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.

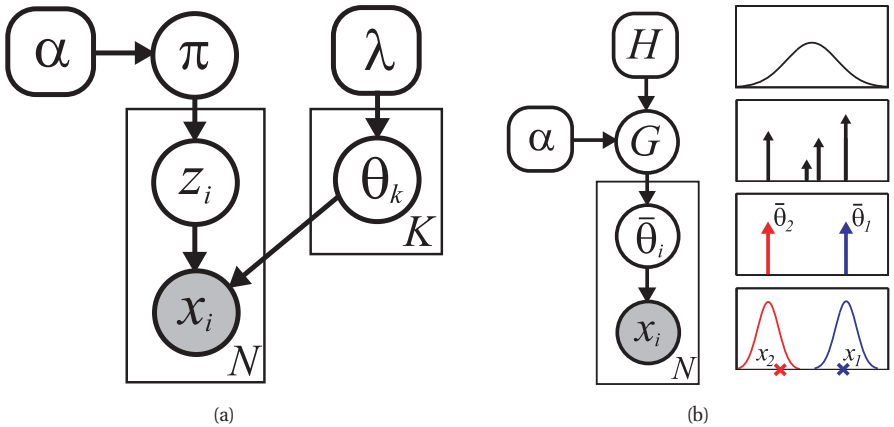


Figure 25.2 Two different representations of a finite mixture model. Left: traditional representation. Right: representation where parameters are samples from G, a discrete measure. The picture on the right illustrates the case where $K = 4$, and we sample 4 Gaussian means θ_k from a Gaussian prior $H(.|\lambda)$. The height of the spikes reflects the mixing weights π_k. This weighted sum of delta functions is G. We then generate two parameters, $\bar{\theta}_1$ and $\bar{\theta}_2$, from G, one per data point. Finally, we generate two data points, x_1 and x_2, from $\mathcal{N}(\bar{\theta}_1, \sigma^2)$ and $\mathcal{N}(\bar{\theta}_2, \sigma^2)$. 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 $\bar{\boldsymbol{\theta}}_i$ 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 $\boldsymbol{\pi} \sim \mathrm{Dir}(\frac{\alpha}{K}\mathbf{1})$, 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 $\bar{\boldsymbol{\theta}}_i$ 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 $G \sim \mathrm{DP}(\alpha, H)$, 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

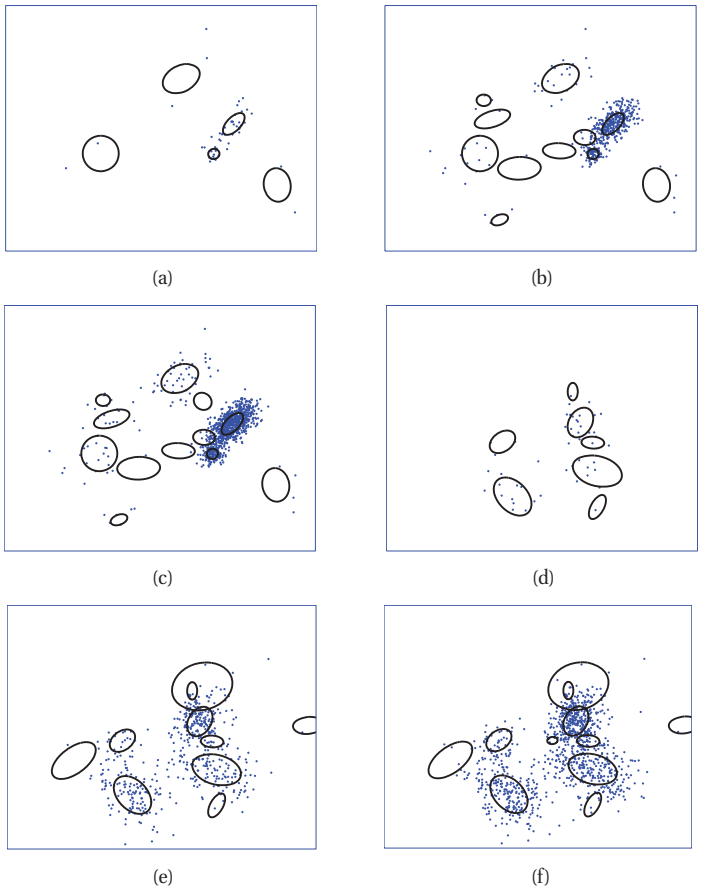


Figure 25.3 Some samples from a Dirichlet process mixture model of 2D Gaussians, with concentration parameter $\alpha = 1$. From left to right, we show $N = 50$, $N = 500$ and $N = 1000$ samples. Each row is a different run. We also show the model parameters as ellipses, which are sampled from a vague 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.

