Chapter 2

Preliminaries

2.1 The Joint Distribution

Let $\mathcal{X} = \{X_1, \ldots, X_N\}$ be a set of random variables. In this chapter we will assume that these variables are discrete, though most of the definitions and proofs that will be covered here generalize to continuous variables as well. Hence, we assume that each variable X_i has a discrete domain $Val(X_i)$, and can only take on values from this domain. We extend the meaning of $Val(\cdot)$ for sets of variables $\mathbf{X} \subseteq \mathcal{X}$, so it returns the set product of the domains of all variables in the set:

$$Val(\mathbf{X}) = \prod_{X \in \mathbf{X}} Val(X)$$

The *joint distribution* $P(X_1, \ldots, X_N)$ over the variables in $\mathcal{X}$ defines the probability for each possible outcome $\mathbf{x} = (x_1, \ldots, x_N) \in Val(\mathcal{X})$. It can thus be written as a table, in which the rows list the outcomes along with their assigned probabilities. If (for simplicity) $|Val(X_i)| = K$ for each i, then there are K^N such outcomes, and each one gets a number between 0 and 1. These numbers have to sum to 1, and hence the number of free parameters that is needed to encode the joint distribution in this case is $K^N - 1$.

The *marginal distribution* over any subset of variables $\mathbf{X} \subseteq \mathcal{X}$, can be obtained from the joint distribution of $\mathcal{X}$ by collapsing all sets of rows in the table that agree

on the same assignment to $\mathbf{X}$ to a single row, setting the probability for that row to the sum over the probabilities in the collapsed rows. More formally we define:

Definition 2.1.1: The *marginal distribution* over $\mathbf{X} \subseteq \mathcal{X}$ is

$$P(\mathbf{X}) = \sum_{z \in Val(\mathcal{X} \setminus \mathbf{X})} P(X, z) \tag{2.1}$$

∎

For example, if $\mathbf{X} = \{X_1, X_2\}$ and $N = 4$ then:

$$P(X_1, X_2) = \sum_{x_3 \in Val(X_3)} \sum_{x_4 \in Val(X_4)} P(X_1, X_2, x_3, x_4) \tag{2.2}$$

In this case we say that we are "marginalizing over X_3, X_4".

Here we want to note the relation between the variables X_i and the possible assignments to them $x_i \in Val(X_i)$. The joint probability is defined over the particular assignments, however **whenever an equation states only the capital letter of a variable, it means that the equation holds for all assignments to the variable**. If we want to write an equation referring to a particular assignment we will use the lowercase letters, or explicitly write the assignment within the equation, for example $P(X_i = v \mid X_j = u)$. Going back to Eq. (2.2), this is really a family of equations, one for each assignment $(x_1, x_2) \in Val(\{X_1, X_2\})$.

Another form of notation for marginalization, writes down as the summand the variables over which we are marginalizing, hence Eq. (2.1) can be written as:

$$P(\mathbf{X}) = \sum_{\mathcal{X} \setminus \mathbf{X}} P(\mathcal{X}) \tag{2.3}$$

The *conditional distribution* $P(\mathbf{Y}|\mathbf{X})$ ("$\mathbf{Y}$ given $\mathbf{X}$") for $\mathbf{X}, \mathbf{Y} \subseteq \mathcal{X}$ is actually a **family of distributions, one for each possible assignment $\mathbf{x} \in \boldsymbol{Val}(\mathbf{X})$**. To obtain $P(\mathbf{Y}|\mathbf{X} = \mathbf{x})$ one needs to remove from the table of the joint distribution over $\mathcal{X}$ all the rows that do not agree with $\mathbf{x}$. Then, normalize the numbers in the remaining rows so that they sum to 1, obtaining a distribution over $\mathcal{X} \setminus \mathbf{X}$, and then

report the marginal of that distribution over $\mathbf{Y}$. Formally it is defined as:

$$P(\mathbf{Y}|\mathbf{X} = \mathbf{x}) = \frac{P(\mathbf{Y}, \mathbf{X} = \mathbf{x})}{P(\mathbf{X} = \mathbf{x})} \tag{2.4}$$

where both $P(\mathbf{Y}, \mathbf{X} = \mathbf{x})$ and $P(\mathbf{X} = \mathbf{x})$ can be obtained from the marginals over $P(\mathcal{X})$. Since the above definition holds for every $\mathbf{x}$, we can simply define:

Definition 2.1.2: The *conditional distribution* $P(\mathbf{Y}|\mathbf{X})$ for $\mathbf{X}, \mathbf{Y} \subseteq \mathcal{X}$ is given by

$$P(\mathbf{Y}|\mathbf{X}) = \frac{P(\mathbf{Y}, \mathbf{X})}{P(\mathbf{X})} \tag{2.5}$$

∎

Corollary 2.1.3: The conditional distribution over $\mathbf{Y}$ given $\mathbf{X} = \mathbf{x}$ is proportional to the joint distribution.

$$P(\mathbf{Y}|\mathbf{X} = \mathbf{x}) \propto P(\mathbf{Y}, \mathbf{x}) \tag{2.6}$$

