_i$ to $\mathbf{y}_i$ arc, but this would require too many parameters if the number of labels is large.[6]

We can fit this model with a small modification of the variational EM algorithm in Section 12.4. If we use this model for regression, rather than classification, we can perform the E step exactly, by modifying the EM algorithm for factor analysis. (Ma et al. 1997) reports that this method converges faster than standard backpropagation.

We can also extend the model so that the prior on $\mathbf{z}_i$ is a mixture of Gaussians using input-dependent means. If the output is Gaussian, this corresponds to a mixture of discriminative factor analysers (Fokoue 2005; Zhou and Liu 2008). If the output is categorical, this would be an (as yet unpublished) model, which we could call "discriminative mixtures of categorical factor analyzers".

27.5 LVMs for graph-structured data

Another source of discrete data is when modeling graph or network structures. To see the connection, recall that any graph on D nodes can be represented as a $D \times D$ **adjacency matrix G**, where $G(i, j) = 1$ iff there is an edge from node i to node j. Such matrices are binary, and often very sparse. See Figure 27.19 for an example.

Graphs arise in many application areas, such as modeling social networks, protein-protein interaction networks, or patterns of disease transmission between people or animals. There are usually two primary goals when analysing such data: first, try to discover some "interesting

6. A non-probabilistic version of this idea, using squared loss, was proposed in (Ji et al. 2010). This is similar to a linear feed-forward neural network with an additional edge from $\mathbf{x}_i$ directly to $\mathbf{y}_i$.

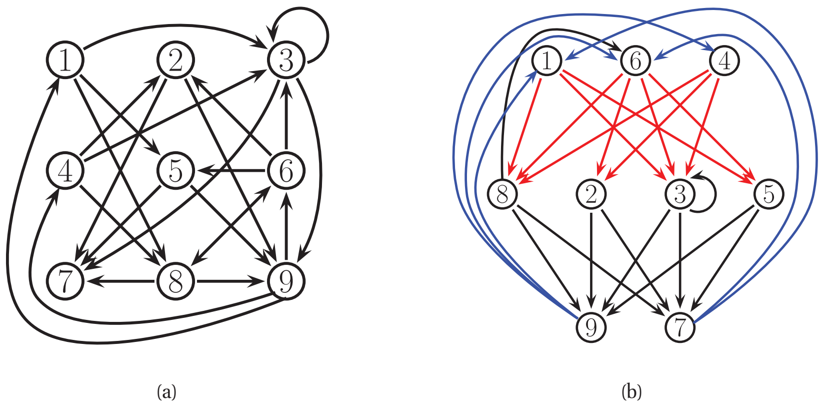


Figure 27.18 (a) A directed graph. (b) The same graph, with the nodes partitioned into 3 groups, making the block structure more apparent.

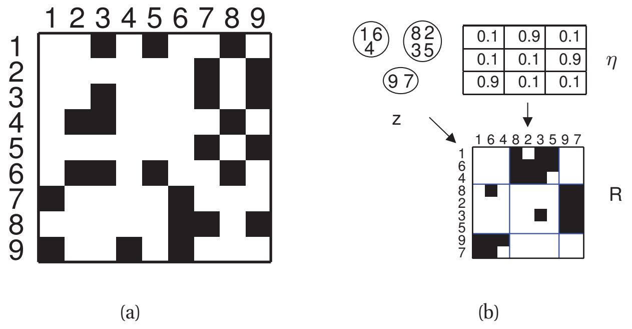


Figure 27.19 (a) Adjacency matrix for the graph in Figure 27.18(a). (b) Rows and columns are shown permuted to show the block structure. We also sketch of how the stochastic block model can generate this graph. From Figure 1 of (Kemp et al. 2006). Used with kind permission of Charles Kemp.

structure" in the graph, such as clusters or communities; second, try to predict which links might occur in the future (e.g., who will make friends with whom). Below we summarize some models that have been proposed for these tasks, some of which are related to LDA. Futher details on these and other approaches can be found in e.g., (Goldenberg et al. 2009) and the references therein.

27.5.1 Stochastic block model

In Figure 27.18(a) we show a directed graph on 9 nodes. There is no apparent structure. However, if we look more deeply, we see it is possible to partition the nodes into three groups or blocks, $B_1 = \{1, 4, 6\}$, $B_2 = \{2, 3, 5, 8\}$, and $B_3 = \{7, 9\}$, such that most of the connections go from nodes in B_1 to B_2, or from B_2 to B_3, or from B_3 to B_1. This is illustrated in Figure 27.18(b).

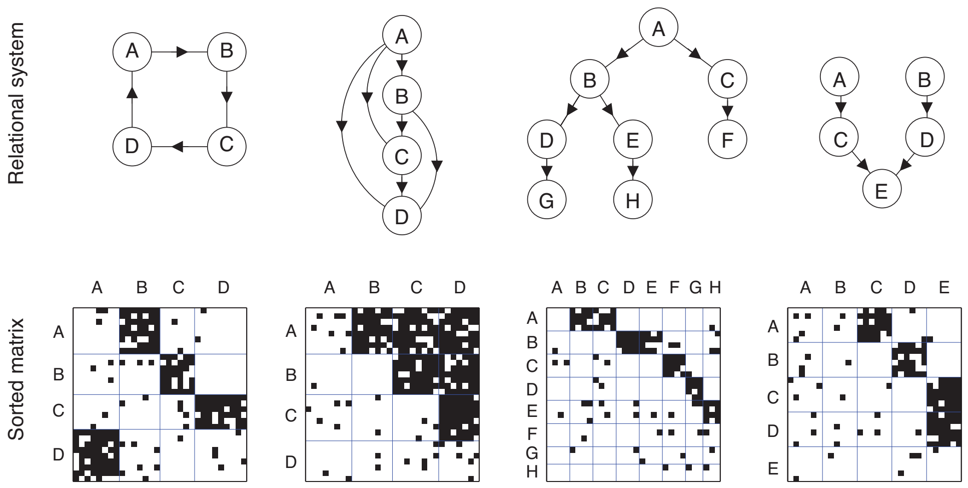


Figure 27.20 Some examples of graphs generated using the stochastic block model with different kinds of connectivity patterns between the blocks. The abstract graph (between blocks) represent a ring, a dominance hierarchy, a common-cause structure, and a common-effect structure. From Figure 4 of (Kemp et al. 2010). Used with kind permission of Charles Kemp.

The problem is easier to understand if we plot the adjacency matrices. Figure 27.19(a) shows the matrix for the graph with the nodes in their original ordering. Figure 27.19(b) shows the matrix for the graph with the nodes in their permtuted ordering. It is clear that there is block structure.

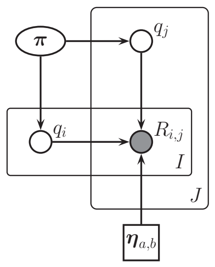
We can make a generative model of block structured graphs as follows. First, for every node, sample a latent block $q_i \sim \text{Cat}(\boldsymbol{\pi})$, where π_k is the probability of choosing block k, for $k = 1 : K$. Second, choose the probability of connecting group a to group b, for all pairs of groups; let us denote this probability by $\eta_{a,b}$. This can come from a beta prior. Finally, generate each edge R_{ij} using the following model:

$$p(R_{ij} = r | q_i = a, q_j = b, \boldsymbol{\eta}) = \text{Ber}(r | \eta_{a,b}) \tag{27.75}$$

This is called the **stochastic block model** (Nowicki and Snijders 2001). Figure 27.21(a) illustrates the model as a DGM, and Figure 27.19(c) illustrates how this model can be used to cluster the nodes in our example.

Note that this is quite different from a conventional clustering problem. For example, we see that all the nodes in block 3 are grouped together, even though there are no connections between them. What they share is the property that they "like to" connect to nodes in block 1, and to receive connections from nodes in block 2. Figure 27.20 illustrates the power of the model for generating many different kinds of graph structure. For example, some social networks have hierarchical structure, which can be modeled by clustering people into different social strata, whereas others consist of a set of cliques.

Unlike a standard mixture model, it is not possible to fit this model using exact EM, because all the latent q_i variables become correlated. However, one can use variational EM (Airoldi et al.



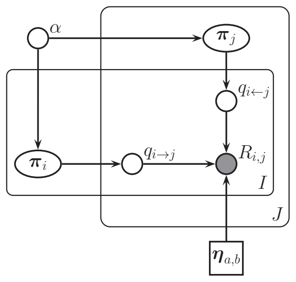


Figure 27.21 (a) Stochastic block model. (b) Mixed membership stochastic block model.

2008), collapsed Gibbs sampling (Kemp et al. 2006), etc. We omit the details (which are similar to the LDA case).

In (Kemp et al. 2006), they lifted the restriction that the number of blocks K be fixed, by replacing the Dirichlet prior on $\boldsymbol{\pi}$ by a Dirichlet process (see Section 25.2.2). This is known as the infinite relational model. See Section 27.6.1 for details.

If we have features associated with each node, we can make a discriminative version of this model, for example by defining

$$p(R_{ij} = r|q_i = a, q_j = b, \mathbf{x}_i, \mathbf{x}_j, \boldsymbol{\theta}) = \text{Ber}(r|\mathbf{w}_{a,b}^T f(\mathbf{x}_i, \mathbf{x}_j)) \tag{27.76}$$

where $f(\mathbf{x}_i, \mathbf{x}_j)$ is some way of combining the feature vectors. For example, we could use concatenation, $[\mathbf{x}_i, \mathbf{x}_j]$, or elementwise product $\mathbf{x}_i \otimes \mathbf{x}_j$ as in supervised LDA. The overall model is like a relational extension of the mixture of experts model.

27.5.2 Mixed membership stochastic block model

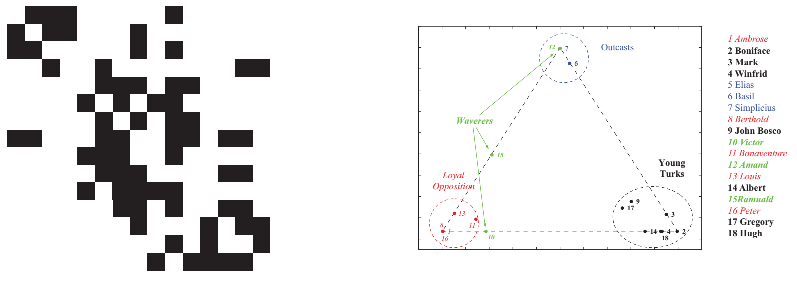
In (Airoldi et al. 2008), they lifted the restriction that each node only belong to one cluster. That is, they replaced $q_i \in \{1, \ldots, K\}$ with $\boldsymbol{\pi}_i \in S_K$. This is known as the **mixed membership stochastic block model**, and is similar in spirit to **fuzzy clustering** or **soft clustering**. Note that π_{ik} is not the same as $p(z_i = k|\mathcal{D})$; the former represents **ontological uncertainty** (to what degree does each object belong to a cluster) wheras the latter represents **epistemological uncertainty** (which cluster does an object belong to). If we want to combine epistemological and ontological uncertainty, we can compute $p(\boldsymbol{\pi}_i|\mathcal{D})$.

In more detail, the generative process is as follows. First, each node picks a distribution over blocks, $\boldsymbol{\pi}_i \sim \text{Dir}(\boldsymbol{\alpha})$. Second, choose the probability of connecting group a to group b, for all pairs of groups, $\eta_{a,b} \sim \beta(\alpha, \beta)$. Third, for each edge, sample two discrete variables, one for each direction:

$$q_{i\to j} \sim \text{Cat}(\boldsymbol{\pi}_i), \;\; q_{i\leftarrow j} \sim \text{Cat}(\boldsymbol{\pi}_j) \tag{27.77}$$

Finally, generate each edge R_{ij} using the following model:

$$p(R_{ij} = 1|q_{i\to j} = a, q_{i\leftarrow j} = b, \boldsymbol{\eta}) = \eta_{a,b} \tag{27.78}$$



(a) (b)

Figure 27.22 (a) Who-likes-whom graph for Sampson's monks. (b) Mixed membership of each monk in one of three groups. From Figures 2-3 of (Airoldi et al. 2008). Used with kind permission of Edo Airoldi.

See Figure 27.21(b) for the DGM.

Unlike the regular stochastic block model, each node can play a different role, depending on who it is connecting to. As an illustration of this, we will consider a data set that is widely used in the social networks analysis literature. The data concerns who-likes-whom amongst of group of 18 monks. It was collected by hand in 1968 by Sampson (Sampson 1968) over a period of months. (These days, in the era of social media such as Facebook, a social network with only 18 people is trivially small, but the methods we are discussing can be made to scale.) Figure 27.22(a) plots the raw data, and Figure 27.22(b) plots $\mathbb{E}\left[\boldsymbol{\pi}\right]_i$ for each monk, where $K = 3$. We see that most of the monk belong to one of the three clusters, known as the "young turks", the "outcasts" and the "loyal opposition". However, some individuals, notably monk 15, belong to two clusters; Sampson called these monks the "waverers". It is interesting to see that the model can recover the same kinds of insights as Sampson derived by hand.

One prevalent problem in social network analysis is missing data. For example, if $R_{ij} = 0$, it may be due to the fact that person i and j have not had an opportunity to interact, or that data is not available for that interaction, as opposed to the fact that these people don't want to interact. In other words, *absence of evidence is not evidence of absence*. We can model this by modifying the observation model so that with probability ρ, we generate a 0 from the background model, and we only force the model to explain observed 0s with probability $1 - \rho$. In other words, we robustify the observation model to allow for outliers, as follows:

$$p(R_{ij} = r | q_{i \to j} = a, q_{i \leftarrow j} = b, \boldsymbol{\eta}) = \rho \delta_0(r) + (1 - \rho) \text{Ber}(r | \eta_{a,b}) \tag{27.79}$$

See (Airoldi et al. 2008) for details.

27.5.3 Relational topic model

In many cases, the nodes in our network have atttributes. For example, if the nodes represent academic papers, and the edges represent citations, then the attributes include the text of the document itself. It is therefore desirable to create a model that can explain the text and the link structure concurrently. Such a model can predict links given text, or even vice versa.

The **relational topic model** (RTM) (Chang and Blei 2010) is one way to do this. This is a

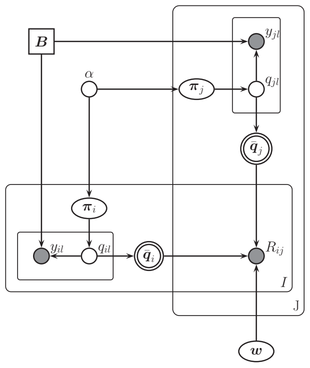


Figure 27.23 DGM for the relational topic model.

simple extension of supervised LDA (Section 27.4.4.1), where the response variable R_{ij} (which represents whether there is an edge between nodes i and j) is modeled as follows:

$$p(R_{ij} = 1|\overline{\mathbf{q}}_i, \overline{\mathbf{q}}_j, \boldsymbol{\theta}) = \text{sigm}(\mathbf{w}^T(\overline{\mathbf{q}}_i \otimes \overline{\mathbf{q}}_j) + w_0) \tag{27.80}$$

Recall that $\overline{\mathbf{q}}_i$ is the empirical topic distribution for document i, $\overline{q}_{ik} \triangleq \frac{1}{L_i}\sum_{i=1}^{L_i} q_{ilk}$. See Figure 27.23

Note that it is important that R_{ij} depend on the actual topics chosen, $\overline{\mathbf{q}}_i$ and $\overline{\mathbf{q}}_j$, and not on the topic distributions, $\boldsymbol{\pi}_i$ and $\boldsymbol{\pi}_j$, otherwise predictive performance is not as good. The intuitive reason for this is as follows: if R_{ij} is a child of $\boldsymbol{\pi}_i$ and $\boldsymbol{\pi}_j$, it will be treated as just another word, similar to the q_{il}'s and y_{il}'s; but since there are many more words than edges, the graph structure information will get "washed out". By making R_{ij} a child of $\overline{\mathbf{q}}_i$ and $\overline{\mathbf{q}}_j$, the graph information can influence the choice of topics more directly.

One can fit this model in a manner similar to SLDA. See (Chang and Blei 2010) for details. The method does better at predicting missing links than the simpler approach of first fitting an LDA model, and then using the $\overline{\mathbf{q}}_i$'s as inputs to a logistic regression problem. The reason is analogous to the superiority of partial least squares (Section 12.5.2) to PCA+ linear regression, namely that the RTM learns a latent space that is forced to be predictive of the graph structure and words, whereas LDA might learn a latent space that is not useful for predicting the graph.

27.6 LVMs for relational data

Graphs can be used to represent data which represents the relation amongst variables of a certain type, e.g., friendship relationships between people. But often we have multiple types of objects, and multiple types of relations. For example, Figure 27.24 illustrates two relations, one between people and people, and one between people and movies.

In general, we define a k-ary **relation** R as a subset of k-tuples of the appropriate types:

$$R \subseteq T_1 \times T_2 \times \cdots \times T_k \tag{27.81}$$

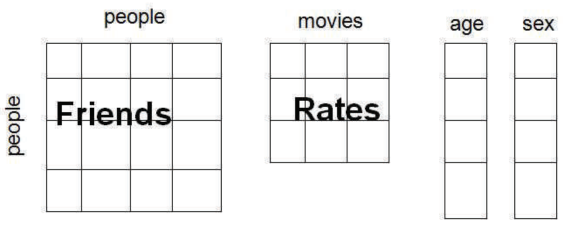


Figure 27.24 Example of relational data. There are two types of objects, *people* and *movies*; one 2-ary relation, *friends*: *people* $\times$ *people* $\rightarrow \{0, 1\}$ and one 2-ary function, *rates*: *people* $\times$ *movie* $\rightarrow \mathbb{R}$. *Age* and *sex* are attributes (unary functions) of the *people* class.

where T_i are sets or types. A binary, pairwise or dyadic relation is a relation defined on pairs of objects. For example, the *seen* relation between people and movies might be represented as the set of movies that people have seen. We can either represent this explicitly as a set, such as

```
seen  = { (Bob, StarWars), (Bob, TombRaider), (Alice, Jaws)}
```

or implicitly, using an indicator function for the set:

```
seen(Bob, StarWars)=1, seen(Bob, TombRaider)=1, seen(Alice, Jaws)=1
```

A relation between two entities of types T^1 and T^2 can be represented as a binary function $R : T^1 \times T^2 \rightarrow \{0, 1\}$, and hence as a binary matrix. This can also be represented as a bipartite graph, in which we have nodes of two types. If $T^1 = T^2$, this becomes a regular directed graph, as in Section 27.5. However, there are some situations that are not so easily modelled by graphs, but which can still be modelled by relations. For example, we might have a ternary relation, $R : T^1 \times T^1 \times T^2 \rightarrow \{0, 1\}$, where, say, $R(i, j, k) = 1$ iff protein i interacts with protein j when chemical k is present. This can be modelled by a 3d binary matrix. We will give some examples of this in Section 27.6.1.

Making probabilistic models of relational data is called **statistical relational learning** (Getoor and Taskar 2007). One approach is to directly model the relationship between the variables using graphical models; this is known as **probabilistic relational modeling**. Another approach is to use latent variable models, as we discuss below.