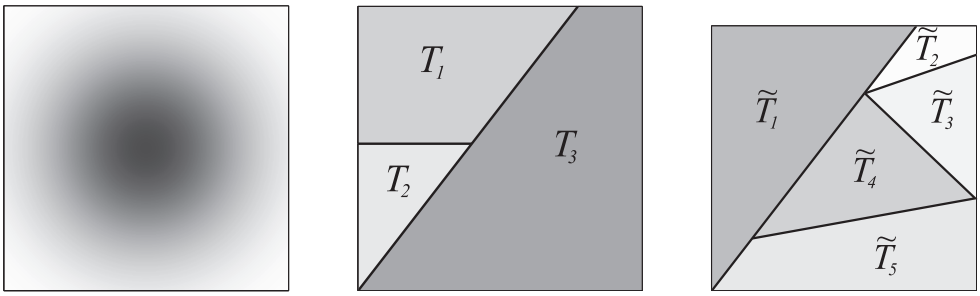


Figure 25.4 (a) A base measure H on a 2d space Θ. (b) One possible partition into $K = 3$ regions, where the shading of cell T_k is proportional to $\mathbb{E}[G(T_k)] = H(T_k)$. (c) A refined partition into $K = 5$ regions. Source: Figure 2.21 of (Sudderth 2006). Used with kind permission of Erik Sudderth.

25.2.2 The Dirichlet process

Recall from Chapter 15 that a Gaussian process is a distribution over functions of the form $f : \mathcal{X} \to R$. 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 $f \sim \mathrm{GP}(\mu(), K())$. 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 $G : \Theta \to \mathbb{R}^+$, 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

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

for any finite partition $(T_1, \ldots, T_K)$ of Θ. If this is the case, we write $G \sim \mathrm{DP}(\alpha, H)$, 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:

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

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

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

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.

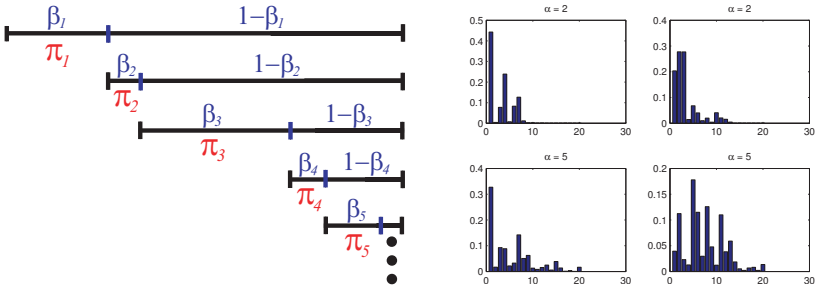


Figure 25.5 Illustration of the stick breaking construction. (a) We have a unit length stick, which we break at a random point β_1; the length of the piece we keep is called π_1; we then recursively break off pieces of the remaining stick, to generate $\pi_2, \pi_3, \ldots$. Source: Figure 2.22 of (Sudderth 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

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

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

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 $G \sim \text{DP}(\alpha, H)$, 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),

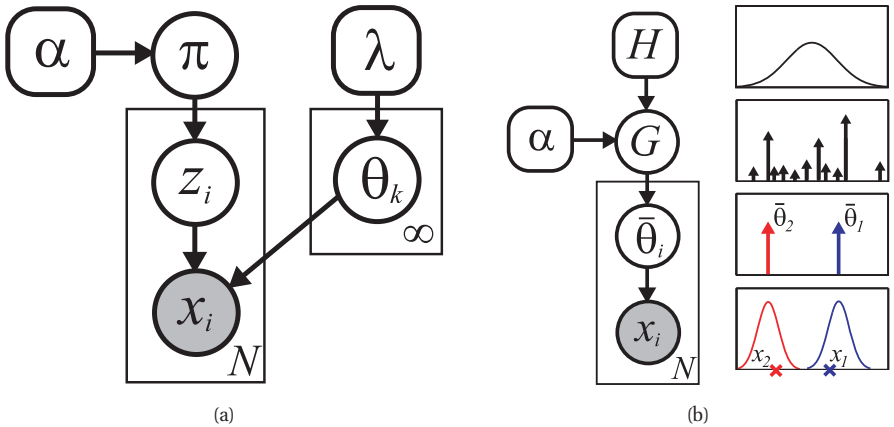


Figure 25.6 Two views of a DP mixture model. Left: infinite number of clusters parameters, $\boldsymbol{\theta}_k$, and $\pi \sim \text{GEM}(\alpha)$. Right: G is drawn from a DP. Compare to Figure 25.2. Source: Figure 2.24 of (Sudderth 2006). Used with kind permission of Erik Sudderth.

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:

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

$$z_i \sim \boldsymbol{\pi} \tag{25.30}$$

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

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

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

for *each* $i = 1 : N$ *in random order* **do**
  Remove $\mathbf{x}_i$'s sufficient statistics from old cluster z_i ;
  for *each* $k = 1 : K$ **do**
    Compute $p_k(\mathbf{x}_i) = p(\mathbf{x}_i|\mathbf{x}_{-i}(k))$;
    Set $N_{k,-i} = \dim(\mathbf{x}_{-i}(k))$;
    Compute $p(z_i = k|\mathbf{z}_{-i}, \mathcal{D}) = \frac{N_{k,-i}}{\alpha+N-1}$;
  Compute $p_*(\mathbf{x}_i) = p(\mathbf{x}_i|\boldsymbol{\lambda})$;
  Compute $p(z_i = *|\mathbf{z}_{-i}, \mathcal{D}) = \frac{\alpha}{\alpha+N-1}$;
  Normalize $p(z_i|\cdot)$;
  Sample $z_i \sim p(z_i|\cdot)$;
  Add $\mathbf{x}_i$'s sufficient statistics to new cluster z_i ;
  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 $\mathrm{Ga}(a, b)$ 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.

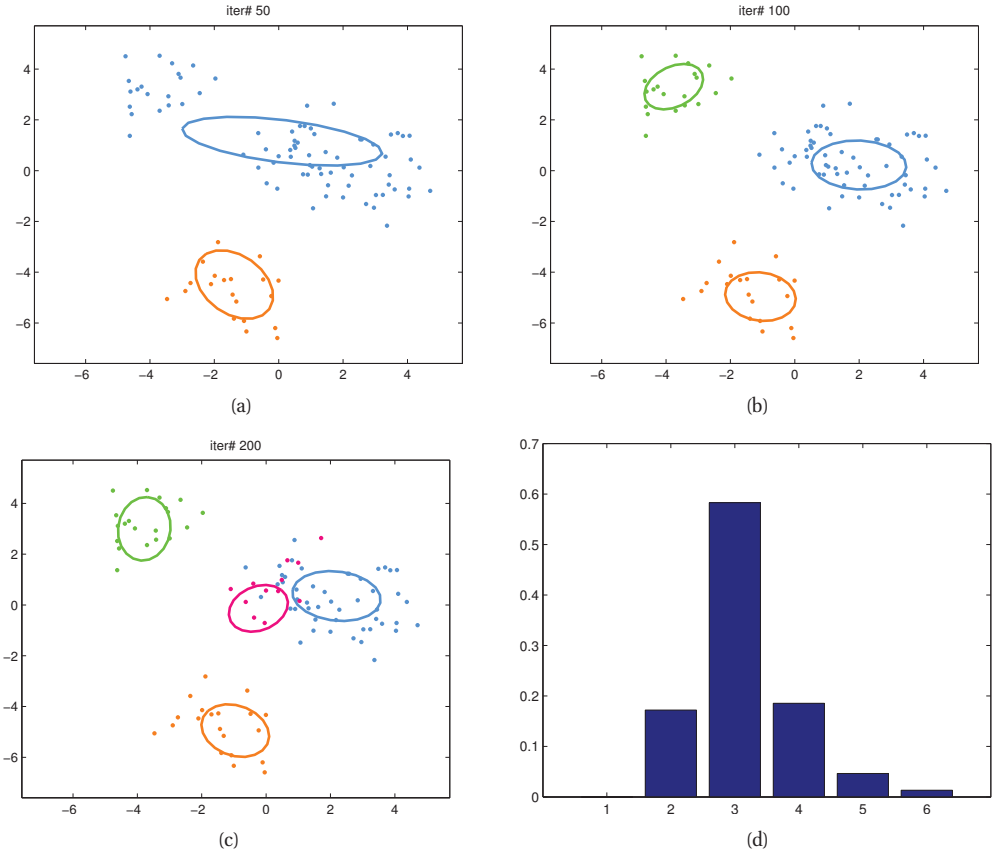


Figure 25.7 100 data points in 2d are clustered using a DP mixture fit with collapsed Gibbs sampling. We show samples from the posterior after 50,100, 200 samples. We also show the posterior over K, based on 200 samples, discarding the first 50 as burnin. Figure generated by `dpmGauss2dDemo`, written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