27.6.1 Infinite relational model

It is straightforward to extend the stochastic block model to model relational data: we just associate a latent variable $q_i^t \in \{1, \ldots, K_t\}$ with each entity i of each type t. We then define the probability of the relation holding between specific entities by looking up the probability of the relation holding between entities of that type. For example, if $R : T^1 \times T^1 \times T^2 \rightarrow \{0, 1\}$, we have

$$p(R(i, j, k) = 1 | q_i^1 = a, q_j^1 = b, q_k^2 = c, \boldsymbol{\eta}) = \eta_{a,b,c} \tag{27.82}$$

If we allow the number of clusters K_t for each type to be unbounded, by using a Dirichlet process, the model is called the **infinite relational model** (IRM) (Kemp et al. 2006). An essentially

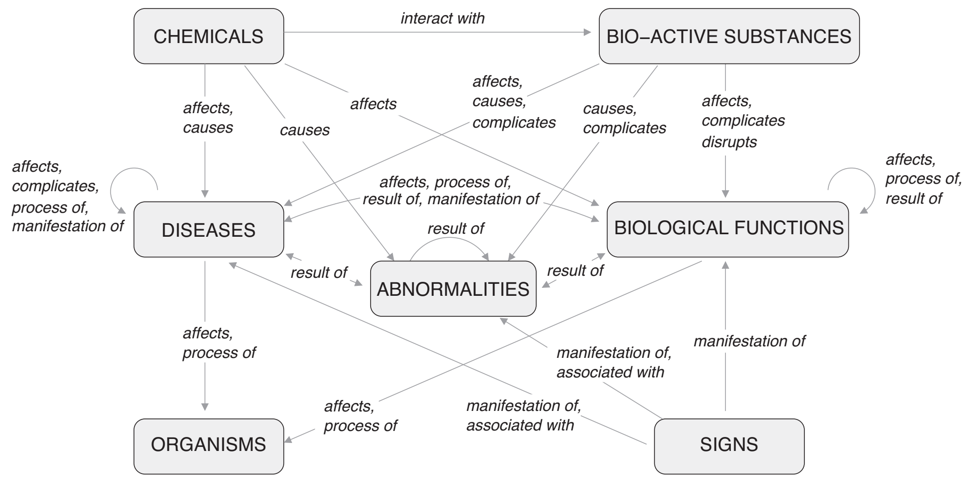

CHEMICALS
interact with
BIO–ACTIVE SUBSTANCES
affects, causes
causes
affects
affects, causes, complicates
causes, complicates
affects, complicates disrupts
affects, complicates, process of, manifestation of
affects, process of, result of, manifestation of
affects, process of, result of
DISEASES
result of
BIOLOGICAL FUNCTIONS
result of
ABNORMALITIES
result of
affects, process of
manifestation of
manifestation of, associated with
affects, process of
manifestation of, associated with
ORGANISMS
SIGNS

Figure 27.25 Illustration of an ontology learned by IRM applied to the Unified Medical Language System. The boxes represent 7 of the 14 concept clusters. Predicates that belong to the same cluster are grouped together, and associated with edges to which they pertain. All links with weight above 0.8 have been included. From Figure 9 of (Kemp et al. 2010). Used with kind permission of Charles Kemp.

identical model, under the name **infinite hidden relational model** (IHRM), was concurrently proposed in (Xu et al. 2006). We can fit this model with variational Bayes (Xu et al. 2006, 2007) or collapsed Gibbs sampling (Kemp et al. 2006). Rather than go into algorithmic detail, we just sketch some interesting applications.

27.6.1.1 Learning ontologies

An **ontology** refers to an organisation of knowledge. In AI, ontologies are often built by hand (see e.g., (Russell and Norvig 2010)), but it is interesting to try and learn them from data. In (Kemp et al. 2006), they show how this can be done using the IRM.

The data comes from the Unified Medical Language System (McCray 2003), which defines a semantic network with 135 concepts (such as "disease or syndrome", "diagnostic procedure", "animal"), and 49 binary predicates (such as "affects", "prevents"). We can represent this as a ternary relation $R : T^1 \times T^1 \times T^2 \to \{0, 1\}$, where T^1 is the set of concepts and T^2 is the set of binary predicates. The result is a 3d cube. We can then apply the IRM to partition the cube into regions of roughly homogoneous response. The system found 14 concept clusters and 21 predicate clusters. Some of these are shown in Figure 27.25. The system learns, for example, that biological functions affect organisms (since $\eta_{a,b,c} \approx 1$ where a represents the biological function cluster, b represents the organism cluster, and c represents the affects cluster).

27.6.1.2 Clustering based on relations and features

We can also use IRM to cluster objects based on their relations and their features. For example, (Kemp et al. 2006) consider a political dataset (from 1965) consisting of 14 countries, 54 binary

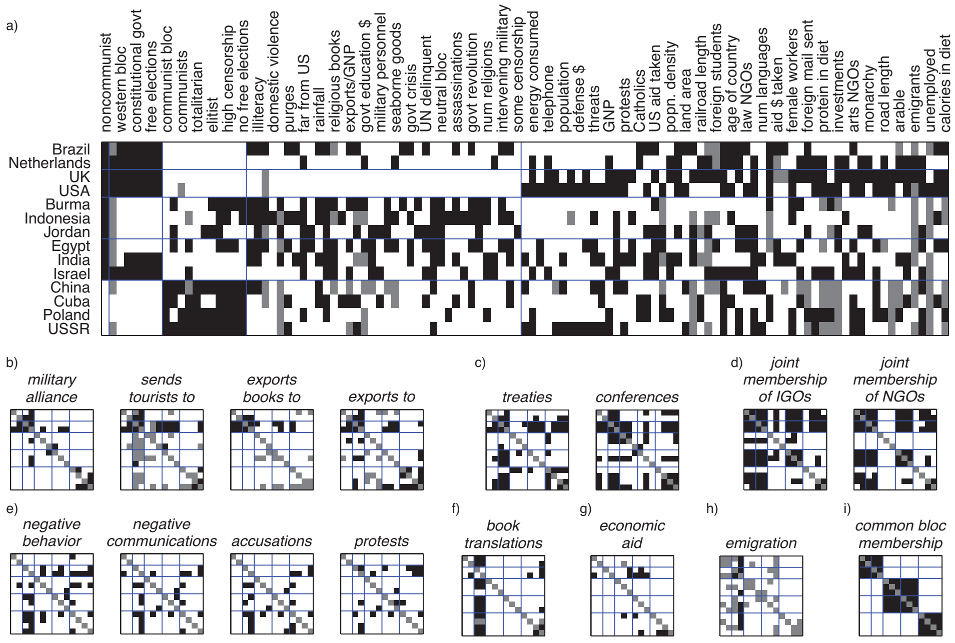


Figure 27.26 Illustration of IRM applied to some political data containing features and pairwise interactions. Top row (a). the partition of the countries into 5 clusters and the features into 5 clusters. Every second column is labelled with the name of the corresponding feature. Small squares at bottom (a-i): these are 8 of the 18 clusters of interaction types. From Figure 6 of (Kemp et al. 2006). Used with kind permission of Charles Kemp.

predicates representing interaction types between countries (e.g., “sends tourists to”, “economic aid”), and 90 features (e.g., “communist”, “monarchy”). To create a binary dataset, real-valued features were thresholded at their mean, and categorical variables were dummy-encoded. The data has 3 types: T^1 represents countries, T^2 represents interactions, and T^3 represents features. We have two relations: $R^1 : T^1 \times T^1 \times T^2 \rightarrow \{0, 1\}$, and $R^2 : T^1 \times T^3 \rightarrow \{0, 1\}$. (This problem therefore combines aspects of both the biclustering model and the ontology discovery model.) When given multiple relations, the IRM treats them as conditionally independent. In this case, we have

$$p(\mathbf{R}^1, \mathbf{R}^2, \mathbf{q}^1, \mathbf{q}^2, \mathbf{q}^3|\boldsymbol{\theta}) = p(\mathbf{R}^1|\mathbf{q}^1, \mathbf{q}^2, \boldsymbol{\theta})p(\mathbf{R}^2|\mathbf{q}^1, \mathbf{q}^3, \boldsymbol{\theta}) \tag{27.83}$$

The results are shown in Figure 27.26. The IRM divides the 90 features into 5 clusters, the first of which contains “noncommunist”, which captures one of the most important aspects of this Cold-War era dataset. It also clusters the 14 countries into 5 clusters, reflecting natural geo-political groupings (e.g., US and UK, or the Communist Bloc), and the 54 predicates into 18 clusters, reflecting similar relationships (e.g., “negative behavior and “accusations”).

27.6.2 Probabilistic matrix factorization for collaborative filtering

As discussed in Section 1.3.4.2, collaborative filtering (CF) requires predicting entries in a matrix $R : T^1 \times T^2 \to \mathbb{R}$, where for example $R(i, j)$ is the rating that user i gave to movie j. Thus we see that CF is a kind of relational learning problem (and one with particular commercial importance).

Much of the work in this area makes use of the data that Netflix made available in their competition. In particular, a large 17,770 $\times$ 480,189 movie x user ratings matrix is provided. The full matrix would have $\sim 8.6 \times 10^9$ entries, but only 100,480,507 (about 1%) of the entries are observed, so the matrix is extremely sparse. In addition the data is quite imbalanced, with many users rating fewer than 5 movies, and a few users rating over 10,000 movies. The validation set is 1,408,395 (movie,user) pairs. Finally, there is a separate test set with 2,817,131 (movie,user) pairs, for which the ranking is known but withheld from contestants. The performance measure is root mean square error:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X(m_i, u_i) - \hat{X}(m_i, u_i))^2} \tag{27.84}$$

where $X(m_i, u_i)$ is the true rating of user u_i on movie m_i, and $\hat{X}(m_i, u_i)$ is the prediction. The baseline system, known as Cinematch, had an RMSE on the training set of 0.9514, and on the test set of 0.9525. To qualify for the grand prize, teams needed to reduce the test RMSE by 10%, i.e., get a test RMSE of 0.8563 or less. We will discuss some of the basic methods used byt the winning team below.

Since the ratings are drawn from the set $\{0, 1, 2, 3, 4, 5\}$, it is tempting to use a categorical observation model. However, this does not capture the fact that the ratings are ordered. Although we could use an ordinal observation model, in practice people use a Gaussian observation model for simplicity. One way to make the model better match the data is to pass the model's predicted mean response through a sigmoid, and then to map the $[0, 1]$ interval to $[0, 5]$ (Salakhutdinov and Mnih 2008). Alternatively we can make the data a better match to the Gaussian model by transforming the data using $R_{ij} = \sqrt{6 - R_{ij}}$ (Aggarwal and Merugu 2007).

We could use the IRM for the CF task, by associating a discrete latent variable for each user q_i^u and for each movie or video q_j^v, and then defining

$$p(R_{ij} = r|q_i^u = a, q_j^v = b, \boldsymbol{\theta}) = \mathcal{N}(r|\mu_{a,b}, \sigma^2) \tag{27.85}$$

This is just another example of co-clustering. We can also extend the model to generate side information, such as attributes about each user and/or movie. See Figure 27.27 for an illustration.

Another possibility is to replace the discrete latent variables with continuous latent variables $\boldsymbol{\pi}_i^u \in S_{K_u}$ and $\boldsymbol{\pi}_j^v \in S_{K_v}$. However, it has been found (see e.g., (Shan and Banerjee 2010)) that one obtains much better results by using unconstrained real-valued latent factors for each user $\mathbf{u}_i \in \mathbb{R}^K$ and each movie $\mathbf{v}_j \in \mathbb{R}^K$.[7] We then use a likelihood of the form

$$p(R_{ij} = r|\mathbf{u}_i, \mathbf{v}_j) = \mathcal{N}(r|\mathbf{u}_i^T\mathbf{v}_j, \sigma^2) \tag{27.86}$$

7. Good results with discrete latent variables have been obtained on some datasets that are smaller than Netflix, such as MovieLens and EachMovie. However, these datasets are much easier to predict, because there is less imbalance between the number of reviews performed by different users (in Netflix, some users have rated more than 10,000 movies, whereas others have rated less than 5).

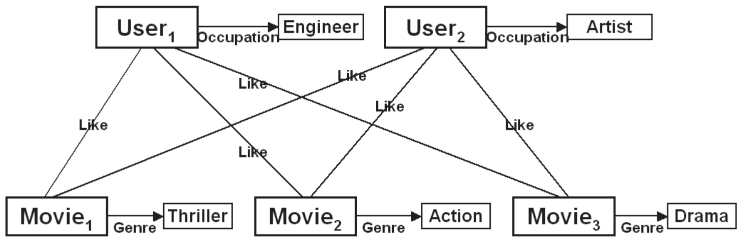
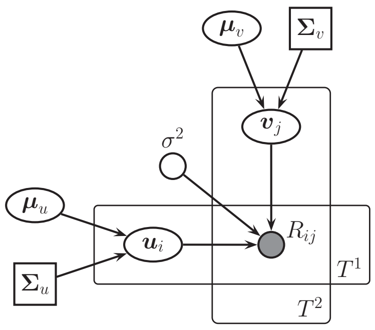
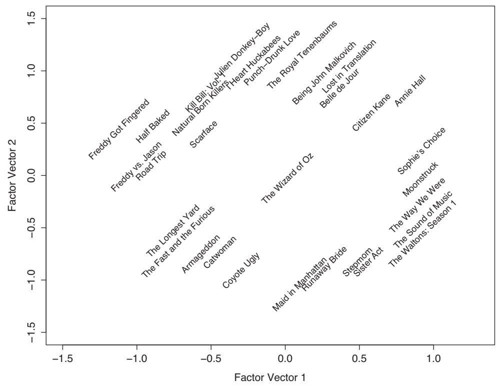


Figure 27.27 Visualization of a small relational dataset, where we have one relation, likes(user, movie), and features for movies (here, genre) and users (here, occupation). From Figure 5 of (Xu et al. 2008). Used with kind permission of Zhao Xu.



(a)



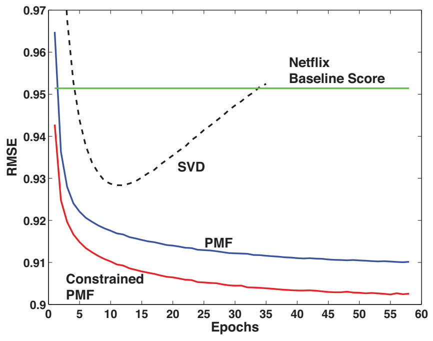
(b)

Figure 27.28 (a) A DGM for probabilistic matrix factorization. (b) Visualization of the first two factors in the PMF model estimated from the Netflix challenge data. Each movie j is plotted at the location specified $\hat{\mathbf{v}}_j$. On the left we have low-brow humor and horror movies (*Half Baked*, *Freddy vs Jason*), and on the right we have more serious dramas (*Sophie's Choice*, *Moonstruck*). On the top we have critically acclaimed independent movies (*Punch-Drunk Love*, *I Heart Huckabees*), and on the bottom we have mainstream Hollywood blockbusters (*Armageddon*, *Runway Bride*). The *Wizard of Oz* is right in the middle of these axes. From Figure 3 of (Koren et al. 2009). Used with kind permission of Yehuda Koren.

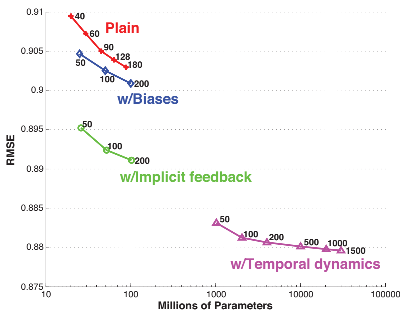
This has been called **probabilistic matrix factorization** (PMF) (Salakhutdinov and Mnih 2008). See Figure 27.28(a) for the DGM. The intuition behind this method is that each user and each movie get embedded into the same low-dimensional continuous space (see Figure 27.28(b)). If a user is close to a movie in that space, they are likely to rate it highly. All of the best entries in the Netflix competition used this approach in one form or another.[8]

PMF is closely related to the SVD. In particular, if there is no missing data, then computing the MLE for the $\mathbf{u}_i$'s and the $\mathbf{v}_j$'s is equivalent to finding a rank K approximation to $\mathbf{R}$. However, as soon as we have missing data, the problem becomes non-convex, as shown in

8. The winning entry was actually an ensemble of different methods, including PMF, nearest neighbor methods, etc.



(a)



(b)

Figure 27.29 (a) RMSE on the validation set for different PMF variants vs number of passes through the data. "SVD" is the unregularized version, $\lambda_U = \lambda_V = 0$. "PMF1" corresponds to $\lambda_U = 0.01$ and $\lambda_V = 0.001$, while "PMF2" corresponds to $\lambda_U = 0.001$ and $\lambda_V = 0.0001$. "PMFA1" corresponds to a version where the mean and diagonal covariance of the Gaussian prior were learned from data. From Figure 2 of (Salakhutdinov and Mnih 2008). Used with kind permission of Ruslan Salakhutdinov. (b) RMSE on the test set (quiz portion) vs number of parameters for several different models. "Plain" is the baseline PMF with suitably chosen λ_U, λ_V. "With biases" adds f_i and g_j offset terms. "With implicit feedback" "With temporal dynamics" allows the offset terms to change over time. The Netflix baseline system achieves an RMSE of 0.9514, and the grand prize's required accuracy is 0.8563 (which was obtained on 21 September 2009). Figure generated by `netflixResultsPlot`. From Figure 4 of (Koren et al. 2009). Used with kind permission of Yehuda Koren.

(Srebro and Jaakkola 2003), and standard SVD methods cannot be applied. (Recall that in the Netflix challenge, only about 1% of the matrix is observed.)

The most straightforward way to fit the PMF model is to minimize the overall NLL:

$$J(\mathbf{U}, \mathbf{V}) = -\log p(\mathbf{R}|\mathbf{U}, \mathbf{V}, \mathbf{O}) = -\log \left(\prod_{i=1}^{N} \prod_{j=1}^{M} \left[\mathcal{N}(R_{ij}|\mathbf{u}_i^T \mathbf{v}_j, \sigma^2) \right]^{\mathbb{I}(O_{ij}=1)} \right) \tag{27.87}$$

where $O_{ij} = 1$ if user i has seen movie j. Since this is non-convex, we can just find a locally optimal MLE. Since the Netflix data is so large (about 100 million observed entries), it is common to use stochastic gradient descent (Section 8.5.2) for this task. The gradient for $\mathbf{u}_i$ is given by

$$\frac{dJ}{d\mathbf{u}_i} = \frac{d}{d\mathbf{u}_i} \frac{1}{2} \sum_{ij} \mathbb{I}(O_{ij} = 1)(R_{ij} - \mathbf{u}_i^T \mathbf{v}_j)^2 = - \sum_{j:O_{ij}=1} e_{ij} \mathbf{v}_j \tag{27.88}$$

where $e_{ij} = R_{ij} - \mathbf{u}_i^T \mathbf{v}_j$ is the error term. By stochastically sampling a single movie j that user i has watched, the update takes the following simple form:

$$\mathbf{u}_i = \mathbf{u}_i + \eta e_{ij} \mathbf{v}_j \tag{27.89}$$

where η is the learning rate. The update for $\mathbf{v}_j$ is similar.

Of course, just maximizing the likelihood results in overfitting, as shown in Figure 27.29(a). We can regularize this by imposing Gaussian priors:

$$p(\mathbf{U},\mathbf{V}) = \prod_i \mathcal{N}(\mathbf{u}_i|\boldsymbol{\mu}_u,\boldsymbol{\Sigma}_u)\prod_j \mathcal{N}(\mathbf{v}_j|\boldsymbol{\mu}_v,\boldsymbol{\Sigma}_v) \tag{27.90}$$

If we use $\boldsymbol{\mu}_u = \boldsymbol{\mu}_v = \mathbf{0}$, $\boldsymbol{\Sigma}_u = \sigma_U^2 \mathbf{I}_K$, and $\boldsymbol{\Sigma}_v = \sigma_V^2 \mathbf{I}_K$, the new objective becomes

$$\begin{aligned} J(\mathbf{U},\mathbf{V}) &= -\log p(\mathbf{R},\mathbf{U},\mathbf{V}|\mathbf{O},\boldsymbol{\theta}) & (27.91)\\ &= \sum_i\sum_j \mathbb{I}(O_{ij}=1)(R_{ij}-\mathbf{u}_i^T\mathbf{v}_j)^2 \\ &\quad +\lambda_U\sum_i ||\mathbf{u}_i||_2^2 + \lambda_V\sum_j ||\mathbf{v}_j||_2^2 + \text{const} & (27.92)\end{aligned}$$

where we have defined $\lambda_U = \sigma^2/\sigma_U^2$ and $\lambda_V = \sigma^2/\sigma_V^2$. By varying the regularizers, we can reduce the effect of overfitting, as shown in Figure 27.29(a). We can find MAP estimates using stochastic gradient descent. We can also compute approximate posteriors using variational Bayes (Ilin and Raiko 2010).

If we use diagonal covariances for the priors, we can penalize each latent dimension by a different amount. Also, if we use non-zero means for the priors, we can account for offset terms. Optimizing the prior parameters $(\boldsymbol{\mu}_u,\boldsymbol{\Sigma}_u,\boldsymbol{\mu}_v,\boldsymbol{\Sigma}_v)$ at the same time as the model parameters $(\mathbf{U},\mathbf{V},\sigma^2)$ is a way to create an adaptive prior. This avoids the need to search for the optimal values of λ_U and λ_V, and gives even better results, as shown in Figure 27.29(a).

It turns out that much of the variation in the data can be explained by movie-specific or user-specific effects. For example, some movies are popular for all types of users. And some users give low scores for all types of movies. We can model this by allowing for user and movie specific offset or bias terms as follows:

$$p(R_{ij}=r|\mathbf{u}_i,\mathbf{v}_j,\boldsymbol{\theta}) = \mathcal{N}(r|\mathbf{u}_i^T\mathbf{v}_j+\mu+f_i+g_j,\sigma^2) \tag{27.93}$$

where μ is the overall mean, f_i is the user bias, g_j is the movie bias, and $\mathbf{u}_i^T\mathbf{v}_j$ is the interaction term. This is equivalent to applying PMF just to the residual matrix, and gives much better results, as shown in Figure 27.29(b). We can estimate the f_i, g_j and μ terms using stochastic gradient descent, just as we estimated $\mathbf{U}$, $\mathbf{V}$ and $\boldsymbol{\theta}$.

We can also allow the bias terms to evolve over time, to reflect the changing preferences of users (Koren 2009b). This is important since in the Netflix competition, the test data was more recent than the training data. Figure 27.29(b) shows that allowing for temporal dynamics can help a lot.

Often we also have **side information** of various kinds. In the Netflix competition, entrants knew which movies the user had rated in the test set, even though they did not know the values of these ratings. That is, they knew the value of the (dense) $\mathbf{O}$ matrix even on the test set. If a user chooses to rate a movie, it is likely because they have seen it, which in turns means they thought they would like it. Thus the very act of rating reveals information. Conversely, if a user chooses not rate a movie, it suggests they knew they would not like it. So the data is not missing at random (see e.g., (Marlin and Zemel 2009)). Exploiting this can improve performance, as shown in Figure 27.29(b). In real problems, information on the test set is not available. However, we often know which movies the user has watched or declined to watch, even if they did not rate them (this is called **implicit feedback**), and this can be used as useful side information.

Another source of side information concerns the content of the movie, such as the movie genre, the list of the actors, or a synopsis of the plot. This can be denoted by $\mathbf{x}_v$, the features of the video. (In the case where we just have the id of the video, we can treat $\mathbf{x}_v$ as a $|\mathcal{V}|$-dimensional bit vector with just one bit turned on.) We may also know features about the user, which we can denote by $\mathbf{x}_u$. In some cases, we only know if the user clicked on the video or not, that is, we may not have a numerical rating. We can then modify the model as follows:

$$p(R(u,v)|\mathbf{x}_u, \mathbf{x}_v, \boldsymbol{\theta}) = \mathrm{Ber}(R(u,v)|(\mathbf{U}\mathbf{x}_u)^T(\mathbf{V}\mathbf{x}_v)) \tag{27.94}$$

where $\mathbf{U}$ is a $|\mathcal{U}| \times K$ matrix, and $\mathbf{V}$ is a $|\mathcal{V}| \times K$ matrix (we can incorporate an offset term by appending a 1 to $\mathbf{x}_u$ and $\mathbf{x}_v$ in the usual way). A method for computing the approximate posterior $p(\mathbf{U}, \mathbf{V}|\mathcal{D})$ in an online fashion, using ADF and EP, was described in (Stern et al. 2009). This was implemented by Microsoft and has been deployed to predict click through rates on all the ads used by Bing.

Unfortunately, fitting this model just from positive binary data can result in an over prediction of links, since no negative examples are included. Better performance is obtained if one has access to the set of all videos shown to the user, of which at most one was picked; data of this form is known as an **impression log**. In this case, we can use a multinomial model instead of a binary model; in (Yang et al. 2011), this was shown to work much better than a binary model. To understand why, suppose some is presented with a choice of an action movie starring Arnold Schwarzenegger, an action movie starring Vin Diesel, and a comedy starring Hugh Grant. If the user picks Arnold Schwarzenegger, we learn not only that they like prefer action movies to comedies, but also that they prefer Schwarzenegger to Diesel. This is more informative than just knowing that they like Schwarzenegger and action movies.

27.7 Restricted Boltzmann machines (RBMs)

So far, all the models we have proposed in this chapter have been representable by directed graphical models. But some models are better represented using undirected graphs. For example, the **Boltzmann machine** (Ackley et al. 1985) is a pairwise MRF with hidden nodes $\mathbf{h}$ and visible nodes $\mathbf{v}$, as shown in Figure 27.30(a). The main problem with the Boltzmann machine is that exact inference is intractable, and even approximate inference, using e.g., Gibbs sampling, can be slow. However, suppose we restrict the architecture so that the nodes are arranged in layers, and so that there are no connections between nodes within the same layer (see Figure 27.30(b)). Then the model has the form

$$p(\mathbf{h}, \mathbf{v}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{r=1}^{R} \prod_{k=1}^{K} \psi_{rk}(v_r, h_k) \tag{27.95}$$

where R is the number of visible (response) variables, K is the number of hidden variables, and $\mathbf{v}$ plays the role of $\mathbf{y}$ earlier in this chapter. This model is known as a **restricted Boltzmann machine** (**RBM**) (Hinton 2002), or a **harmonium** (Smolensky 1986).

An RBM is a special case of a **product of experts** (PoE) (Hinton 1999), which is so-called because we are multiplying together a set of "experts" (here, potential functions on each edge)

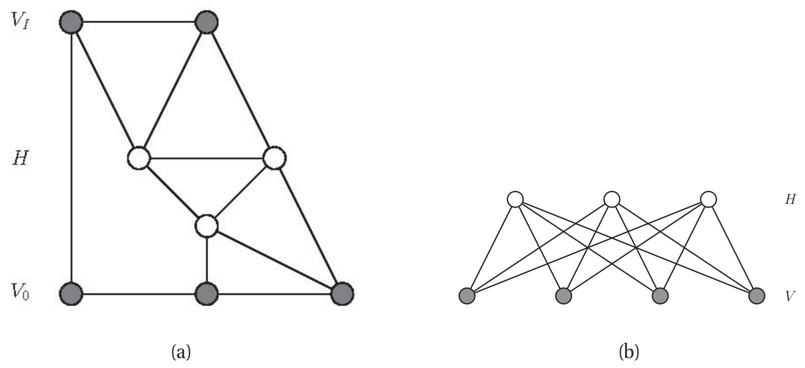


Figure 27.30 (a) A general Boltzmann machine, with an arbitrary graph structure. The shaded (visible) nodes are partitioned into input and output, although the model is actually symmetric and defines a joint density on all the nodes. (b) A restricted Boltzmann machine with a bipartite structure. Note the lack of intra-layer connections.

and then normalizing, whereas in a mixture of experts, we take a convex combination of normalized distributions. The intuitive reason why PoE models might work better than a mixture is that each expert can enforce a constraint (if the expert has a value which is $\gg 1$ or $\ll 1$) or a "don't care" condition (if the expert has value 1). By multiplying these experts together in different ways we can create "sharp" distributions which predict data which satisfies the specified constraints (Hinton and Teh 2001). For example, consider a distributed model of text. A given document might have the topics "government", "mafia" and "playboy". If we "multiply" the predictions of each topic together, the model may give very high probability to the word "Berlusconi"[9] (Salakhutdinov and Hinton 2010). By contrast, adding together experts can only make the distribution broader (see Figure 14.17).

Typically the hidden nodes in an RBM are binary, so $\mathbf{h}$ specifies which constraints are active. It is worth comparing this with the directed models we have discussed. In a mixture model, we have one hidden variable $q \in \{1, \ldots, K\}$. We can represent this using a set of K bits, with the restriction that exactly one bit is on at a time. This is called a **localist encoding**, since only one hidden unit is used to generate the response vector. This is analogous to the hypothetical notion of **grandmother cells** in the brain, that are able to recognize only one kind of object. By contrast, an RBM uses a **distributed encoding**, where many units are involved in generating each output. Models that used vector-valued hidden variables, such as $\boldsymbol{\pi} \in S_K$, as in mPCA/LDA, or $\mathbf{z} \in \mathbb{R}^K$, as in ePCA also use distributed encodings.

The main difference between an RBM and directed two-layer models is that the hidden variables are conditionally independent given the visible variables, so the posterior factorizes:

$$p(\mathbf{h}|\mathbf{v}, \boldsymbol{\theta}) = \prod_k p(h_k|\mathbf{v}, \boldsymbol{\theta}) \tag{27.96}$$

This makes inference much simpler than in a directed model, since we can estimate each h_k

9. Silvio Berlusconi is the current (2011) prime minister of Italy.

Visible	Hidden	Name	Reference
Binary	Binary	Binary RBM	(Hinton 2002)
Gaussian	Binary	Gaussian RBM	(Welling and Sutton 2005)
Categorical	Binary	Categorical RBM	(Salakhutdinov et al. 2007)
Multiple categorical	Binary	Replicated softmax/ undirected LDA	(Salakhutdinov and Hinton 2010)
Gaussian	Gaussian	Undirected PCA	(Marks and Movellan 2001)
Binary	Gaussian	Undirected binary PCA	(Welling and Sutton 2005)

Table 27.2 Summary of different kinds of RBM.

independently and in parallel, as in a feedforward neural network. The disadvantage is that training undirected models is much harder, as we discuss below.

27.7.1 Varieties of RBMs

In this section, we describe various forms of RBMs, by defining different pairwise potential functions. See Table 27.2 for a summary. All of these are special cases of the **exponential family harmonium** (Welling et al. 2004).

27.7.1.1 Binary RBMs

The most common form of RBM has binary hidden nodes and binary visible nodes. The joint distribution then has the following form:

$$p(\mathbf{v}, \mathbf{h}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp(-E(\mathbf{v}, \mathbf{h}; \boldsymbol{\theta})) \tag{27.97}$$

$$E(\mathbf{v}, \mathbf{h}; \boldsymbol{\theta}) \triangleq -\sum_{r=1}^{R}\sum_{k=1}^{K} v_r h_k W_{rk} - \sum_{r=1}^{R} v_r b_r - \sum_{k=1}^{K} h_k c_k \tag{27.98}$$

$$= -(\mathbf{v}^T\mathbf{W}\mathbf{h} + \mathbf{v}^T\mathbf{b} + \mathbf{h}^T\mathbf{c}) \tag{27.99}$$

$$Z(\boldsymbol{\theta}) = \sum_{\mathbf{v}}\sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}; \boldsymbol{\theta})) \tag{27.100}$$

where E is the energy function, $\mathbf{W}$ is a $R \times K$ weight matrix, $\mathbf{b}$ are the visible bias terms, $\mathbf{c}$ are the hidden bias terms, and $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b}, \mathbf{c})$ are all the parameters. For notational simplicity, we will absorb the bias terms into the weight matrix by clamping dummy units $v_0 = 1$ and $h_0 = 1$ and setting $\mathbf{w}_{0,:} = \mathbf{c}$ and $\mathbf{w}_{:,0} = \mathbf{b}$. Note that naively computing $Z(\boldsymbol{\theta})$ takes $O(2^R 2^K)$ time but we can reduce this to $O(\min\{R2^K, K2^R\})$ time (Exercise 27.1).

When using a binary RBM, the posterior can be computed as follows:

$$p(\mathbf{h}|\mathbf{v}, \boldsymbol{\theta}) = \prod_{k=1}^{K} p(h_k|\mathbf{v}, \boldsymbol{\theta}) = \prod_k \text{Ber}(h_k|\text{sigm}(\mathbf{w}_{:,k}^T \mathbf{v})) \tag{27.101}$$

By symmetry, one can show that we can generate data given the hidden variables as follows:

$$p(\mathbf{v}|\mathbf{h}, \boldsymbol{\theta}) = \prod_r p(v_r|\mathbf{h}, \boldsymbol{\theta}) = \prod_r \text{Ber}(v_r|\text{sigm}(\mathbf{w}_{r,:}^T \mathbf{h})) \tag{27.102}$$

We can write this in matrix-vetor notation as follows:

$$\begin{aligned} \mathbb{E}[\mathbf{h}|\mathbf{v}\boldsymbol{\theta}] &= \text{sigm}(\mathbf{W}^T\mathbf{v}) && (27.103) \\ \mathbb{E}[\mathbf{v}|\mathbf{h},\boldsymbol{\theta}] &= \text{sigm}(\mathbf{W}\mathbf{h}) && (27.104) \end{aligned}$$

The weights in $\mathbf{W}$ are called the **generative weights**, since they are used to generate the observations, and the weights in $\mathbf{W}^T$ are called the **recognition weights**, since they are used to recognize the input.

From Equation 27.101, we see that we activate hidden node k in proportion to how much the input vector $\mathbf{v}$ "looks like" the weight vector $\mathbf{w}_{:,k}$ (up to scaling factors). Thus each hidden node captures certain features of the input, as encoded in its weight vector, similar to a feedforward neural network.

27.7.1.2 Categorical RBM

We can extend the binary RBM to categorical visible variables by using a 1-of-C encoding, where C is the number of states for each v_{ir}. We define a new energy function as follows (Salakhutdinov et al. 2007; Salakhutdinov and Hinton 2010):

$$E(\mathbf{v},\mathbf{h};\boldsymbol{\theta}) \triangleq -\sum_{r=1}^{R}\sum_{k=1}^{K}\sum_{c=1}^{C} v_r^c h_k W_{rk}^c - \sum_{r=1}^{R}\sum_{c=1}^{C} v_r^c b_r^c - \sum_{k=1}^{K} h_k c_k \quad (27.105)$$

The full conditionals are given by

$$\begin{aligned} p(v_r|\mathbf{h},\boldsymbol{\theta}) &= \text{Cat}(\mathcal{S}(\{b_r^c + \sum_k h_k W_{rk}^c\}_{c=1}^C)) && (27.106) \\ p(h_k = 1|\mathbf{c},\boldsymbol{\theta}) &= \text{sigm}(c_k + \sum_r \sum_c v_r^c W_{rk}^c) && (27.107) \end{aligned}$$

27.7.1.3 Gaussian RBM

We can generalize the model to handle real-valued data. In particular, a **Gaussian RBM** has the following energy function:

$$E(\mathbf{v},\mathbf{h}|\boldsymbol{\theta}) = -\sum_{r=1}^{R}\sum_{k=1}^{K} W_{rk} h_k v_r - \frac{1}{2}\sum_{r=1}^{R}(v_r - b_r)^2 - \sum_{k=1}^{K} a_k h_k \quad (27.108)$$

The parameters of the model are $\boldsymbol{\theta} = (w_{rk}, a_k, b_r)$. (We have assumed the data is standardized, so we fix the variance to $\sigma^2 = 1$.) Compare this to a Gaussian in information form:

$$\mathcal{N}_c(\mathbf{v}|\boldsymbol{\eta},\boldsymbol{\Lambda}) \propto \exp(\boldsymbol{\eta}^T\mathbf{v} - \frac{1}{2}\mathbf{v}^T\boldsymbol{\Lambda}\mathbf{v}) \quad (27.109)$$

where $\boldsymbol{\eta} = \boldsymbol{\Lambda}\boldsymbol{\mu}$. We see that we have set $\boldsymbol{\Lambda} = \mathbf{I}$, and $\boldsymbol{\eta} = \sum_k h_k \mathbf{w}_{:,k}$. Thus the mean is given by $\boldsymbol{\mu} = \boldsymbol{\Lambda}^{-1}\boldsymbol{\eta} = \sum_k h_k \mathbf{w}_{:,k}$. The full conditionals, which are needed for inference and learning, are given by

$$p(v_r|\mathbf{h},\boldsymbol{\theta}) = \mathcal{N}(v_r|b_r + \sum_k w_{rk}h_k, 1) \tag{27.110}$$

$$p(h_k = 1|\mathbf{v},\boldsymbol{\theta}) = \text{sigm}\left(c_k + \sum_r w_{rk}v_r\right) \tag{27.111}$$

We see that each visible unit has a Gaussian distribution whose mean is a function of the hidden bit vector. More powerful models, which make the (co)variance depend on the hidden state, can also be developed (Ranzato and Hinton 2010).

27.7.1.4 RBMs with Gaussian hidden units

If we use Gaussian latent variables and Gaussian visible variables, we get an undirected version of factor analysis. However, it turns out that it is identical to the standard directed version (Marks and Movellan 2001).

If we use Gaussian latent variables and categorical observed variables, we get an undirected version of categorical PCA (Section 27.2.2). In (Salakhutdinov et al. 2007), this was applied to the Netflix collaborative filtering problem, but was found to be significantly inferior to using binary latent variables, which have more expressive power.

27.7.2 Learning RBMs

In this section, we discuss some ways to compute ML parameter estimates of RBMs, using gradient-based optimizers. It is common to use stochastic gradient descent, since RBMs often have many parameters and therefore need to be trained on very large datasets. In addition, it is standard to use ℓ_2 regularization, a technique that is often called weight decay in this context. This requires a very small change to the objective and gradient, as discussed in Section 8.3.6.