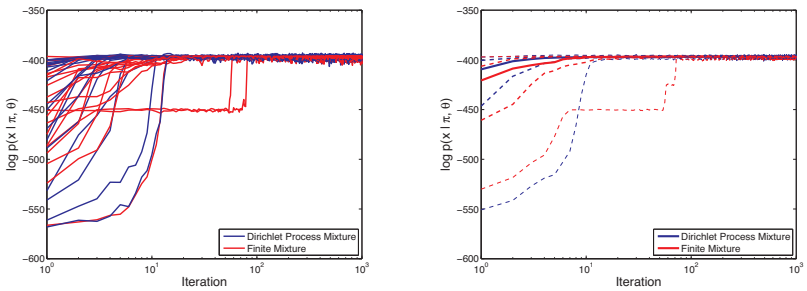


Figure 25.8 Comparison of collapsed Gibbs samplers for a DP mixture (dark blue) and a finite mixture (light red) with $K = 4$ applied to $N = 300$ data points (shown in Figure 25.7). Left: logprob vs iteration for 20 different starting values. Right: median (thick line) and quantiles (dashed lines) over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

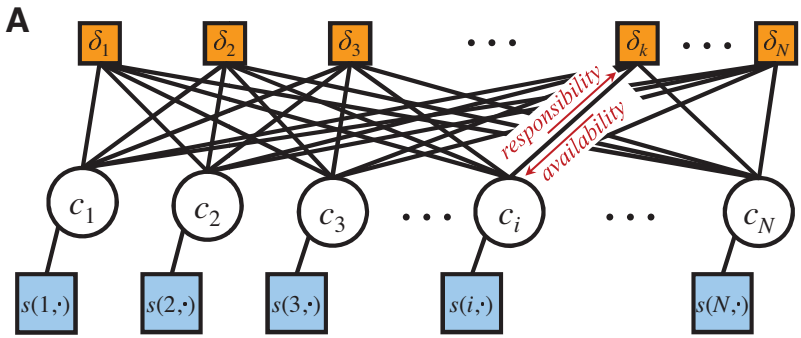


Figure 25.9 Factor graphs for affinity propagation. Circles are variables, squares are factors. Each c_i node has N possible states. From Figure S2 of (Frey and Dueck 2007). Used with kind permission of Brendan Frey.

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.

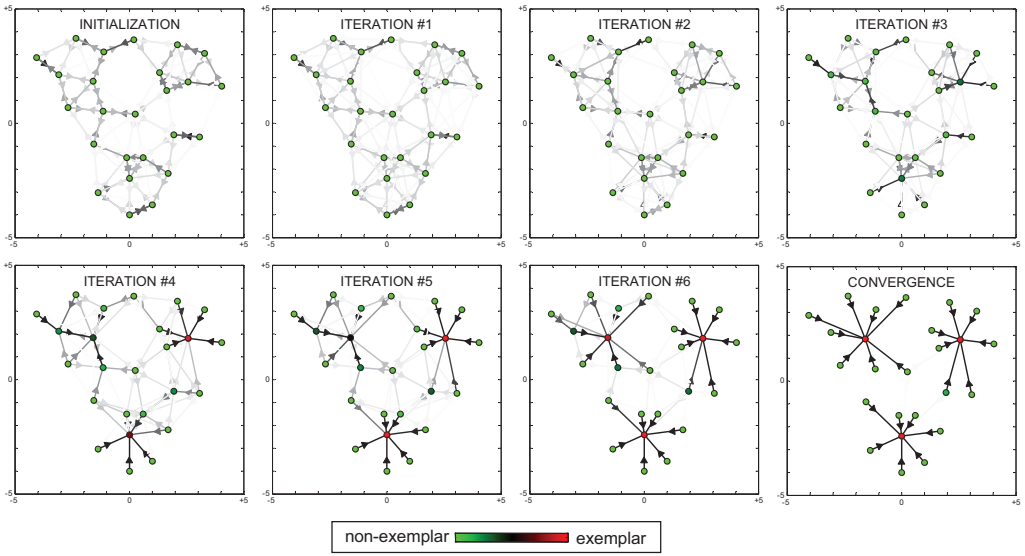


Figure 25.10 Example of affinity propagation. Each point is colored coded by how much it wants to be an exemplar (red is the most, green is the least). This can be computed by summing up all the incoming availability messages and the self-similarity term. The darkness of the $i \to k$ arrow reflects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

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 $\mathbf{L}\mathbf{u} = \lambda\mathbf{D}\mathbf{U}$.)