27.7.2.1 Deriving the gradient using $p(\mathbf{h}, \mathbf{v}|\boldsymbol{\theta})$

To compute the gradient, we can modify the equations from Section 19.5.2, which show how to fit a generic latent variable maxent model. In the context of the Boltzmann machine, we have one feature per edge, so the gradient is given by

$$\frac{\partial \ell}{\partial w_{rk}} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}\left[v_r h_k|\mathbf{v}_i,\boldsymbol{\theta}\right] - \mathbb{E}\left[v_r h_k|\boldsymbol{\theta}\right] \tag{27.112}$$

We can write this in matrix-vector form as follows:

$$\nabla_{\mathbf{w}}\, \ell = \mathbb{E}_{p_{\text{emp}}(\cdot|\boldsymbol{\theta})}\left[\mathbf{v}\mathbf{h}^T\right] - \mathbb{E}_{p(\cdot|\boldsymbol{\theta})}\left[\mathbf{v}\mathbf{h}^T\right] \tag{27.113}$$

where $p_{\text{emp}}(\mathbf{v},\mathbf{h}|\boldsymbol{\theta}) \triangleq p(\mathbf{h}|\mathbf{v},\boldsymbol{\theta})p_{\text{emp}}(\mathbf{v})$, and $p_{\text{emp}}(\mathbf{v}) = \frac{1}{N}\sum_{i=1}^{N}\delta_{\mathbf{v}_i}(\mathbf{v})$ is the empirical distribution. (We can derive a similar expression for the bias terms by setting $v_r = 1$ or $h_k = 1$.)

The first term on the gradient, when $\mathbf{v}$ is fixed to a data case, is sometimes called the **clamped phase**, and the second term, when $\mathbf{v}$ is free, is sometimes called the **unclamped**

phase. When the model expectations match the empirical expectations, the two terms cancel out, the gradient becomes zero and learning stops. This algorithm was first proposed in (Ackley et al. 1985). The main problem is efficiently computing the expectations. We discuss some ways to do this below.

27.7.2.2 Deriving the gradient using $p(\mathbf{v}|\boldsymbol{\theta})$

We now present an alternative way to derive Equation 27.112, which also applies to other energy based models. First we marginalize out the hidden variables and write the RBM in the form $p(\mathbf{v}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp(-F(\mathbf{v};\boldsymbol{\theta}))$, where $F(\mathbf{v};\boldsymbol{\theta})$ is the **free energy**:

$$\begin{aligned}
F(\mathbf{v}) &\triangleq \sum_{\mathbf{h}} E(\mathbf{v},\mathbf{h}) = \sum_{\mathbf{h}} \exp\left(\sum_{r=1}^{R}\sum_{k=1}^{K} v_r h_k W_{rk}\right) && (27.114)\\
&= \sum_{\mathbf{h}} \prod_{k=1}^{K} \exp\left(\sum_{r=1}^{R} v_r h_k W_{rk}\right) && (27.115)\\
&= \prod_{k=1}^{K} \sum_{h_r \in \{0,1\}} \exp\left(\sum_{r=1}^{R} v_r h_r W_{rk}\right) && (27.116)\\
&= \prod_{k=1}^{K} \left(1 + \exp(\sum_{r=1}^{R} v_r W_{rk})\right) && (27.117)
\end{aligned}$$

Next we write the (scaled) log-likelihood in the following form:

$$\ell(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \log p(\mathbf{v}_i|\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N} F(\mathbf{v}_i|\boldsymbol{\theta}) - \log Z(\boldsymbol{\theta}) \quad (27.118)$$

Using the fact that $Z(\boldsymbol{\theta}) = \sum_{\mathbf{v}} \exp(-F(\mathbf{v};\boldsymbol{\theta}))$ we have

$$\begin{aligned}
\nabla \ell(\boldsymbol{\theta}) &= -\frac{1}{N}\sum_{i=1}^{N} \nabla F(\mathbf{v}_i) - \frac{\nabla Z}{Z} && (27.119)\\
&= -\frac{1}{N}\sum_{i=1}^{N} \nabla F(\mathbf{v}_i) + \sum_{\mathbf{v}} \nabla F(\mathbf{v}) \frac{\exp(-F(\mathbf{v}))}{Z} && (27.120)\\
&= -\frac{1}{N}\sum_{i=1}^{N} \nabla F(\mathbf{v}_i) + \mathbb{E}\left[\nabla F(\mathbf{v})\right] && (27.121)
\end{aligned}$$

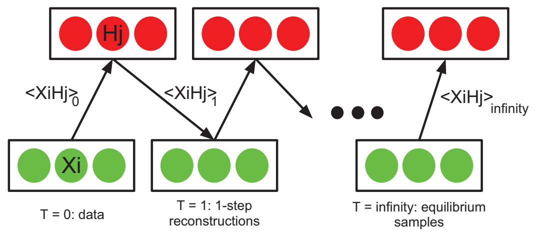
Plugging in the free energy (Equation 27.117), one can show that

$$\frac{\partial}{\partial w_{rk}} F(\mathbf{v}) = -v_r \mathbb{E}\left[h_k|\mathbf{v},\boldsymbol{\theta}\right] = -\mathbb{E}\left[v_r h_k|\mathbf{v},\boldsymbol{\theta}\right] \quad (27.122)$$

Hence

$$\frac{\partial}{\partial w_{rk}} \ell(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}\left[v_r h_k|\mathbf{v},\boldsymbol{\theta}\right] - \mathbb{E}\left[v_r h_k|\boldsymbol{\theta}\right] \quad (27.123)$$

which matches Equation 27.112.



Hj
Xi
<XiHj>0
<XiHj>1
<XiHj>infinity
T = 0: data
T = 1: 1-step reconstructions
T = infinity: equilibrium samples

Figure 27.31 Illustration of Gibbs sampling in an RBM. The visible nodes are initialized at a datavector, then we sample a hidden vector, then another visible vector, etc. Eventually (at "infinity") we will be producing samples from the joint distribution $p(\mathbf{v}, \mathbf{h}|\boldsymbol{\theta})$.

27.7.2.3 Approximating the expectations

We can approximate the expectations needed to evaluate the gradient by performing block Gibbs sampling, using Equations 27.101 and 27.102. In more detail, we can sample from the joint distribution $p(\mathbf{v}, \mathbf{h}|\boldsymbol{\theta})$ as follows: initialize the chain at vv^1 (e.g. by setting $\mathbf{v}^1 = \mathbf{v}_i$ for some data vector), and then sample from $\mathbf{h}^1 \sim p(\mathbf{h}|\mathbf{v}^1)$, then from $\mathbf{v}^2 \sim p(\mathbf{v}|\mathbf{h}^1)$, then from $\mathbf{h}^2 \sim p(\mathbf{h}|\mathbf{v}^2)$, etc. See Figure 27.31 for an illustration. Note, however, that we have to wait until the Markov chain reaches equilibrium (i.e., until it has "burned in") before we can interpret the samples as coming from the joint distribution of interest, and this might take a long time.

A faster alternative is to use mean field, where we make the approximation $\mathbb{E}[v_r h_k] \approx \mathbb{E}[v_r]\,\mathbb{E}[h_k]$. However, since $p(\mathbf{v}, \mathbf{h})$ is typically multimodal, this is usually a very poor approximation, since it will average over different modes (see Section 21.2.2). Furthermore, there is a more subtle reason not to use mean field: since the gradient has the form $\mathbb{E}[v_r h_k|\mathbf{v}] - \mathbb{E}[v_r h_k]$, we see that the negative sign in front means that the method will try to make the variational bound as loose as possible (Salakhutdinov and Hinton 2009). This explains why earlier attempts to use mean field to learn Boltzmann machines (e.g., (Kappen and Rodriguez 1998)) did not work.

27.7.2.4 Contrastive divergence

The problem with using Gibbs sampling to compute the gradient is that it is slow. We now present a faster method known as **contrastive divergence** or **CD** (Hinton 2002). CD was originally derived by approximating an objective function defined as the difference of two KL divergences, rather than trying to maximize the likelihood itself. However, from an algorithmic point of view, it can be thought of as similar to stochastic gradient descent, except it approximates the "unclamped" expectations with "brief" Gibbs sampling where we initialize each Markov chain at the data vectors. That is, we approximate the gradient for one datavector as follows:

$$\nabla_{\mathbf{w}}\, \ell \approx \mathbb{E}\left[\mathbf{v}\mathbf{h}^T|\mathbf{v}_i\right] - \mathbb{E}_q\left[\mathbf{v}\mathbf{h}^T\right] \tag{27.124}$$

where q corresponds to the distribution generated by K up-down Gibbs sweeps, started at $\mathbf{v}_i$, as in Figure 27.31. This is known as CD-K. In more detail, the procedure (for $K = 1$) is as follows:

$$\mathbf{h}_i \sim p(\mathbf{h}|\mathbf{v}_i, \boldsymbol{\theta}) \tag{27.125}$$

$$\mathbf{v}'_i \sim p(\mathbf{v}|\mathbf{h}_i, \boldsymbol{\theta}) \tag{27.126}$$

$$\mathbf{h}'_i \sim p(\mathbf{h}|\mathbf{v}'_i, \boldsymbol{\theta}) \tag{27.127}$$

We then make the approximation

$$\mathbb{E}_q \left[\mathbf{v}\mathbf{h}^T\right] \approx \mathbf{v}_i(\mathbf{h}'_i)^T \tag{27.128}$$

Such samples are sometimes called **fantasy data**. We can think of $\mathbf{v}'_i$ as the model's best attempt at reconstructing $\mathbf{v}_i$ after being coded and then decoded by the model. This is similar to the way we train auto-encoders, which are models which try to “squeeze” the data through a restricted parametric “bottleneck” (see Section 28.3.2).

In practice, it is common to use $\mathbb{E}[\mathbf{h}|\mathbf{v}'_i]$ instead of a sampled value $\mathbf{h}'_i$ in the final upwards pass, since this reduces the variance. However, it is not valid to use $\mathbb{E}[\mathbf{h}|\mathbf{v}_i]$ instead of sampling $\mathbf{h}_i \sim p(\mathbf{h}|\mathbf{v}_i)$ in the earlier upwards passes, because then each hidden unit would be able to pass more than 1 bit of information, so it would not act as much of a bottleneck.

The whole procedure is summarized in Algorithm 3. (Note that we follow the positive gradient since we are maximizing likelihood.) Various tricks can be used to speed this algorithm up, such as using a momentum term (Section 8.3.2), using mini-batches, averaging the updates, etc. Such details can be found in (Hinton 2010; Swersky et al. 2010).

Algorithm 27.3: CD-1 training for an RBM with binary hidden and visible units

```
1  Initialize weights $\mathbf{W} \in \mathbb{R}^{R \times K}$ randomly;
2  $t := 0$;
3  for each epoch do
4      $t := t + 1$ ;
5      for each minibatch of size $B$ do
6          Set minibatch gradient to zero, $\mathbf{g} := \mathbf{0}$ ;
7          for each case $\mathbf{v}_i$ in the minibatch do
8              Compute $\boldsymbol{\mu}_i = \mathbb{E}[\mathbf{h}|\mathbf{v}_i, \mathbf{W}]$;
9              Sample $\mathbf{h}_i \sim p(\mathbf{h}|\mathbf{v}_i, \mathbf{W})$;
10             Sample $\mathbf{v}'_i \sim p(\mathbf{v}|\mathbf{h}_i, \mathbf{W})$;
11             Compute $\boldsymbol{\mu}'_i = \mathbb{E}[\mathbf{h}|\mathbf{v}'_i, \mathbf{W}]$;
12             Compute gradient $\nabla_{\mathbf{W}} = (\mathbf{v}_i)(\boldsymbol{\mu}_i)^T - (\mathbf{v}'_i)(\boldsymbol{\mu}'_i)^T$ ;
13             Accumulate $\mathbf{g} := \mathbf{g} + \nabla_{\mathbf{W}}$;
14         Update parameters $\mathbf{W} := \mathbf{W} + (\alpha_t/B)\mathbf{g}$
```

27.7.2.5 Persistent CD

In Section 19.5.5, we presented a technique called stochastic maximum likelihood (SML) for fitting maxent models. This avoids the need to run MCMC to convergence at each iteration, by exploiting the fact that the parameters are changing slowly, so the Markov chains will not be pushed too far from equilibrium after each update (Younes 1989). In other words, there are two dynamical processes running at different time scales: the states change quickly, and the parameters change slowly. This algorithm was independently rediscovered in (Tieleman 2008), who called it **persistent CD**. See Algorithm 3 for the pseudocode.

PCD often works better than CD (see e.g., (Tieleman 2008; Marlin et al. 2010; Swersky et al. 2010)), although CD can be faster in the early stages of learning.

Algorithm 27.4: Persistent CD for training an RBM with binary hidden and visible units

```
1  Initialize weights $\mathbf{W} \in \mathbb{R}^{D \times L}$ randomly;
2  Initialize chains $(\mathbf{v}_s, \mathbf{h}_s)_{s=1}^S$ randomly ;
3  for $t = 1, 2, \ldots$ do
4      // Mean field updates ;
5      for each case $i = 1 : N$ do
6          $\mu_{ik} = \text{sigm}(\mathbf{v}_i^T \mathbf{w}_{:,k})$
7      // MCMC updates ;
8      for each sample $s = 1 : S$ do
9          Generate $(\mathbf{v}_s, \mathbf{h}_s)$ by brief Gibbs sampling from old $(\mathbf{v}_s, \mathbf{h}_s)$
10     // Parameter updates ;
11     $\mathbf{g} = \frac{1}{N} \sum_{i=1}^N \mathbf{v}_i (\boldsymbol{\mu}_i)^T - \frac{1}{S} \sum_{s=1}^S \mathbf{v}_s (\mathbf{h}_s)^T$ ;
12     $\mathbf{W} := \mathbf{W} + \alpha_t \mathbf{g}$;
13     Decrease $\alpha_t$
```

27.7.3 Applications of RBMs

The main application of RBMs is as a building block for deep generative models, which we discuss in Section 28.2. But they can also be used as substitutes for directed two-layer models. They are particularly useful in cases where inference of the hidden states at test time must be fast. We give some examples below.

27.7.3.1 Language modeling and document retrieval

We can use a categorical RBM to define a generative model for bag-of-words, as an alternative to LDA. One subtlety is that the partition function in an undirected models depends on how big the graph is, and therefore on how long the document is. A solution to this was proposed in (Salakhutdinov and Hinton 2010): use a categorical RBM with tied weights, but multiply the hidden activation bias terms c_k by the document length L to compensate form the fact that the observed word-count vector $\mathbf{v}$ is larger in magnitude:

$$E(\mathbf{v}, \mathbf{h}; \boldsymbol{\theta}) \triangleq -\sum_{k=1}^K \sum_{c=1}^C v^c h_k W_k^c - \sum_{c=1}^C v^c b_r^c - L \sum_{k=1}^K h_k c_k \tag{27.129}$$

Data set	Number of docs		K	$\bar{D}$	St. Dev.	Avg. Test perplexity per word (in nats)			
	Train	Test				LDA-50	LDA-200	R. Soft-50	Unigram
NIPS	1,690	50	13,649	98.0	245.3	3576	3391	3405	4385
20-news	11,314	7,531	2,000	51.8	70.8	1091	1058	953	1335
Reuters	794,414	10,000	10,000	94.6	69.3	1437	1142	988	2208

Figure 27.32 Comparison of RBM (replicated softmax) and LDA on three corpora. K is the number of words in the vocabulary, $\overline{D}$ is the average document length, and St. Dev. is the standard deviation of the document length. Source: (Salakhutdinov and Hinton 2010) .

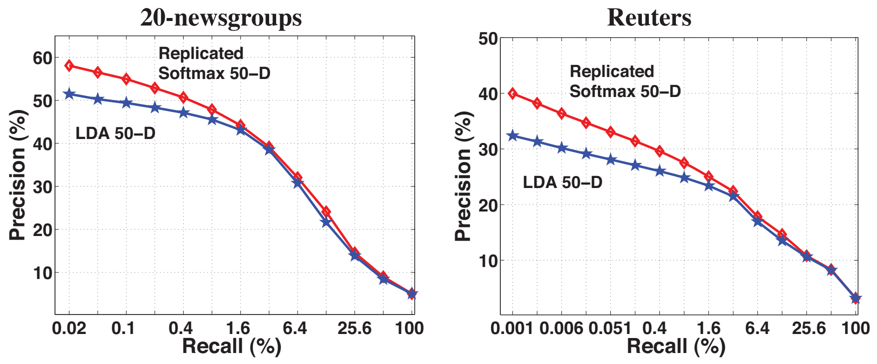


Figure 27.33 Precision-recall curves for RBM (replicated softmax) and LDA on two corpora. From Figure 3 of (Salakhutdinov and Hinton 2010). Used with kind permission of Ruslan Salakhutdinov.

where $v^c = \sum_{l=1}^{L} \mathbb{I}(y_{il} = c)$. This is like having a single multinomial node (so we have dropped the r subscript) with C states, where C is the number of words in the vocabulary. This is called the **replicated softmax model** (Salakhutdinov and Hinton 2010), and is an undirected alternative to mPCA/ LDA.

We can compare the modeling power of RBMs vs LDA by measuring the perplexity on a test set. This can be approximated using annealing importance sampling (Section 24.6.2). The results are shown in Figure 27.32. We see that the LDA is significantly better than a unigram model, but that an RBM is significantly better than LDA.

Another advantage of the LDA is that inference is fast and exact: just a single matrix-vector multiply followed by a sigmoid nonlinearity, as in Equation 27.107. In addition to being faster, the RBM is more accurate. This is illustrated in Figure 27.33, which shows precision-recall curves for RBMs and LDA on two different corpora. These curves were generated as follows: a query document from the test set is taken, its similarity to all the training documents is computed, where the similarity is defined as the cosine of the angle between the two topic vectors, and then the top M documents are returned for varying M. A retrieved document is considered relevant if it has the same class label as that of the query's (this is the only place where labels are used).

27.7.3.2 RBMs for collaborative filtering

RBMs have been applied to the Netflix collaborative filtering competition (Salakhutdinov et al. 2007). In fact, an RBM with binary hidden nodes and categorical visible nodes can slightly outperform SVD. By combining the two methods, performance can be further improved. (The winning entry in the challenge was an ensemble of many different types of model (Koren 2009a).)

Exercises

Exercise 27.1 Partition function for an RBM

Show how to compute $Z(\boldsymbol{\theta})$ for an RBM with K binary hidden nodes and R binary observed nodes in $O(R2^K)$ time, assuming $K < R$.

28 *Deep learning*

28.1 Introduction

Many of the models we have looked at in this book have a simple two-layer architecture of the form $\mathbf{z} \rightarrow \mathbf{y}$ for unsupervised latent variable models, or $\mathbf{x} \rightarrow \mathbf{y}$ for supervised models. However, when we look at the brain, we seem many levels of processing. It is believed that each level is learning features or representations at increasing levels of abstraction. For example, the **standard model** of the visual cortex (Hubel and Wiesel 1962; Serre et al. 2005; Ranzato et al. 2007) suggests that (roughly speaking) the brain first extracts edges, then patches, then surfaces, then objects, etc. (See e.g., (Palmer 1999; Kandel et al. 2000) for more information about how the brain might perform vision.)

This observation has inspired a recent trend in machine learning known as **deep learning** (see e.g., (Bengio 2009), `deeplearning.net`, and the references therein), which attempts to replicate this kind of architecture in a computer. (Note the idea can be applied to non-vision problems as well, such as speech and language.)

In this chapter, we give a brief overview of this new field. However, we caution the reader that the topic of deep learning is currently evolving very quickly, so the material in this chapter may soon be outdated.

28.2 Deep generative models

Deep models often have millions of parameters. Acquiring enough labeled data to train such models is diffcult, despite crowd sourcing sites such as Mechanical Turk. In simple settings, such as hand-written character recognition, it is possible to generate lots of labeled data by making modified copies of a small manually labeled training set (see e.g., Figure 16.13), but it seems unlikely that this approach will scale to complex scenes.[1]

To overcome the problem of needing labeled training data, we will focus on unsupervised learning. The most natural way to perform this is to use generative models. In this section, we discuss three different kinds of deep generative models: directed, undirected, and mixed.

1. There have been some attempts to use computer graphics and video games to generate realistic-looking images of complex scenes, and then to use this as training data for computer vision systems. However, often graphics programs cut corners in order to make perceptually appealing images which are not reflective of the natural statistics of real-world images.

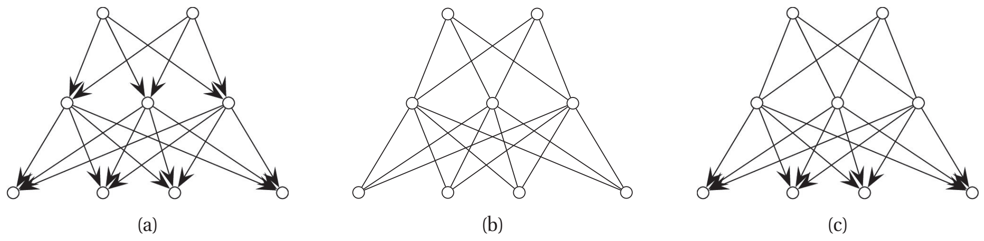


Figure 28.1 Some deep multi-layer graphical models. Observed variables are at the bottom. (a) A directed model. (b) An undirected model (deep Boltzmann machine). (c) A mixed directed-undirected model (deep belief net).

28.2.1 Deep directed networks

Perhaps the most natural way to build a deep generative model is to construct a deep directed graphical model, as shown in Figure 28.1(a). The bottom level contains the observed pixels (or whatever the data is), and the remaining layers are hidden. We have assumed just 3 layers for notational simplicity. The number and size of layers is usually chosen by hand, although one can also use non-parametric Bayesian methods (Adams et al. 2010) or boosting (Chen et al. 2010) to infer the model structure.

We shall call models of this form **deep directed networks** or DDNs. If all the nodes are binary, and all CPDs are logistic functions, this is called a **sigmoid belief net** (Neal 1992). In this case, the model defines the following joint distribution:

$$p(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{v}|\boldsymbol{\theta}) = \prod_i \text{Ber}(v_i|\text{sigm}(\mathbf{h}_1^T \mathbf{w}_{0i})) \prod_j \text{Ber}(h_{1j}|\text{sigm}(\mathbf{h}_2^T \mathbf{w}_{1j})) \tag{28.1}$$

$$\prod_k \text{Ber}(h_{2k}|\text{sigm}(\mathbf{h}_3^T \mathbf{w}_{2k})) \prod_l \text{Ber}(h_{3l}|w_{3l}) \tag{28.2}$$

Unfortunately, inference in directed models such as these is intractable because the posterior on the hidden nodes is correlated due to explaining away. One can use fast mean field approximations (Jaakkola and Jordan 1996a; Saul and Jordan 2000), but these may not be very accurate, since they approximate the correlated posterior with a factorial posterior. One can also use MCMC inference (Neal 1992; Adams et al. 2010), but this can be quite slow because the variables are highly correlated. Slow inference also results in slow learning.

28.2.2 Deep Boltzmann machines

A natural alternative to a directed model is to construct a deep undirected model. For example, we can stack a series of RBMs on top of each other, as shown in Figure 28.1(b). This is known as a **deep Boltzmann machine** or **DBM** (Salakhutdinov and Hinton 2009). If we have 3 hidden layers, the model is defined as follows:

$$p(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{v}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left(\sum_{ij} v_i h_{1j} W_{1ij} + \sum_{jk} h_{1j} h_{2j} W_{2jk} + \sum_{kl} h_{2k} h_{3l} W_{3kl} \right) \tag{28.3}$$

where we are ignoring constant offset or bias terms.

The main advantage over the directed model is that one can perform efficient block (layer-wise) Gibbs sampling, or block mean field, since all the nodes in each layer are conditionally independent of each other given the layers above and below (Salakhutdinov and Larochelle 2010). The main disadvantage is that training undirected models is more difficult, because of the partition function. However, below we will see a greedy layer-wise strategy for learning deep undirected models.

28.2.3 Deep belief networks

An interesting compromise is to use a model that is partially directed and partially undirected. In particular, suppose we construct a layered model which has directed arrows, except at the top, where there is an undirected bipartite graph, as shown in Figure 28.1(c). This model is known as a **deep belief network** (Hinton et al. 2006) or **DBN**.[2] If we have 3 hidden layers, the model is defined as follows:

$$p(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{v}|\boldsymbol{\theta}) = \prod_i \text{Ber}(v_i|\text{sigm}(\mathbf{h}_1^T \mathbf{w}_{1i}) \prod_j \text{Ber}(h_{1j}|\text{sigm}(\mathbf{h}_2^T \mathbf{w}_{2j}) \quad (28.4)$$

$$\frac{1}{Z(\boldsymbol{\theta})} \exp\left(\sum_{kl} h_{2k} h_{3l} W_{3kl}\right) \quad (28.5)$$

Essentially the top two layers act as an associative memory, and the remaining layers then generate the output.

The advantage of this peculiar architecture is that we can infer the hidden states in a fast, bottom-up fashion. To see why, suppose we only have two hidden layers, and that $\mathbf{W}_2 = \mathbf{W}_1^T$, so the second level weights are tied to the first level weights (see Figure 28.2(a)). This defines a model of the form $p(\mathbf{h}_1, \mathbf{h}_2, \mathbf{v}|\mathbf{W}_1)$. One can show that the distribution $p(\mathbf{h}_1, \mathbf{v}|\mathbf{W}_1) = \sum_{\mathbf{h}_2} p(\mathbf{h}_1, \mathbf{h}_2, \mathbf{v}|\mathbf{W}_1)$ has the form $p(\mathbf{h}_1, \mathbf{v}|\mathbf{W}_1) = \frac{1}{Z(\mathbf{W}_1)} \exp(\mathbf{v}^T \mathbf{W}_1 \mathbf{h}_1)$, which is equivalent to an RBM. Since the DBN is equivalent to the RBM as far as $p(\mathbf{h}_1, \mathbf{v}|\mathbf{W}_1)$ is concerned, we can infer the posterior $p(\mathbf{h}_1|\mathbf{v}, \mathbf{W}_1)$ in the DBN exactly as in the RBM. This posterior is exact, even though it is fully factorized.

Now the only way to get a factored posterior is if the prior $p(\mathbf{h}_1|\mathbf{W}_1)$ is a **complementary prior**. This is a prior which, when multiplied by the likelihood $p(\mathbf{v}|\mathbf{h}_1)$, results in a perfectly factored posterior. Thus we see that the top level RBM in a DBN acts as a complementary prior for the bottom level directed sigmoidal likelihood function.

If we have multiple hidden levels, and/or if the weights are not tied, the correspondence between the DBN and the RBM does not hold exactly any more, but we can still use the factored inference rule as a form of approximate bottom-up inference. Below we show that this is a valid variational lower bound. This bound also suggests a layer-wise training strategy, that we will explain in more detail later. Note, however, that top-down inference in a DBN is not tractable, so DBNs are usually only used in a feedforward manner.

2. Unforuntately the acronym "DBN" also stands for "dynamic Bayes net" (Section 17.6.7). Geoff Hinton, who invented deep belief networks, has suggested the acronyms **DeeBNs** and **DyBNs** for these two different meanings. However, this terminology is non-standard.

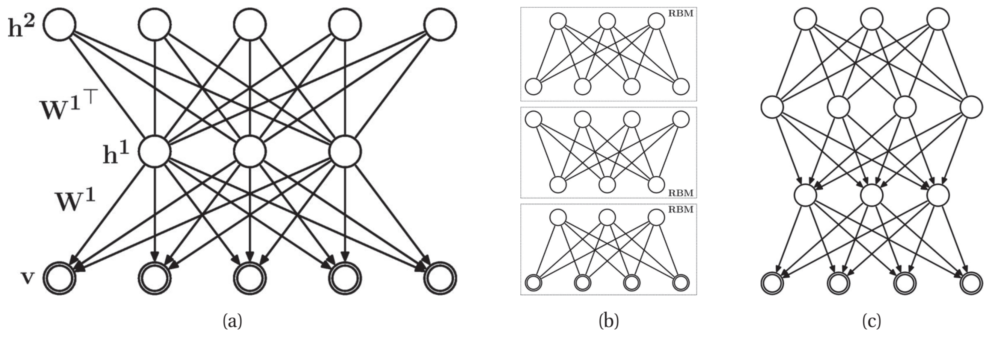


Figure 28.2 (a) A DBN with two hidden layers and tied weights that is equivalent to an RBM. Source: Figure 2.2 of (Salakhutdinov 2009). (b) A stack of RBMs trained greedily. (c) The corresponding DBN. Source: Figure 2.3 of (Salakhutdinov 2009). Used with kind permission of Ruslan Salakhutdinov.

28.2.4 Greedy layer-wise learning of DBNs

The equivalence between DBNs and RBMs suggests the following strategy for learning a DBN.

- Fit an RBM to learn $\mathbf{W}_1$ using methods described in Section 27.7.2.
- Unroll the RBM into a DBN with 2 hidden layers, as in Figure 28.2(a). Now "freeze" the directed weights $\mathbf{W}_1$ and let $\mathbf{W}_2$ be "untied" so it is no longer forced to be equal to $\mathbf{W}_1^T$. We will now learn a better prior for $p(\mathbf{h}_1|\mathbf{W}_2)$ by fitting a second RBM. The input data to this new RBM is the activation of the hidden units $\mathbb{E}[\mathbf{h}_1|\mathbf{v}, \mathbf{W}_1]$ which can be computed using a factorial approximation.
- Continue to add more hidden layers until some stopping criterion is satisified, e.g., you run out of time or memory, or you start to overfit the validation set. Construct the DBN from these RBMs, as illustrated in Figure 28.2(c).

One can show (Hinton et al. 2006) that this procedure always increases a lower bound the observed data likelihood. Of course this procedure might result in overfitting, but that is a different matter.

In practice, we want to be able to use any number of hidden units in each level. This means we will not be able to initialize the weights so that $\mathbf{W}_\ell = \mathbf{W}_{\ell-1}^T$. This voids the theoretical guarantee. Nevertheless the method works well in practice, as we will see. The method can also be extended to train DBMs in a greedy way (Salakhutdinov and Larochelle 2010).

After using the greedy layer-wise training strategy, it is standard to "fine tune" the weights, using a technique called **backfitting**. This works as follows. Perform an upwards sampling pass to the top. Then perform brief Gibbs sampling in the top level RBM, and perform a CD update of the RBM parameters. Finally, perform a downwards ancestral sampling pass (which is an approximate sample from the posterior), and update the logistic CPD parameters using a small gradient step. This is called the **up-down** procedure (Hinton et al. 2006). Unfortunately this procedure is very slow.

28.3 Deep neural networks

Given that DBNs are often only used in a feed-forward, or bottom-up, mode, they are effectively acting like neural networks. In view of this, it is natural to dispense with the generative story and try to fit deep neural networks directly, as we discuss below. The resulting training methods are often simpler to implement, and can be faster.

Note, however, that performance with deep neural nets is sometimes not as good as with probabilistic models (Bengio et al. 2007). One reason for this is that probabilistic models support top-down inference as well as bottom-up inference. (DBNs do not support efficient top-down inference, but DBMs do, and this has been shown to help (Salakhutdinov and Larochelle 2010).) Top-down inference is useful when there is a lot of ambiguity about the correct interpretation of the signal.

It is interesting to note that in the mammalian visual cortex, there are many more feedback connections than there are feedforward connections (see e.g., (Palmer 1999; Kandel et al. 2000)). The role of these feedback connections is not precisely understood, but they presumably provide contextual prior information (e.g., coming from the previous "frame" or retinal glance) which can be used to disambiguate the current bottom-up signals (Lee and Mumford 2003).

Of course, we can simulate the effect of top-down inference using a neural network. However the models we discuss below do not do this.

28.3.1 Deep multi-layer perceptrons

Many decision problems can be reduced to classification, e.g., predict which object (if any) is present in an image patch, or predict which phoneme is present in a given acoustic feature vector. We can solve such problems by creating a deep feedforward neural network or multi-layer perceptron (MLP), as in Section 16.5, and then fitting the parameters using gradient descent (aka back-propagation).

Unfortunately, this method does not work very well. One problem is that the gradient becomes weaker the further we move away from the data; this is known as the "**vanishing gradient**" problem (Bengio and Frasconi 1995). A related problem is that there can be large plateaus in the error surface, which cause simple first-order gadient-based methods to get stuck (Glorot and Bengio 2010).

Consequently early attempts to learn deep neural networks proved unsuccesful. Recently there has been some progress, due to the adoption of GPUs (Ciresan et al. 2010) and second-order optimization algorithms (Martens 2010). Nevertheless, such models remain difficult to train.

Below we discuss a way to initialize the parameters using unsupervised learning; this is called **generative pre-training**. The advantage of performing unsupervised learning first is that the model is forced to model a high-dimensional response, namely the input feature vector, rather than just predicting a scalar response. This acts like a data-induced regularizer, and helps backpropagation find local minima with good generalization properties (Erhan et al. 2010; Glorot and Bengio 2010).

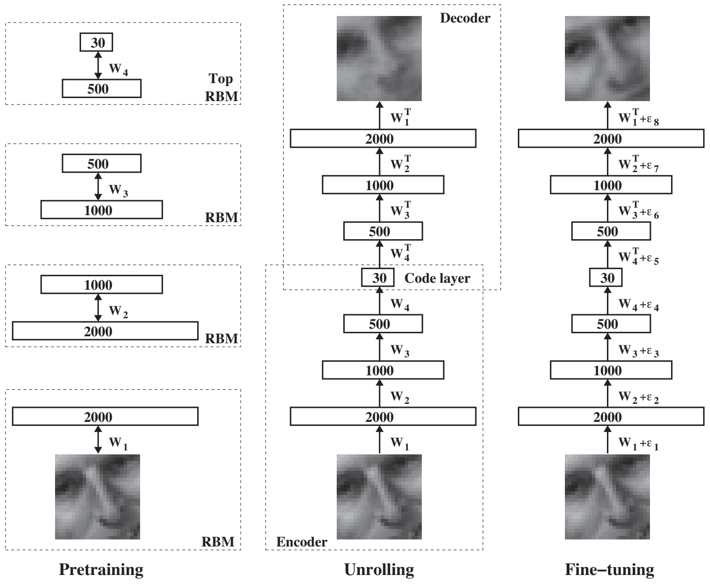


Figure 28.3 Training a deep autoencoder. (a) First we greedily train some RBMs. (b) Then we construct the auto-encoder by replicating the weights. (c) Finally we fine-tune the weights using back-propagation. From Figure 1 of (Hinton and Salakhutdinov 2006). Used with kind permission of Ruslan Salakhutdinov.

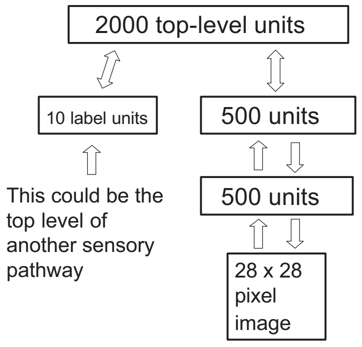
28.3.2 Deep auto-encoders

An **auto-encoder** is a kind of unsupervised neural network that is used for dimensionality reduction and feature discovery. More precisely, an auto-encoder is a feedforward neural network that is trained to predict the input itself. To prevent the system from learning the trivial identity mapping, the hidden layer in the middle is usually constrained to be a narrow **bottleneck**. The system can minimize the reconstruction error by ensuring the hidden units capture the most relevant aspects of the data.

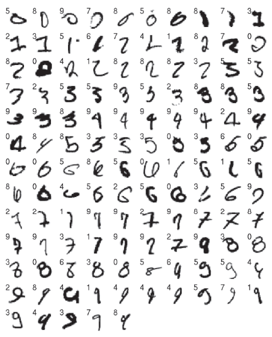
Suppose the system has one hidden layer, so the model has the form $\mathbf{v} \to \mathbf{h} \to \mathbf{v}$. Further, suppose all the functions are linear. In this case, one can show that the weights to the K hidden units will span the same subspace as the first K principal components of the data (Karhunen and Joutsensalo 1995; Japkowicz et al. 2000). In other words, linear auto-encoders are equivalent to PCA. However, by using nonlinear activation functions, one can discover nonlinear representations of the data.

More powerful representations can be learned by using **deep auto-encoders**. Unfortunately training such models using back-propagation does not work well, because the gradient signal becomes too small as it passes back through multiple layers, and the learning algorithm often gets stuck in poor local minima.

One solution to this problem is to greedily train a series of RBMs and to use these to initialize an auto-encoder, as illustrated in Figure 28.3. The whole system can then be fine-tuned using backprop in the usual fashion. This approach, first suggested in (Hinton and Salakhutdinov



(a)



(b)

Figure 28.4 (a) A DBN architecture for classifying MNIST digits. Source: Figure 1 of (Hinton et al. 2006). Used with kind permission of Geoff Hinton. (b) These are the 125 errors made by the DBN on the 10,000 test cases of MNIST. Above each image is the estimated label. Source: Figure 6 of (Hinton et al. 2006). Used with kind permission of Geoff Hinton. Compare to Figure 16.15.

2006), works much better than trying to fit the deep auto-encoder directly starting with random weights.

28.3.3 Stacked denoising auto-encoders

A standard way to train an auto-encoder is to ensure that the hidden layer is narrower than the visible layer. This prevents the model from learning the identity function. But there are other ways to prevent this trivial solution, which allow for the use of an over-complete representation. One approach is to impose sparsity constraints on the activation of the hidden units (Ranzato et al. 2006). Another approach is to add noise to the inputs; this is called a **denoising auto-encoder** (Vincent et al. 2010). For example, we can corrupt some of the inputs, for example by setting them to zero, so the model has to learn to predict the missing entries. This can be shown to be equivalent to a certain approximate form of maximum likelihood training (known as score matching) applied to an RBM (Vincent 2011).