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 $p_{ij} = w_{ij}/d_i$

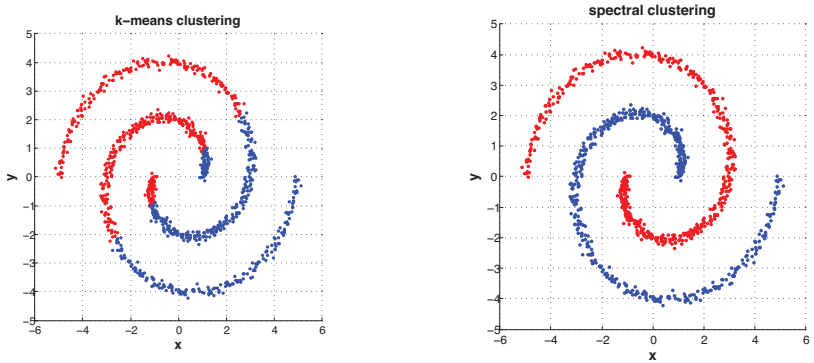


Figure 25.11 Clustering data consisting of 2 spirals. (a) K-means. (b) Spectral clustering. Figure generated by `spectralClusteringDemo`, written by Wei-Lwun Lu.

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

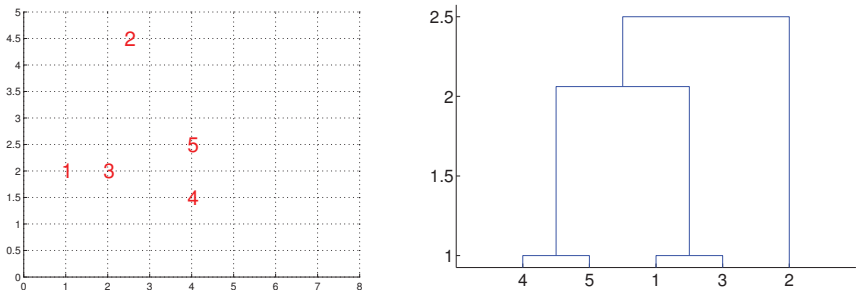
Figure 25.11 illustrates the method in action. In Figure 25.11(a), we see that K-means does a poor job of clustering, since it implicitly assumes each cluster corresponds to a spherical Gaussian. Next we try spectral clustering. We define a similarity matrix using the Gaussian kernel. We compute the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

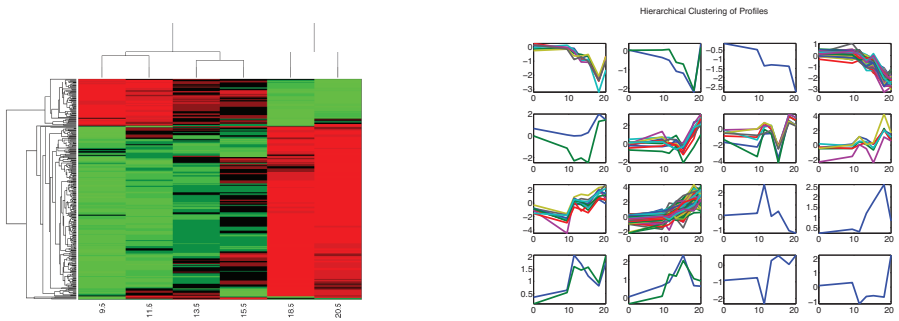
Mixture models, whether finite or infinite, produce a "flat" clustering. Often we want to learn a **hierarchical clustering**, where clusters can be nested inside each other.

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

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

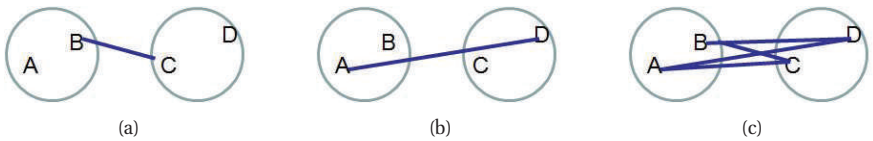


Figure 25.14 Illustration of (a) Single linkage. (b) Complete linkage. (c) Average linkage.