Of course, we can stack these models on top of each other to learn a deep stacked denoising auto-encoder, which can be discriminatively fine-tuned just like a feedforward neural network, if desired.

28.4 Applications of deep networks

In this section, we mention a few applications of the models we have been discussing.

28.4.1 Handwritten digit classification using DBNs

Figure 28.4(a) shows a DBN (from (Hinton et al. 2006)) consisting of 3 hidden layers. The visible layer corresponds to binary images of handwritten digits from the MNIST data set. In addition, the top RBM is connected to a softmax layer with 10 units, representing the class label.

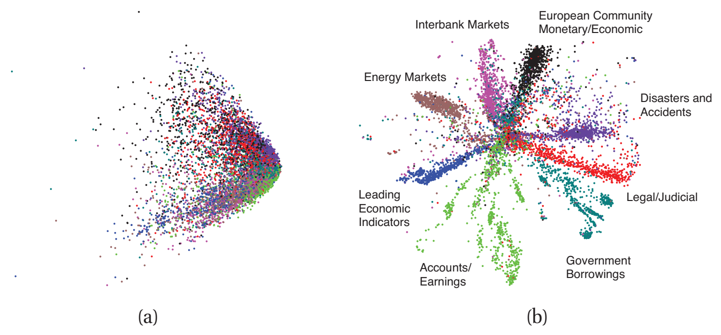


Figure 28.5 2d visualization of some bag of words data from the Reuters RCV1-v2 corpus. (a) Results of using LSA. (b) results of using a deep auto-encoder. Source: Figure 4 of (Hinton and Salakhutdinov 2006). Used with kind permission of Ruslan Salakhutdinov.

The first 2 hidden layers were trained in a greedy unsupervised fashion from 50,000 MNIST digits, using 30 epochs (passes over the data) and stochastic gradient descent, with the CD heuristic. This process took "a few hours per layer" (Hinton et al. 2006, p1540). Then the top layer was trained using as input the activations of the lower hidden layer, as well as the class labels. The corresponding generative model had a test error of about 2.5%. The network weights were then carefully fine-tuned on all 60,000 training images using the up-down procedure. This process took "about a week" (Hinton et al. 2006, p1540). The model can be used to classify by performing a deterministic bottom-up pass, and then computing the free energy for the top-level RBM for each possible class label. The final error on the test set was about 1.25%. The misclassified examples are shown in Figure 28.4(b).

This was the best error rate of any method on the permutation-invariant version of MNIST at that time. (By permutation-invariant, we mean a method that does not exploit the fact that the input is an image. Generic methods work just as well on permuted versions of the input (see Figure 1.5), and can therefore be applied to other kinds of datasets.) The only other method that comes close is an SVM with a degree 9 polynomial kernel, which has achieved an error rate of 1.4% (Decoste and Schoelkopf 2002). By way of comparison, 1-nearest neighbor (using all 60,000 examples) achieves 3.1% (see `mnist1NNdemo`). This is not as good, although 1-NN is much simpler.[3]

28.4.2 Data visualization and feature discovery using deep auto-encoders

Deep autoencoders can learn informative features from raw data. Such features are often used as input to standard supervised learning methods.

To illustrate this, consider fitting a deep auto-encoder with a 2d hidden bottleneck to some

3. One can get much improved performance on this task by exploiting the fact that the input is an image. One way to do this is to create distorted versions of the input, adding small shifts and translations (see Figure 16.13 for some examples). Applying this trick reduced the SVM error rate to 0.56%. Similar error rates can be achieved using convolutional neural networks (Section 16.5.1) trained on distorted images ((Simard et al. 2003) got 0.4%). However, the point of DBNs is that they offer a way to learn such prior knowledge, without it having to be hand-crafted.

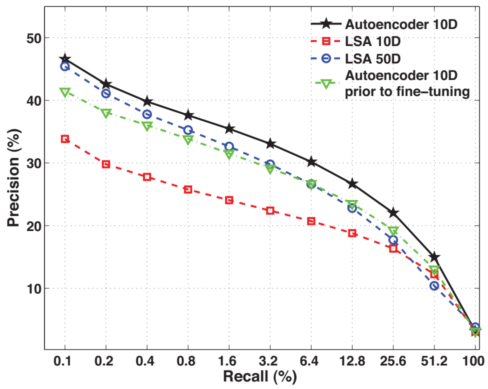


Figure 28.6 Precision-recall curves for document retrieval in the Reuters RCV1-v2 corpus. Source: Figure 3.9 of (Salakhutdinov 2009). Used with kind permission of Ruslan Salakhutdinov.

text data. The results are shown in Figure 28.5. On the left we show the 2d embedding produced by LSA (Section 27.2.2), and on the right, the 2d embedding produced by the auto-encoder. It is clear that the low-dimensional representation created by the auto-encoder has captured a lot of the meaning of the documents, even though class labels were not used.[4]

Note that various other ways of learning low-dimensional continuous embeddings of words have been proposed. See e.g., (Turian et al. 2010) for details.

28.4.3 Information retrieval using deep auto-encoders (semantic hashing)

In view of the sucess of RBMs for information retrieval discussed in Section 27.7.3.1, it is natural to wonder if deep models can do even better. In fact they can, as is shown in Figure 28.6.

More interestingly, we can use a binary low-dimensional representation in the middle layer of the deep auto-encoder, rather than a continuous representation as we used above. This enables very fast retrieval of related documents. For example, if we use a 20-bit code, we can precompute the binary representation for all the documents, and then create a hash-table mapping codewords to documents. This approach is known as **semantic hashing**, since the binary representation of semantically similar documents will be close in Hamming distance.

For the 402,207 test documents in Reuters RCV1-v2, this results in about 0.4 documents per entry in the table. At test time, we compute the codeword for the query, and then simply retrieve the relevant documents in *constant time* by looking up the contents of the relevant address in memory. To find other other related documents, we can compute all the codewords within a

4. Some details. Salakhutdinov and Hinton used the Reuters RCV1-v2 data set, which consists of 804,414 newswire articles, manually classified into 103 topics. They represent each document by counting how many times each of the top 2000 most frequent words occurs. They trained a deep auto-encoder with 2000-500-250-125-2 layers on half of the data. The 2000 visible units use a replicated softmax distribution, the 2 hidden units in the middle layer have a Gaussian distribution, and the remaining units have the usual Bernoulli-logistic distribution. When fine tuning the auto-encoder, a cross-entropy loss function (equivalent to maximum likelihood under a multinoulli distribution) was used. See (Hinton and Salakhutdinov 2006) for further details.

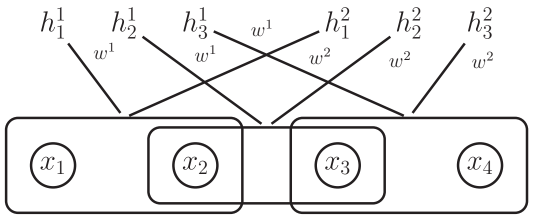


Figure 28.7 A small 1d convolutional RBM with two groups of hidden units, each associated with a filter of size 2. h_1^1 and h_1^2 are two different "views" of the data in the first window, (x_1, x_2). The first view is computed using the filter $\mathbf{w}^1$, the second view using filter $\mathbf{w}^2$. Similarly, h_2^1 and h_2^2 are the views of the data in the second window, (x_2, x_3), computed using $\mathbf{w}^1$ and $\mathbf{w}^2$ respectively.

Hamming distance of, say, 4. This results in retrieving about $6196 \times 0.4 \approx 2500$ documents[5]. The key point is that the total time is independent of the size of the corpus.

Of course, there are other techniques for fast document retrieval, such as inverted indices. These rely on the fact that individual words are quite informative, so we can simply intersect all the documents that contain each word. However, when performing image retrieval, it is clear that we do not want to work at the pixel level. Recently (Krizhevsky and Hinton 2010) showed that a deep autoencoder could learn a good semantic hashing function that outperformed previous techniques (Torralba et al. 2008; Weiss et al. 2008) on the 80 million tiny images dataset. It is hard to apply inverted indexing techniques to real-valued data (although one could imagine vector quantizing image patches).

28.4.4 Learning audio features using 1d convolutional DBNs

To apply DBNs to time series of unbounded length, it is necessary to use some form of parameter tying. One way to do this is to use **convolutional DBNs** (Lee et al. 2009; Desjardins and Bengio 2008), which use convolutional RBMs as their basic unit. These models are a generative version of convolutional neural nets discussed in Section 16.5.1. The basic idea is illustrated in Figure 28.7. The hidden activation vector for each group is computed by convolving the input vector with that group's filter (weight vector or matrix). In other words, each node within a hidden group is a weighted combination of a subset of the inputs. We compute the activaton of all the hidden nodes by "sliding" this weight vector over the input. This allows us to model **translation invariance**, since we use the same weights no matter where in the input vector the pattern occurs.[6] Each group has its own filter, corresponding to its own pattern detector.

5. Note that $6196 = \sum_{k=0}^{4} \binom{20}{k}$ is the number of bit vectors that are up to a Hamming distance of 4 away.

6. It is often said that the goal of deep learnng is to discover **invariant features**, e.g., a representation of an object that does not change even as nuisance variables, such as the lighting, do change. However, sometimes these so-called "nuisance variables" may be the variables of interest. For example if the task is to determine if a photograph was taken in the morning or the evening, then lighting is one of the more salient features, and object identity may be less relevant. As always, one task's "signal" is another task's "noise", so it unwise to "throw away" apparently irrelevant information

More formally, for binary 1d signals, we can define the full conditionals in a convolutional RBM as follows (Lee et al. 2009):

$$p(h_t^k = 1|\mathbf{v}) = \text{sigm}((\mathbf{w}^k \otimes \mathbf{v})_t + b_t) \tag{28.6}$$

$$p(v_s = 1|\mathbf{h}) = \text{sigm}(\sum_k (\mathbf{w}^k \otimes \mathbf{h}^k)_s + c_s) \tag{28.7}$$

where $\mathbf{w}^k$ is the weight vector for group k, b_t and c_s are bias terms, and $\mathbf{a} \otimes \mathbf{b}$ represents the convolution of vectors $\mathbf{a}$ and $\mathbf{b}$.

It is common to add a **max pooling** layer as well as a convolutional layer, which computes a local maximum over the filtered response. This allows for a small amount of translation invariance. It also reduces the size of the higher levels, which speeds up computation considerably. Defining this for a neural network is simple, but defining this in a way which allows for information flow backwards as well as forwards is a bit more involved. The basic idea is similar to a noisy-OR CPD (Section 10.2.3), where we define a probabilistic relationship between the max node and the parts it is maxing over. See (Lee et al. 2009) for details. Note, however, that the top-down generative process will be difficult, since the max pooling operation throws away so much information.

(Lee et al. 2009) applies 1d convolutional DBNs of depth 2 to auditory data. When the input consists of speech signals, the method recovers a representation that is similar to phonemes. When applied to music classification and speaker identification, their method outperforms techniques using standard features such as MFCC. (All features were fed into the same discriminative classifier.)

In (Seide et al. 2011), a deep neural net was used in place of a GMM inside a conventional HMM. The use of DNNs significantly improved performance on conversational speech recognition. In an interview, the tech lead of this project said "historically, there have been very few individual technologies in speech recognition that have led to improvements of this magnitude".[7]

28.4.5 Learning image features using 2d convolutional DBNs

We can extend a convolutional DBN from 1d to 2d in a straightforward way (Lee et al. 2009), as illustrated in Figure 28.8. The results of a 3 layer system trained on four classes of visual objects (cars, motorbikes, faces and airplanes) from the Caltech 101 dataset are shown in Figure 28.9. We only show the results for layers 2 and 3, because layer 1 learns Gabor-like filters that are very similar to those learned by sparse coding, shown in Figure 13.21(b). We see that layer 2 has learned some generic visual parts that are shared amongst object classes, and layer 3 seems to have learned filters that look like grandmother cells, that are specific to individual object classes, and in some cases, to individual objects.

28.5 Discussion

So far, we have been discussing models inspired by low-level processing in the brain. These models have produced useful features for simple classification tasks. But can this pure bottom-up

too early.

7. Source: `http://research.microsoft.com/en-us/news/features/speechrecognition-082911.aspx`.

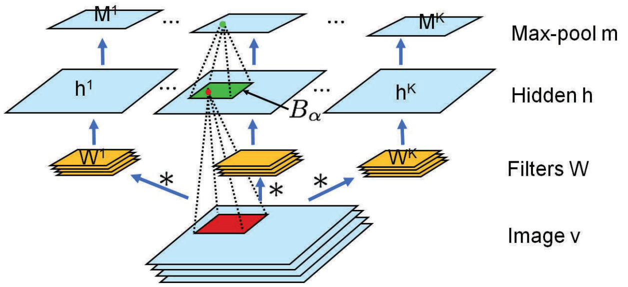


Figure 28.8 A 2d convolutional RBM with max-pooling layers. The input signal is a stack of 2d images (e.g., color planes). Each input layer is passed through a different set of filters. Each hidden unit is obtained by convolving with the appropriate filter, and then summing over the input planes. The final layer is obtained by computing the local maximum within a small window. Source: Figure 1 of (Chen et al. 2010) . Used with kind permission of Bo Chen.

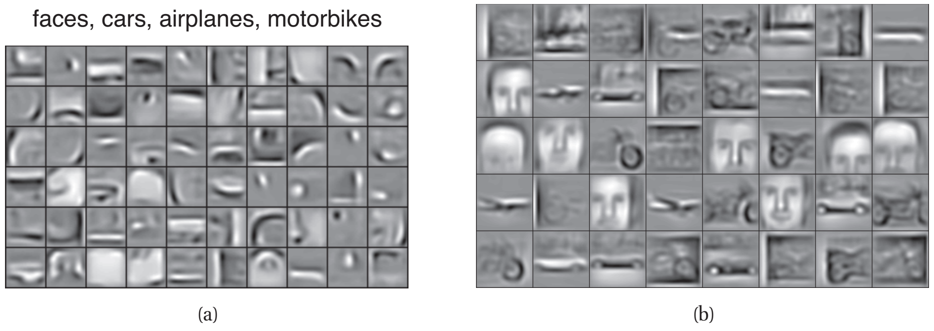


Figure 28.9 Visualization of the filters learned by a convolutional DBN in layers two and three. Source: Figure 3 of (Lee et al. 2009). Used with kind permission of Honglak Lee.

approach scale to more challenging problems, such as scene interpretation or natural language understanding?

To put the problem in perspective, consider the DBN for handwritten digit classification in Figure 28.4(a). This has about 1.6M free parameters ($28 \times 28 \times 500 + 500 \times 500 + 510 \times 2000 = 1,662,000$). Although this is a lot, it is tiny compared to the number of neurons in the brain. As Hinton says,

> This is about as many parameters as 0.002 cubic millimetres of mouse cortex, and several hundred networks of this complexity could fit within a single voxel of a high-resolution fMRI scan. This suggests that much bigger networks may be required to compete with human shape recognition abilities. — (Hinton et al. 2006, p1547).

To scale up to more challenging problems, various groups are using GPUs (see e.g., (Raina et al. 2009)) and/or parallel computing. But perhaps a more efficient approach is to work at a higher level of abstraction, where inference is done in the space of objects or their parts, rather than in the space of bits and pixels. That is, we want to bridge the **signal-to-symbol** divide, where by "symbol" we mean something atomic, that can be combined with other symbols in a compositional way.

The question of how to convert low level signals into a more structured/ "semantic" representation is known as the **symbol grounding** problem (Harnard 1990). Traditionally such symbols are associated with words in natural language, but it seems unlikely we can jump directly from low-level signals to high-level semantic concepts. Instead, what we need is an intermediate level of symbolic or atomic parts.

A very simple way to create such parts from real-valued signals, such as images, is to apply vector quantization. This generates a set of **visual words**. These can then be modelled using some of the techniques from Chapter 27 for modeling bags of words. Such models, however, are still quite "shallow".

It is possible to define, and learn, deep models which use discrete latent parts. Here we just mention a few recent approaches, to give a flavor of the possibilites. (Salakhutdinov et al. 2011) combine RBMs with hierarchical latent Dirichlet allocation methods, trained in an unsupervised way. (Zhu et al. 2010) use latent and-or graphs, trained in a manner similar to a latent structural SVM. A similar approach, based on grammars, is described in (Girshick et al. 2011). What is interesting about these techniques is that they apply data-driven machine learning methods to rich structured/symbolic "AI-style" models. This seems like a promising future direction for machine learning.

Notation

Introduction

It is very difficult to come up with a single, consistent notation to cover the wide variety of data, models and algorithms that we discuss. Furthermore, conventions differ between machine learning and statistics, and between different books and papers. Nevertheless, we have tried to be as consistent as possible. Below we summarize most of the notation used in this book, although individual sections may introduce new notation. Note also that the same symbol may have different meanings depending on the context, although we try to avoid this where possible.

General math notation

Symbol	Meaning
$\lfloor x \rfloor$	Floor of x, i.e., round down to nearest integer
$\lceil x \rceil$	Ceiling of x, i.e., round up to nearest integer
$\mathbf{x} \otimes \mathbf{y}$	Convolution of $\mathbf{x}$ and $\mathbf{y}$
$\mathbf{x} \odot \mathbf{y}$	Hadamard (elementwise) product of $\mathbf{x}$ and $\mathbf{y}$
$a \wedge b$	logical AND
$a \vee b$	logical OR
$\neg a$	logical NOT
$\mathbb{I}(x)$	Indicator function, $\mathbb{I}(x) = 1$ if x is true, else $\mathbb{I}(x) = 0$
∞	Infinity
$\rightarrow$	Tends towards, e.g., $n \rightarrow \infty$
$\propto$	Proportional to, so $y = ax$ can be written as $y \propto x$
$\lvert x \rvert$	Absolute value
$\lvert \mathcal{S} \rvert$	Size (cardinality) of a set
$n!$	Factorial function
∇	Vector of first derivatives
∇^2	Hessian matrix of second derivatives
$\triangleq$	Defined as
$O(\cdot)$	Big-O: roughly means order of magnitude
$\mathbb{R}$	The real numbers
$1 : n$	Range (Matlab convention): $1 : n = \{1, 2, \ldots, n\}$
$\approx$	Approximately equal to
$\mathrm{argmax}_x \, f(x)$	Argmax: the value x that maximizes f

$B(a,b)$	Beta function, $B(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$
$B(\boldsymbol{\alpha})$	Multivariate beta function, $\frac{\prod_k \Gamma(\alpha_k)}{\Gamma(\sum_k \alpha_k)}$
$\binom{n}{k}$	n choose k, equal to $n!/(k!(n-k)!)$
$\delta(x)$	Dirac delta function, $\delta(x) = \infty$ if $x = 0$, else $\delta(x) = 0$
δ_{ij}	Kronecker delta, equals 1 if $i = j$, otherwise equals 0
$\delta_x(y)$	Kronecker delta, equals 1 if $x = y$, otherwise equals 0
$\exp(x)$	Exponential function e^x
$\Gamma(x)$	Gamma function, $\Gamma(x) = \int_0^\infty u^{x-1}e^{-u}du$
$\Psi(x)$	Digamma function, $\Psi(x) = \frac{d}{dx}\log\Gamma(x)$
$\mathcal{X}$	A set from which values are drawn (e.g., $\mathcal{X} = \mathbb{R}^D$)

Linear algebra notation

We use boldface lowercase to denote vectors, such as $\mathbf{a}$, and boldface uppercase to denote matrices, such as $\mathbf{A}$. Vectors are assumed to be column vectors, unless noted otherwise.