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

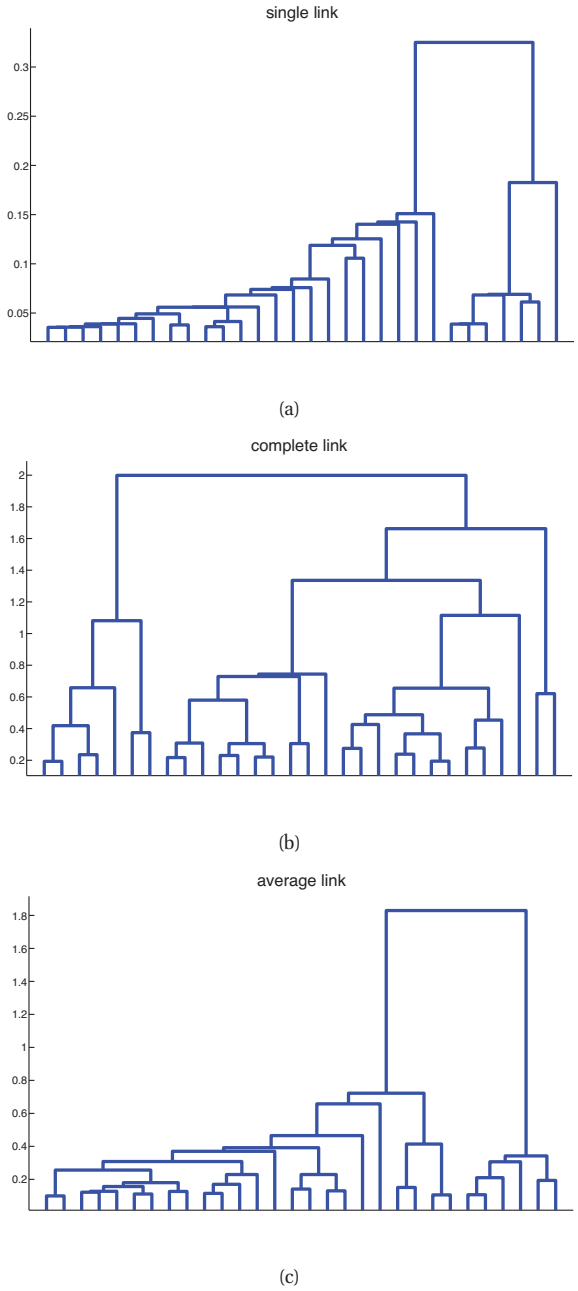


Figure 25.15 Hierarchical clustering of yeast gene expression data. (a) Single linkage. (b) Complete linkage. (c) Average linkage. Figure generated by `hclustYeastDemo`.

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

Initialize $\mathcal{D}_i = \{\mathbf{x}_i\}$, $i = 1 : N$;
Compute $p(\mathcal{D}_i|T_i)$, $i = 1 : N$;
repeat
  for *each pair of clusters* i, j **do**
    Compute $p(\mathcal{D}_{ij}|T_{ij})$
  Find the pair $\mathcal{D}_i$ and $\mathcal{D}_j$ with highest merge probability r_{ij};
  Merge $\mathcal{D}_k := \mathcal{D}_i \cup \mathcal{D}_j$;
  Delete $\mathcal{D}_i$, $\mathcal{D}_j$;
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	$\mathbf{0.828 \pm 0.025}$
Newsgroups	0.275 ± 0.001	0.315 ± 0.008	0.282 ± 0.002	$\mathbf{0.465 \pm 0.016}$
Spambase	0.598 ± 0.017	0.699 ± 0.017	0.668 ± 0.019	$\mathbf{0.728 \pm 0.029}$
Digits	0.224 ± 0.004	0.299 ± 0.006	0.342 ± 0.005	$\mathbf{0.393 \pm 0.015}$
Fglass	0.478 ± 0.009	0.476 ± 0.009	$\mathbf{0.491 \pm 0.009}$	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.