Symbol	Meaning
$\mathbf{A} \succ 0$	$\mathbf{A}$ is a positive definite matrix
$\text{tr}(\mathbf{A})$	Trace of a matrix
$\det(\mathbf{A})$	Determinant of matrix $\mathbf{A}$
$\|\mathbf{A}\|$	Determinant of matrix $\mathbf{A}$
$\mathbf{A}^{-1}$	Inverse of a matrix
$\mathbf{A}^\dagger$	Pseudo-inverse of a matrix
$\mathbf{A}^T$	Transpose of a matrix
$\mathbf{a}^T$	Transpose of a vector
$\text{diag}(\mathbf{a})$	Diagonal matrix made from vector $\mathbf{a}$
$\text{diag}(\mathbf{A})$	Diagonal vector extracted from matrix $\mathbf{A}$
$\mathbf{I}$ or $\mathbf{I}_d$	Identity matrix of size $d \times d$ (ones on diagonal, zeros off)
$\mathbf{1}$ or $\mathbf{1}_d$	Vector of ones (of length d)
$\mathbf{0}$ or $\mathbf{0}_d$	Vector of zeros (of length d)
$\|\|\mathbf{x}\|\| = \|\|\mathbf{x}\|\|_2$	Euclidean or ℓ_2 norm $\sqrt{\sum_{j=1}^d x_j^2}$
$\|\|\mathbf{x}\|\|_1$	ℓ_1 norm $\sum_{j=1}^d \|x_j\|$
$\mathbf{A}_{:,j}$	j'th column of matrix
$\mathbf{A}_{i,:}$	transpose of i'th row of matrix (a column vector)
A_{ij}	Element (i,j) of matrix $\mathbf{A}$
$\mathbf{x} \otimes \mathbf{y}$	Tensor product of $\mathbf{x}$ and $\mathbf{y}$

Probability notation

We denote random and fixed scalars by lower case, random and fixed vectors by bold lower case, and random and fixed matrices by bold upper case. Occastionally we use non-bold upper case to denote scalar random variables. Also, we use $p()$ for both discrete and continuous random variables.

Symbol	Meaning
$X \perp Y$	X is independent of Y
$X \not\perp Y$	X is not independent of Y
$X \perp Y \vert Z$	X is conditionally independent of Y given Z
$X \not\perp Y \vert Z$	X is not conditionally independent of Y given Z
$X \sim p$	X is distributed according to distribution p
$\boldsymbol{\alpha}$	Parameters of a Beta or Dirichlet distribution
$\mathrm{cov}\left[\mathbf{x}\right]$	Covariance of $\mathbf{x}$
$\mathbb{E}\left[X\right]$	Expected value of X
$\mathbb{E}_q\left[X\right]$	Expected value of X wrt distribution q
$\mathbb{H}\left(X\right)$ or $\mathbb{H}\left(p\right)$	Entropy of distribution $p(X)$
$\mathbb{I}\left(X;Y\right)$	Mutual information between X and Y
$\mathbb{KL}\left(p \vert\vert q\right)$	KL divergence from distribution p to q
$\ell(\boldsymbol{\theta})$	Log-likelihood function
$L(\theta, a)$	Loss function for taking action a when true state of nature is θ
λ	Precision (inverse variance) $\lambda = 1/\sigma^2$
$\boldsymbol{\Lambda}$	Precision matrix $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$
$\mathrm{mode}\left[X\right]$	Most probable value of X
μ	Mean of a scalar distribution
$\boldsymbol{\mu}$	Mean of a multivariate distribution
$p(x)$	Probability density or mass function
$p(x \vert y)$	Conditional probability density of x given y
Φ	cdf of standard normal
ϕ	pdf of standard normal
$\boldsymbol{\pi}$	multinomial parameter vector, Stationary distribution of Markov chain
ρ	Correlation coefficient
$\mathrm{sigm}(x)$	Sigmoid (logistic) function, $\frac{1}{1+e^{-x}}$
σ^2	Variance
$\boldsymbol{\Sigma}$	Covariance matrix
$\mathrm{var}\left[x\right]$	Variance of x
ν	Degrees of freedom parameter
Z	Normalization constant of a probability distribution

Machine learning/statistics notation

In general, we use upper case letters to denote constants, such as C, D, K, N, S, T, etc. We use lower case letters as dummy indexes of the appropriate range, such as $c = 1 : C$ to index classes, $j = 1 : D$ to index input features, $k = 1 : K$ to index states or clusters, $s = 1 : S$ to index samples, $t = 1 : T$ to index time, etc. To index data cases, we use the notation $i = 1 : N$, although the notation $n = 1 : N$ is also widely used.

We use $\mathbf{x}$ to represent an observed data vector. In a supervised problem, we use y or $\mathbf{y}$ to represent the desired output label. We use $\mathbf{z}$ to represent a hidden variable. Sometimes we also use q to represent a hidden discrete variable.

Symbol	Meaning
C	Number of classes
D	Dimensionality of data vector (number of features)
R	Number of outputs (response variables)
$\mathcal{D}$	Training data $\mathcal{D} = \{\mathbf{x}_i \vert i = 1 : N\}$ or $\mathcal{D} = \{(\mathbf{x}_i, y_i) \vert i = 1 : N\}$
$\mathcal{D}_{\text{test}}$	Test data
$J(\boldsymbol{\theta})$	Cost function
K	Number of states or dimensions of a variable (often latent)
$\kappa(\mathbf{x}, \mathbf{y})$	Kernel function
$\mathbf{K}$	Kernel matrix
λ	Strength of ℓ_2 or ℓ_1 regularizer
N	Number of data cases
N_c	Number of examples of class c, $N_c = \sum_{n=1}^{N} \mathbb{I}(y_n = c)$
$\boldsymbol{\phi}(\mathbf{x})$	Basis function expansion of feature vector $\mathbf{x}$
$\boldsymbol{\Phi}$	Basis function expansion of design matrix X
$q()$	Approximate or proposal distribution
$Q(\boldsymbol{\theta}, \boldsymbol{\theta}_{old})$	Auxiliary function in EM
S	Number of samples
T	Length of a sequence
$T(\mathcal{D})$	Test statistic for data
$\mathbf{T}$	Transition matrix of Markov chain
$\boldsymbol{\theta}$	Parameter vector
$\boldsymbol{\theta}^{(s)}$	s'th sample of parameter vector
$\hat{\boldsymbol{\theta}}$	Estimate (usually MLE or MAP) of $\boldsymbol{\theta}$
$\hat{\boldsymbol{\theta}}_{ML}$	Maximum likelihood estimate of $\boldsymbol{\theta}$
$\hat{\boldsymbol{\theta}}_{MAP}$	MAP estimate of $\boldsymbol{\theta}$
$\overline{\boldsymbol{\theta}}$	Estimate (usually posterior mean) of $\boldsymbol{\theta}$
$\mathbf{w}$	Vector of regression weights (called $\boldsymbol{\beta}$ in statistics)
$\mathbf{W}$	Matrix of regression weights
x_{ij}	Component (i.e., feature) j of data case i, for $i = 1 : N$,$j = 1 : D$
$\mathbf{x}_i$	Training case, $i = 1 : N$
$\mathbf{X}$	Design matrix of size $N \times D$
$\overline{\mathbf{x}}$	Empirical mean $\overline{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i$
$\tilde{\mathbf{x}}$	Future test case
$\mathbf{x}_*$	Future test case
$\mathbf{y}$	Vector of all training labels $\mathbf{y} = (y_1, \ldots, y_N)$
z_{ij}	Latent component j for case i

Graphical model notation

In graphical models, we index nodes by $s, t, u \in \mathcal{V}$, and states by $i, j, k \in \mathcal{X}$.

Symbol	Meaning
$s \sim t$	Node s is connected to node t
bel	Belief function
$\mathcal{C}$	Cliques of a graph
ch_j	Child of node j in a DAG
desc_j	Descendants of node j in a DAG
G	A graph
$\mathcal{E}$	Edges of a graph
mb_t	Markov blanket of node t
nbd_t	Neighborhood of node t
pa_t	Parents of node t in a DAG
pred_t	Predecessors of node t in a DAG wrt some ordering
$\psi_c(\mathbf{x}_c)$	Potential function for clique c
$\mathcal{S}$	Separators of a graph
θ_{sjk}	prob. node s is in state k given its parents are in states j
$\mathcal{V}$	Nodes of a graph

List of commonly used abbreviations

Abbreviation	Meaning
cdf	Cumulative distribution function
CPD	Conditional probability distribution
CPT	Conditional probability table
CRF	Conditional random field
DAG	Directed acyclic graphic
DGM	Directed graphical model
EB	Empirical Bayes
EM	Expectation maximization algorithm
EP	Expectation propagation
GLM	Generalized linear model
GMM	Gaussian mixture model
HMM	Hidden Markov model
iid	Independent and identically distributed
iff	If and only if
KL	Kullback Leibler divergence
LDS	Linear dynamical system
LHS	Left hand side (of an equation)
MAP	Maximum A Posterior estimate
MCMC	Markov chain Monte Carlo
MH	Metropolis Hastings
MLE	Maximum likelihood estimate
MPM	Maximum of Posterior Marginals
MRF	Markov random field
MSE	Mean squared error
NLL	Negative log likelihood
OLS	Ordinary least squares
pd	Positive definite (matrix)
pdf	Probability density function
pmf	Probability mass function
RBPF	Rao-Blackwellised particle filter
RHS	Right hand side (of an equation)
RJMCMC	Reversible jump MCMC
RSS	Residual sum of squares
SLDS	Switching linear dynamical system
SSE	Sum of squared errors
UGM	Undirected graphical model
VB	Variational Bayes
wrt	With respect to

Bibliography

Abend, K., T. J. Harley, and L. N. Kanal (1965). Classification of Binary Random Patterns. *IEEE Transactions on Information Theory 11(4)*, 538–544.

Ackley, D., G. Hinton, and T. Sejnowski (1985). A learning algorithm for boltzmann machines. *Cognitive Science 9*, 147–169.

Adams, R. P., H. Wallach, and Z. Ghahramani (2010). Learning the structure of deep sparse graphical models. In *AI/Statistics*.

Aggarwal, D. and S. Merugu (2007). Predictive discrete latent factor models for large scale dyadic data. In *Proc. of the Int'l Conf. on Knowledge Discovery and Data Mining*.

Ahmed, A. and E. Xing (2007). On tight approximate inference of the logistic-normal topic admixture model. In *AI/Statistics*.

Ahn, J.-H. and J.-H. Oh (2003). A Constrained EM Algorithm for Principal Component Analysis. *Neural Computation 15*, 57–65.

Ahn, S., A. Korattikara, and M. Welling (2012). Bayesian Posterior Sampling via Stochastic Gradient Fisher Scoring. In *Intl. Conf. on Machine Learning*.

Airoldi, E., D. Blei, S. Fienberg, and E. Xing (2008). Mixed-membership stochastic blockmodels. *J. of Machine Learning Research 9*, 1981–2014.

Aitchison, J. (1982). The statistical analysis of compositional data. *J. of Royal Stat. Soc. Series B 44*(2), 139–177.

Aji, S. M. and R. J. McEliece (2000, March). The generalized distributive law. *IEEE Trans. Info. Theory 46*(2), 325–343.

Alag, S. and A. Agogino (1996). Inference using message propogation and topology transformation in vector Gaussian continuous networks. In *UAI*.

Albers, C., M. Leisink, and H. Kappen (2006). The Cluster Variation Method for Efficient Linkage Analysis on Extended Pedigrees. *BMC Bioinformatics 7*.

Albert, J. and S. Chib (1993). Bayesian analysis of binary and polychotomous response data. *J. of the Am. Stat. Assoc. 88*(422), 669–679.

Allwein, E., R. Schapire, and Y. Singer (2000). Reducing multiclass to binary: A unifying approach for margin classifiers. *J. of Machine Learning Research*, 113–141.

Aloise, D., A. Deshpande, P. Hansen, and P. Popat (2009). NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning 75*, 245–249.

Alpaydin, E. (2004). *Introduction to machine learning*. MIT Press.

Altun, Y., T. Hofmann, and I. Tsochantaridis (2006). Large Margin Methods for Structured and Interdependent Output Variables. In G. Bakir, T. Hofmann, B. Scholkopf, A. Smola, B. Taskar, and S. Vishwanathan (Eds.), *Machine Learning with Structured Outputs*. MIT Press.

Amir, E. (2010). Approximation Algorithms for Treewidth. *Algorithmica 56*(4), 448.

Amir, E. and S. McIlraith (2005). Partition-based logical reasoning for first-order and propositional theories. *Artificial Intelligence 162*(1), 49–88.

Ando, R. and T. Zhang (2005). A framework for learning predictive structures from multiple tasks and unlabeled data. *J. of Machine Learning Research 6*, 1817–1853.

Andrews, D. and C. Mallows (1974). Scale mixtures of Normal distributions. *J. of Royal Stat. Soc. Series B 36*, 99–102.

Andrieu, C., N. de Freitas, and A. Doucet (2000). Sequential Bayesian estimation and model selection for dynamic kernel machines. Technical report, Cambridge Univ.

Andrieu, C., N. de Freitas, and A. Doucet (2001). Robust Full Bayesian Learning for Radial Basis Networks. *Neural Computation 13*(10), 2359–2407.

Andrieu, C., N. de Freitas, A. Doucet, and M. Jordan (2003). An introduction to MCMC for machine learning. *Machine Learning 50*, 5–43.

Andrieu, C., A. Doucet, and V. Tadic (2005). Online EM for parameter estimation in nonlinear-non Gaussian state-space models. In *Proc. IEEE CDC*.

Andrieu, C. and J. Thoms (2008). A tutorial on adaptive MCMC. *Statistical Computing 18*, 343–373.

Aoki, M. (1987). *State space modeling of time series*. Springer.

Archambeau, C. and F. Bach (2008). Sparse probabilistic projections. In *NIPS*.

Argyriou, A., T. Evgeniou, and M. Pontil (2008). Convex multi-task feature learning. *Machine Learning 73*(3), 243–272.

Armagan, A., D. Dunson, and J. Lee (2011). Generalized double pareto shrinkage. Technical report, Duke.

Armstrong, H. (2005). *Bayesian estimation of decomposable Gaussian graphical models*. Ph.D. thesis, UNSW.

Armstrong, H., C. Carter, K. Wong, and R. Kohn (2008). Bayesian Covariance Matrix Estimation using a Mixture of Decomposable Graphical Models. *Statistics and Computing*, 1573–1375.

Arnborg, S., D. G. Corneil, and A. Proskurowski (1987). Complexity of finding embeddings in a k-tree. *SIAM J. on Algebraic and Discrete Methods 8*, 277–284.

Arora, S. and B. Barak (2009). *Complexity Theory: A Modern Approach*. Cambridge.

Arthur, D. and S. Vassilvitskii (2007). k-means++: the advantages of careful seeding. In *Proc. 18th ACM-SIAM symp. on Discrete algorithms*, pp. 1027âĂŞ1035.

Arulampalam, M., S. Maskell, N. Gordon, and T. Clapp (2002, February). A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking. *IEEE Trans. on Signal Processing 50*(2), 174–189.

Asavathiratham, C. (2000). *The Influence Model: A Tractable Representation for the Dynamics of Networked Markov Chains*. Ph.D. thesis, MIT, Dept. EECS.

Atay-Kayis, A. and H. Massam (2005). A Monte Carlo method for computing the marginal likelihood in non-decomposable Gaussian graphical models. *Biometrika 92*, 317–335.

Attenberg, J., K. Weinberger, A. Smola, A. Dasgupta, and M. Zinkevich (2009). Collaborative spam filtering with the hashing trick. In *Virus Bulletin*.

Attias, H. (1999). Independent factor analysis. *Neural Computation 11*, 803–851.

Attias, H. (2000). A variational Bayesian framework for graphical models. In *NIPS-12*.

Bach, F. (2008). Bolasso: Model Consistent Lasso Estimation through the Bootstrap. In *Intl. Conf. on Machine Learning*.

Bach, F. and M. Jordan (2001). Thin junction trees. In *NIPS*.

Bach, F. and M. Jordan (2005). A probabilistic interpretation of canonical correlation analysis. Technical Report 688, U. C. Berkeley.

Bach, F. and E. Moulines (2011). Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *NIPS*.

Bahmani, B., B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii (2012). Scalable k-Means++. In *VLDB*.

Bakker, B. and T. Heskes (2003). Task Clustering and Gating for Bayesian Multitask Learning. *J. of Machine Learning Research 4*, 83–99.

Baldi, P. and Y. Chauvin (1994). Smooth online learning algorithms for hidden Markov models. *Neural Computation 6*, 305–316.

Balding, D. (2006). A tutorial on statistical methods for population association studies. *Nature Reviews Genetics 7*, 81–91.

Banerjee, O., L. E. Ghaoui, and A. d'Aspremont (2008). Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data. *J. of Machine Learning Research 9*, 485–516.

Bar-Shalom, Y. and T. Fortmann (1988). *Tracking and data association*. Academic Press.

Bar-Shalom, Y. and X. Li (1993). *Estimation and Tracking: Principles, Techniques and Software*. Artech House.

Barash, Y. and N. Friedman (2002). Context-specific Bayesian clustering for gene expression data. *J. Comp. Bio. 9*, 169–191.

Barber, D. (2006). Expectation Correction for Smoothed Inference in Switching Linear Dynamical Systems. *J. of Machine Learning Research 7*, 2515–2540.

Barber, D. and C. Bishop (1998). Ensemble Learning in Bayesian Neural Networks. In C. Bishop (Ed.), *Neural Networks and Machine Learning*, pp. 215–237. Springer.

Barber, D. and S. Chiappa (2007). Unified inference for variational bayesian linear gaussian state space models. In *NIPS*.

Barbieri, M. and J. Berger (2004). Optimal predictive model selection. *Annals of Statistics 32*, 870–897.

Bartlett, P., M. Jordan, and J. McAuliffe (2006). Convexity, Classification, and Risk Bounds. *J. of the Am. Stat. Assoc. 101*(473), 138–156.

Baruniak, R. (2007). Compressive sensing. *IEEE Signal Processing Magazine*.

Barzilai, J. and J. Borwein (1988). Two point step size gradient methods. *IMA J. of Numerical Analysis 8*, 141–148.

Basu, S., T. Choudhury, B. Clarkson, and A. Pentland (2001). Learning human interactions with the influence model. Technical Report 539, MIT Media Lab. ftp://whitechapel.media.mit.edu/pub/tech-reports/TR-539-ABSTRACT.html.

Baum, L. E., T. Petrie, G. Soules, and N. Weiss (1970). A maximization technique occuring in the statistical analysis of probabalistic functions in markov chains. *The Annals of Mathematical Statistics 41*, 164–171.

Beal, M. (2003). *Variational Algorithms for Approximate Bayesian Inference*. Ph.D. thesis, Gatsby Unit.

Beal, M. and Z. Ghahramani (2006). Variational Bayesian Learning of Directed Graphical Models with Hidden Variables. *Bayesian Analysis 1*(4).

Beal, M. J., Z. Ghahramani, and C. E. Rasmussen (2002). The infinite hidden Markov model. In *NIPS-14*.

Beck, A. and M. Teboulle (2009). A fast iterative shrinkage-thresholding algorothm for linear inverse problems. *SIAM J. on Imaging Sciences 2*(1), 183–202.

Beinlich, I., H. Suermondt, R. Chavez, and G. Cooper (1989). The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proc. of the Second European Conf. on AI in Medicine*, pp. 247–256.

Bekkerman, R., M. Bilenko, and J. Langford (Eds.) (2011). *Scaling Up Machine Learning*. Cambridge.

Bell, A. J. and T. J. Sejnowski (1995). An information maximisation approach to blind separation and blind deconvolution. *Neural Computation 7*(6), 1129–1159.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning 2*(1), 1–127.

Bengio, Y. and S. Bengio (2000). Modeling high-dimensional discrete data with multi-layer neural networks. In *NIPS*.

Bengio, Y., O. Delalleau, N. Roux, J. Paiement, P. Vincent, and M. Ouimet (2004). Learning eigenfunctions links spectral embedding and kernel PCA. *Neural Computation 16*, 2197–2219.

Bengio, Y. and P. Frasconi (1995). Diffusion of context and credit information in markovian models. *J. of AI Research 3*, 249–270.

Bengio, Y. and P. Frasconi (1996). Input/output HMMs for sequence processing. *IEEE Trans. on Neural Networks 7*(5), 1231–1249.

Bengio, Y., P. Lamblin, D. Popovici, and H. Larochelle (2007). Greedy layer-wise training of deep networks. In *NIPS*.

Berchtold, A. (1999). The double chain markov model. *Comm. Stat. Theor. Methods 28*, 2569–2589.

Berger, J. (1985). Bayesian salesmanship. In P. K. Goel and A. Zellner (Eds.), *Bayesian Inference and Decision Techniques with Applications: Essays in Honor of Bruno deFinetti*. North-Holland.

Berger, J. and R. Wolpert (1988). *The Likelihood Principle*. The Institute of Mathematical Statistics. 2nd edition.

Berkhin, P. (2006). A survey of clustering datamining techniques. In J. Kogan, C. Nicholas, and M. Teboulle (Eds.), *Grouping Multidimensional Data: Recent Advances in Clustering*, pp. 25–71. Springer.

Bernardo, J. and A. Smith (1994). *Bayesian Theory*. John Wiley.

Berrou, C., A. Glavieux, and P. Thitimajashima (1993). Near Shannon limit error-correcting coding and decoding: Turbo codes. *Proc. IEEE Intl. Comm. Conf.*.

Berry, D. and Y. Hochberg (1999). Bayesian perspectives on multiple comparisons. *J. Statist. Planning and Inference 82*, 215–227.

Bertele, U. and F. Brioschi (1972). *Nonserial Dynamic Programming*. Academic Press.

Bertsekas, D. (1997). *Parallel and Distribution Computation: Numerical Methods*. Athena Scientific.

Bertsekas, D. (1999). *Nonlinear Programming* (Second ed.). Athena Scientific.

Bertsekas, D. and J. Tsitsiklis (2008). *Introduction to Probability*. Athena Scientific. 2nd Edition.

Besag, J. (1975). Statistical analysis of non-lattice data. *The Statistician 24*, 179–196.

Bhatnagar, N., C. Bogdanov, and E. Mossel (2010). The computational complexity of estimating convergence time. Technical report, .

Bhattacharya, A. and D. B. Dunson (2011). Simplex factor models for multivariate unordered categorical data. *J. of the Am. Stat. Assoc.*. To appear.

Bickel, P. and E. Levina (2004). Some theory for Fisher's linear discriminant function, "Naive Bayes", and some alternatives when there are many more variables than observations. *Bernoulli 10*, 989–1010.

Bickson, D. (2009). *Gaussian Belief Propagation: Theory and Application*. Ph.D. thesis, Hebrew University of Jerusalem.

Bilmes, J. (2000). Dynamic Bayesian multinets. In *UAI*.

Bilmes, J. A. (2001). Graphical models and automatic speech recognition. Technical Report UWEETR-2001-0005, Univ. Washington, Dept. of Elec. Eng.

Binder, J., D. Koller, S. J. Russell, and K. Kanazawa (1997). Adaptive probabilistic networks with hidden variables. *Machine Learning 29*, 213–244.

Binder, J., K. Murphy, and S. Russell (1997). Space-efficient inference in dynamic probabilistic networks. In *Intl. Joint Conf. on AI*.

Birnbaum, A. (1962). On the foundations of statistical infernece. *J. of the Am. Stat. Assoc. 57*, 269–326.

Bishop, C. (1999). Bayesian PCA. In *NIPS*.

Bishop, C. (2006a). *Pattern recognition and machine learning*. Springer.

Bishop, C. (2006b). *Pattern recognition and machine learning*. Springer.

Bishop, C. and G. James (1993). Analysis of multiphase flows using dual-energy densitometry and neural networks. *Nuclear Instruments and Methods in Physics Research A327*, 580–593.

Bishop, C. and M. Svensén (2003). Bayesian hierarchical mixtures of experts. In *UAI*.

Bishop, C. and M. Tipping (2000). Variational relevance vector machines. In *UAI*.

Bishop, C. M. (1994). Mixture density networks. Technical Report NCRG 4288, Neural Computing Research Group, Department of Computer Science, Aston University.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Clarendon Press.

Bishop, Y., S. Fienberg, and P. Holland (1975). *Discrete Multivariate Analysis: Theory and Practice*. MIT Press.

Bistarelli, S., U. Montanari, and F. Rossi (1997). Semiring-based constraint satisfaction and optimization. *J. of the ACM 44*(2), 201–236.

Blake, A., P. Kohli, and C. Rother (Eds.) (2011). *Advances in Markov Random Fields for Vision and Image Processing*. MIT Press.

Blei, D. and J. Lafferty (2006a). Correlated topic models. In *NIPS*.

Blei, D. and J. Lafferty (2006b). Dynamic topic models. In *Intl. Conf. on Machine Learning*, pp. 113–120.

Blei, D. and J. Lafferty (2007). A Correlated Topic Model of "Science". *Annals of Applied Stat. 1*(1), 17–35.

Blei, D. and J. McAuliffe (2010, March). Supervised topic models. Technical report, Princeton.

Blei, D., A. Ng, and M. Jordan (2003). Latent dirichlet allocation. *J. of Machine Learning Research 3*, 993–1022.

Blumensath, T. and M. Davies (2007). On the difference between Orthogonal Matching Pursuit and Orthogonal Least Squares. Technical report, U. Edinburgh.

Bo, L., C. Sminchisescu, A. Kanaujia, and D. Metaxas (2008). Fast Algorithms for Large Scale Conditional 3D Prediction. In *CVPR*.

Bohning, D. (1992). Multinomial logistic regression algorithm. *Annals of the Inst. of Statistical Math. 44*, 197–200.

Bollen, K. (1989). *Structural Equation Models with Latent Variables*. John Wiley & Sons.

Bordes, A., L. Bottou, and P. Gallinari (2009, July). Sgd-qn: Careful quasi-newton stochastic gradient descent. *J. of Machine Learning Research 10*, 1737–1754.

Bordes, A., L. Bottou, P. Gallinari, J. Chang, and S. A. Smith (2010). Erratum: SGDQN is Less Careful than Expected. *J. of Machine Learning Research 11*, 2229–2240.

Boser, B. E., I. M. Guyon, and V. N. Vapnik (1992). A training algorithm for optimal margin classifiers. In *Proc. of the Workshop on Computational Learning Theory*.

Bottcher, S. G. and C. Dethlefsen (2003). deal: A package for learning bayesian networks. *J. of Statistical Software 8*(20).

Bottolo, L. and S. Richardson (2010). Evolutionary stochastic search. *Bayesian Analysis 5*(3), 583–618.

Bottou, L. (1998). Online algorithms and stochastic approximations. In D. Saad (Ed.), *Online Learning and Neural Networks*. Cambridge.

Bottou, L. (2007). Learning with large datasets (nips tutorial).

Bottou, L., O. Chapelle, D. DeCoste, and J. Weston (Eds.) (2007). *Large Scale Kernel Machines*. MIT Press.

Bouchard, G. (2007). Efficient bounds for the softmax and applications to approximate inference in hybrid models. In *NIPS 2007 Workshop on Approximate Inference in Hybrid Models*.

Bouchard-Cote, A. and M. Jordan (2009). Optimization of structured mean field objectives. In *UAI*.

Bowman, A. and A. Azzalini (1997). *Applied Smoothing Techniques for Data Analysis*. Oxford.

Box, G. and N. Draper (1987). *Empirical Model-Building and Response Surfaces*. Wiley.

Box, G. and G. Tiao (1973). *Bayesian inference in statistical analysis*. Addison-Wesley.

Boyd, S. and L. Vandenberghe (2004). *Convex optimization*. Cambridge.

Boyen, X. and D. Koller (1998). Tractable inference for complex stochastic processes. In *UAI*.

Boykov, Y., O. Veksler, and R. Zabih (2001). Fast approximate energy minimization via graph cuts. *IEEE Trans. on Pattern Analysis and Machine Intelligence 23*(11).

Brand, M. (1996). Coupled hidden Markov models for modeling interacting processes. Technical Report 405, MIT Lab for Perceptual Computing.

Brand, M. (1999). Structure learning in conditional probability models via an entropic prior and parameter extinction. *Neural Computation 11*, 1155–1182.

Braun, M. and J. McAuliffe (2010). Variational Inference for Large-Scale Models of Discrete Choice. *J. of the Am. Stat. Assoc. 105*(489), 324–335.

Breiman, L. (1996). Bagging predictors. *Machine Learning 24*, 123–140.

Breiman, L. (1998). Arcing classifiers. *Annals of Statistics 26*, 801–849.

Breiman, L. (2001a). Random forests. *Machine Learning 45*(1), 5–32.

Breiman, L. (2001b). Statistical modeling: the two cultures. *Statistical Science 16*(3), 199–231.

Breiman, L., J. Friedman, and R. Olshen (1984). *Classification and regression trees*. Wadsworth.

Breslow, N. E. and D. G. Clayton (1993). Approximate inference in generalized linear mixed models. *J. of the Am. Stat. Assoc. 88*(421), 9–25.

Briers, M., A. Doucet, and S. Maskel (2010). Smoothing algorithms for state-space models. *Annals of the Institute of Statistical Mathematics 62*(1), 61–89.

Brochu, E., M. Cora, and N. de Freitas (2009, November). A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. Technical Report TR-2009-23, Department of Computer Science, University of British Columbia.

Brooks, S. and G. Roberts (1998). Assessing convergence of Markov Chain Monte Carlo algorithms. *Statistics and Computing 8*, 319–335.

Brown, L., T. Cai, and A. DasGupta (2001). Interval estimation for a binomial proportion. *Statistical Science 16*(2), 101–133.

Brown, M. P., R. Hughey, A. Krogh, I. S. Mian, K. Sjölander, and D. Haussler (1993). Using dirichlet mixtures priors to derive hidden Markov models for protein families. In *Intl. Conf. on Intelligent Systems for Molecular Biology*, pp. 47–55.

Brown, P., M. Vannucci, and T. Fearn (1998). Multivariate Bayesian variable selection and prediction. *J. of the Royal Statistical Society B 60*(3), 627–641.

Bruckstein, A., D. Donoho, and M. Elad (2009). From sparse solutions of systems of equations to sparse modeling of signals and images. *SIAM Review 51*(1), 34–81.

Bryson, A. and Y.-C. Ho (1969). *Applied optimal control: optimization, estimation, and control*. Blaisdell Publishing Company.

Buhlmann, P. and T. Hothorn (2007). Boosting Algorithms: Regularization, Prediction and Model Fitting. *Statistical Science 22*(4), 477–505.

Buhlmann, P. and S. van de Geer (2011). *Statistics for High-Dimensional Data: Methodology, Theory and Applications*. Springer.

Buhlmann, P. and B. Yu (2003). Boosting with the L2 loss: Regression and classification. *J. of the Am. Stat. Assoc. 98*(462), 324–339.

Buhlmann, P. and B. Yu (2006). Sparse boosting. *J. of Machine Learning Research 7*, 1001–1024.

Bui, H., S. Venkatesh, and G. West (2002). Policy Recognition in the Abstract Hidden Markov Model. *J. of AI Research 17*, 451–499.

Buntine, W. (2002). Variational Extensions to EM and Multinomial PCA. In *Intl. Conf. on Machine Learning*.

Buntine, W. and A. Jakulin (2004). Applying Discrete PCA in Data Analysis. In *UAI*.

Buntine, W. and A. Jakulin (2006). Discrete Component Analysis. In *Subspace, Latent Structure and Feature Selection: Statistical and Optimization Perspectives Workshop*.

Buntine, W. and A. Weigend (1991). Bayesian backpropagation. *Complex Systems 5*, 603–643.

Burges, C. J., T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender (2005). Learning to rank using gradient descent. In *Intl. Conf. on Machine Learning*, pp. 89–96.

Burkard, R., M. Dell'Amico, and S. Martello (2009). *Assignment Problems*. SIAM.

Byran, K. and T. Leise (2006). The 25,000,000,000 Eigenvector: The Linear Algebra behind Google. *SIAM Review 48*(3).

Calvetti, D. and E. Somersalo (2007). *Introduction to Bayesian Scientific Computing*. Springer.

Candes, E., J. Romberg, and T. Tao (2006). Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE. Trans. Inform. Theory 52*(2), 489–509.

Candes, E. and M. Wakin (2008, March). An introduction to compressive sampling. *IEEE Signal Processing Magazine 21*.

Candes, E., M. Wakin, and S. Boyd (2008). Enhancing sparsity by reweighted l1 minimization. *J. of Fourier Analysis and Applications 1*, 877–905.

Cannings, C., E. A. Thompson, and M. H. Skolnick (1978). Probability functions in complex pedigrees. *Advances in Applied Probability 10*, 26–61.

Canny, J. (2004). Gap: a factor model for discrete data. In *Proc. Annual Intl. ACM SIGIR Conference*, pp. 122–129.

Cao, Z., T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li (2007). Learning to rank: From pairwise approach to listwise approach. In *Intl. Conf. on Machine Learning*, pp. 129âĂŞ136.

Cappe, O. (2010). Online Expectation Maximisation. In K. Mengersen, M. Titterington, and C. Robert (Eds.), *Mixtures*.

Cappe, O. and E. Mouline (2009, June). Online EM Algorithm for Latent Data Models. *J. of Royal Stat. Soc. Series B 71*(3), 593–613.

Cappe, O., E. Moulines, and T. Ryden (2005). *Inference in Hidden Markov Models*. Springer.

Carbonetto, P. (2003). Unsupervised statistical models for general object recognition. Master's thesis, University of British Columbia.

Carlin, B. P. and T. A. Louis (1996). *Bayes and Empirical Bayes Methods for Data Analysis*. Chapman and Hall.

Caron, F. and A. Doucet (2008). Sparse Bayesian nonparametric regression. In *Intl. Conf. on Machine Learning*.

Carreira-Perpinan, M. and C. Williams (2003). An isotropic gaussian mixture can have more modes than components. Technical Report EDI-INF-RR-0185, School of Informatics, U. Edinburgh.

Carter, C. and R. Kohn (1994). On Gibbs sampling for state space models. *Biometrika 81*(3), 541–553.

Carterette, B., P. Bennett, D. Chickering, and S. Dumais (2008). Here or There: Preference Judgments for Relevance. In *Proc. ECIR*.

Caruana, R. (1998). A dozen tricks with multitask learning. In G. Orr and K.-R. Mueller (Eds.), *Neural Networks: Tricks of the Trade*. Springer-Verlag.

Caruana, R. and A. Niculescu-Mizil (2006). An empirical comparison of supervised learning algorithms. In *Intl. Conf. on Machine Learning*.

Carvahlo, C., N. Polson, and J. Scott (2010). The horseshoe estimator for sparse signals. *Biometrika 97*(2), 465.

Carvahlo, L. and C. Lawrence (2007). Centroid estimation in discrete high-dimensional spaces with applications in biology. *Proc. of the National Academy of Science, USA 105*(4).

Carvalho, C. M. and M. West (2007). Dynamic matrix-variate graphical models. *Bayesian Analysis 2*(1), 69–98.

Casella, G. and R. Berger (2002). *Statistical inference*. Duxbury. 2nd edition.

Castro, M., M. Coates, and R. D. Nowak (2004). Likelihood based hierarchical clustering. *IEEE Trans. in Signal Processing 52*(8), 230.

Celeux, G. and J. Diebolt (1985). The SEM algorithm: A probabilistic teacher derive from the EM algorithm for the mixture problem. *Computational Statistics Quarterly 2*, 73–82.

Cemgil, A. T. (2001). A technique for painless derivation of kalman filtering recursions. Technical report, U. Nijmegen.

Cesa-Bianchi, N. and G. Lugosi (2006). *Prediction, learning, and games*. Cambridge University Press.

Cevher, V. (2009). Learning with compressible priors. In *NIPS*.

Chai, K. M. A. (2010). *Multi-task learning with Gaussian processes*. Ph.D. thesis, U. Edinburgh.

Chang, H., Y. Weiss, and W. Freeman (2009). Informative Sensing. Technical report, Hebrew U. Submitted to IEEE Transactions on Info. Theory.

Chang, J. and D. Blei (2010). Hierarchical relational models for document networks. *The Annals of Applied Statistics 4*(1), 124–150.

Chang, J., J. Boyd-Graber, S. Gerrish, C. Wang, and D. Blei (2009). Reading tea leaves: How humans interpret topic models. In *NIPS*.

Chapelle, O. and L. Li (2011). An empirical evaluation of Thompson sampling. In *NIPS*.

Chartrand, R. and W. Yin (2008). Iteratively reweighted algorithms for compressive sensing. In *Intl. Conf. on Acoustics, Speech and Signal Proc.*

Chechik, G., A. G. N. Tishby, and Y. Weiss (2005). Information bottleneck for gaussian variables. *J. of Machine Learning Research 6*, 165âĂŞ188.

Cheeseman, P., J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman (1988). Autoclass: A Bayesian classification system. In *Proc. of the Fifth Intl. Workshop on Machine Learning*.

Cheeseman, P. and J. Stutz (1996). Bayesian classification (autoclass): Theory and results. In Fayyad, Pratetsky-Shapiro, Smyth, and Uthurasamy (Eds.), *Advances in Knowledge Discovery and Data Mining*. MIT Press.

Chen, B., K. Swersky, B. Marlin, and N. de Freitas (2010). Sparsity priors and boosting for learning localized distributed feature representations. Technical report, UBC.

Chen, B., J.-A. Ting, B. Marlin, and N. de Freitas (2010). Deep learning of invariant spatio-temporal features from video. In *NIPS Workshop on Deep Learning*.