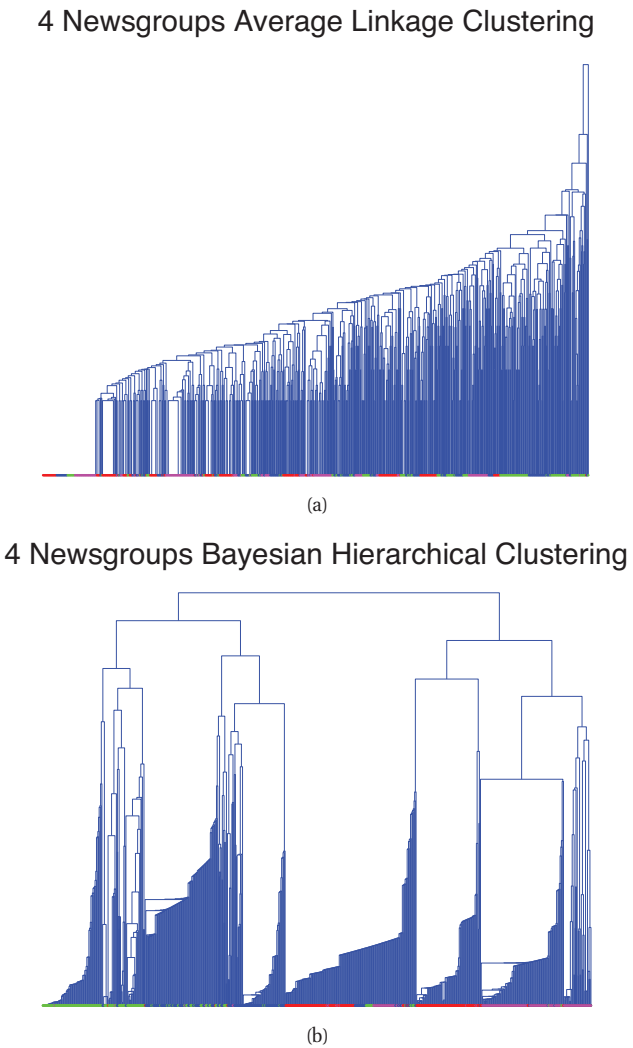


Figure 25.16 Hierarchical clustering applied to 800 documents from 4 newsgroups (red is rec.autos, blue is rec.sport.baseball, green is rec.sport.hockey, and magenta is sci.space). Top: average linkage hierarchical clustering. Bottom: Bayesian hierarchical clustering. Each of the leaves is labeled with 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.

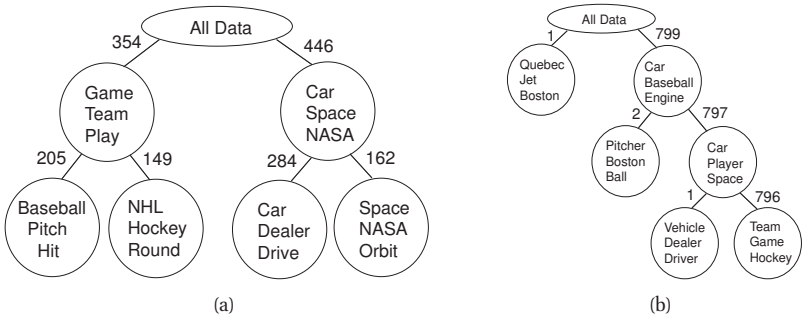


Figure 25.17 Zoom-in on the top nodes in the trees of Figure 25.16. (a) Bayesian method. (b) Average linkage. We show the 3 most probable words per cluster. The number of documents at each cluster is also given. Source: Figure 5 of (Heller and Ghahramani 2005). Used with kind permission of Katherine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

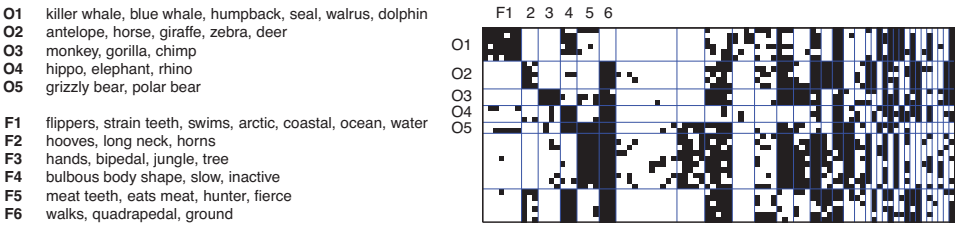


Figure 25.18 Illustration of biclustering . We show 5 of the 12 animal clusters, and 6 of the 33 feature clusters. The original data matrix is shown, partitioned according to the discovered clusters. From Figure 3 of (Kemp et al. 2006). Used with kind permission of Charles Kemp.

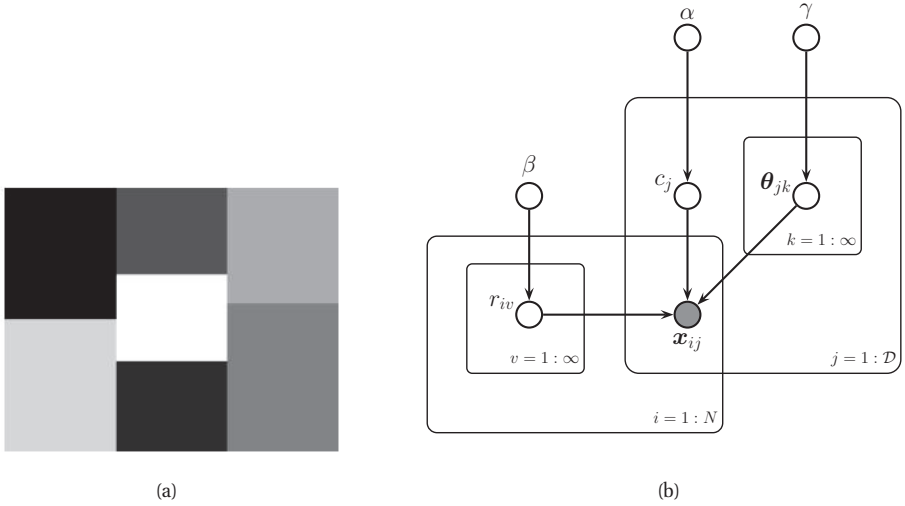


Figure 25.19 (a) Illustration of multi-view clustering. Here we have 3 views (column partitions). In the first view, we have 2 clusters (row partitions). In the second view, we have 3 clusters. In the third view, we have 2 clusters. The number of views and partitions are inferred from data. Rows within 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:

$$p(\mathbf{c}, \mathbf{r}, \mathcal{D}) = p(\mathbf{c})p(\mathbf{r}|\mathbf{c})p(\mathcal{D}|\mathbf{r}, \mathbf{c}) \tag{25.63}$$

$$p(\mathbf{c}) = \text{DP}(\mathbf{c}|\alpha) \tag{25.64}$$

$$p(\mathbf{r}|\mathbf{c}) = \prod_{v=1}^{V(\mathbf{c})} \text{DP}(\mathbf{r}_v|\beta) \tag{25.65}$$

$$p(\mathcal{D}|\mathbf{r}, \mathbf{c}, \boldsymbol{\theta}) = \prod_{v=1}^{V(\mathbf{c})} \prod_{j:c_j=v} \left[\prod_{k=1}^{K(\mathbf{r}_v)} \int \prod_{i:r_{iv}=k} p(x_{ij}|\boldsymbol{\theta}_{jk}) p(\boldsymbol{\theta}_{jk}) d\boldsymbol{\theta}_{jk} \right] \tag{25.66}$$

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.

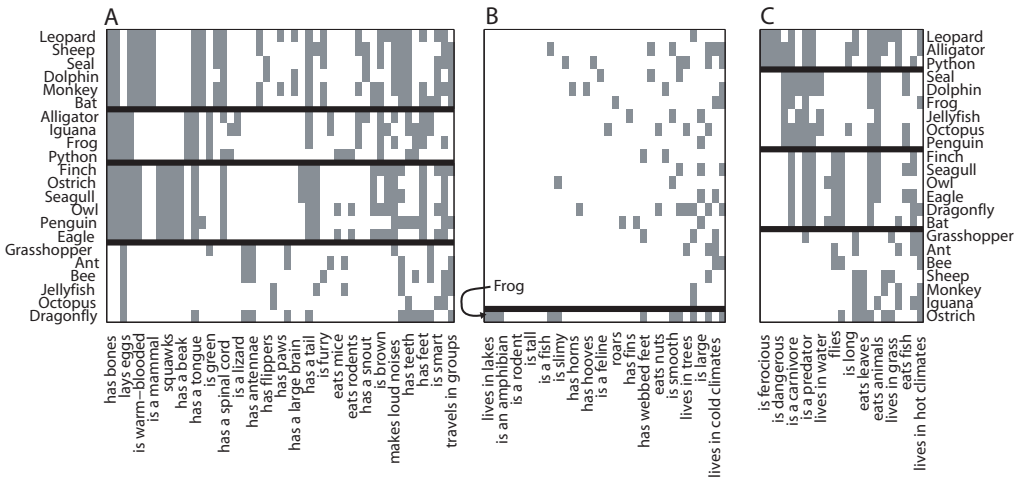


Figure 25.20 MAP estimate produced by the crosscat system when applied to a binary data matrix of animals (rows) by features (columns). See text for details. Source: Figure 7 of (Shafto et al. 2006) . Used with kind permission of Vikash Mansingkha.

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).)