The Chain Rule

An alternative way to represent the joint distribution is as a product of conditional distributions, using the *chain rule*:

$$P(X_1, \ldots, X_N) = P(X_1)P(X_2 \mid X_1) \ldots P(X_N \mid X_{N-1} \ldots X_1) \tag{2.7}$$

(any order of the variables works). Under this representation, instead of having one table for the entire joint distribution, we represent each *conditional probability distribution* (CPD) separately. Assuming again that $Val(X_i) = K$ for all i, we need:

- one table with $K - 1$ free parameters to represent the CPD of $P(X_1)$.
- K tables, each with $K - 1$ parameters, for the CPD of $P(X_2 \mid X_1)$.
- K^2 tables, each with $K - 1$ parameters, for the CPD of $P(X_3 \mid X_2, X_1)$.
  ⋮
- K^{N-1} tables, each with $K-1$ parameters, for the CPD of $P(X_N \mid X_{N-1}, \ldots, X_1)$.

which is a total of $(K-1)(1+K+\ldots+K^{N-1}) = K^N - 1$ free parameters, and it is not surprising of course that this is exactly the same number of parameters used by the original "big-table" representation.

The number of parameters over the joint distribution grows then exponentially with the number of random variables. This explosion of parameters makes it hard to store (in memory) distributions with many variables. In addition, even when we can store such distributions, computing the marginals and the conditional distributions over a small subset of variables (which is often what we are interested in) requires summing over all assignments to the other variables, which again come in an exponential number.

An even bigger concern is the problem of over-fitting. Because one of the main goals of learning a probability distribution is to use it to generalize to unseen data, we want to have a plausible model. Intuitively, if a model has many parameters, then it can "match" any training data by becoming arbitrarily complex. By having fewer parameters, we force our model to be simpler, and thus it is more likely to correctly generalize to new instances. This is a form of *regularization*, in which the goal is to "smooth out" the learned distribution in an attempt to improve it's generalization performance. Figure **??** shows an example of this phenomenon. The goal is to fit a curve to the green points, which is assumed to be generated by a polynomial function plus white noise. The red curve is a degree 10 polynomial (11 parameters), that exactly hits the green points, while the blue curve is a line (2 parameters) that only "comes close" to the target points. However, because the blue line is simpler (has fewer parameters), we are more likely to believe its prediction of the next point (the blue diamond), rather than the higher order prediction given by the red diamond.

Hence to be able to work efficiently with joint distributions, we have to reduce the number of parameters and find compact representations. One of the tools in our disposal for this purpose is conditional independence.

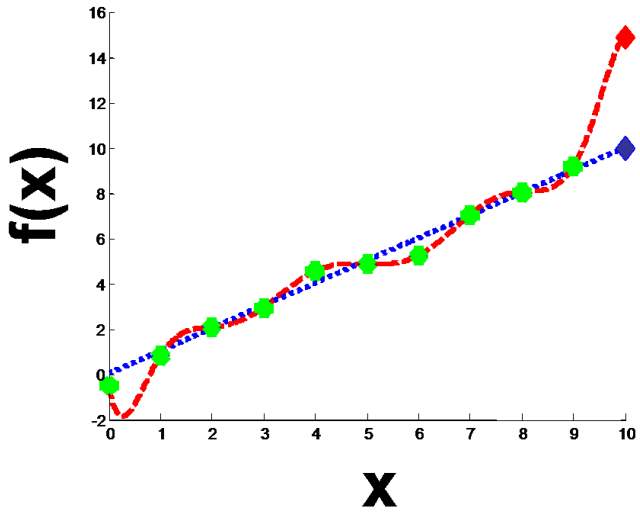


Figure 2.1: The problem of over-fitting.

2.1.1 Independence

In the chain-rule formulation above (Eq. (2.7)), we had to maintain a different table for the distribution over X_i for each assignment to the variables $X_1 \ldots, X_{i-1}$. Hence the CPD of X_i had K^{i-1} tables, which is quite a lot for large i's. Independence assumptions allow you to collapse these tables into a much smaller set, by allowing different assignments to $X_1, \ldots, X_{i-1}$ to share the same table.

Definition 2.1.4: Given three disjoint sets of variables $\mathbf{X}, \mathbf{Y}, \mathbf{Z} \subseteq \mathcal{X}$, we say that $\mathbf{X}$ and $\mathbf{Y}$ are *conditionally independent* given $\mathbf{Z}$ if and only if $P(\mathbf{X} \mid \mathbf{Y}, \mathbf{Z}) = P(\mathbf{X} \mid \mathbf{Z})$ whenever $P(\mathbf{Z}) > 0$. We denote this property by:

$$(\mathbf{X} \perp \mathbf{Y} \mid \mathbf{Z})$$

If $\mathbf{Z}$ is the empty set, we may also say that $\mathbf{X}$ and $\mathbf{Y}$ are *marginally independent*. ∎

Corollary 2.1.5: If $(\mathbf{X} \perp \mathbf{Y} \mid \mathbf{Z})$ then $(\mathbf{X} \perp \mathbf{Y}' \mid \mathbf{Z})$, for every $\mathbf{Y}' \subseteq \mathbf{Y}$.

Going back to the chain rule formulation of the joint distribution, if someone tells us (for example) that $(X_i \perp X_1, \ldots, X_{i-2} \mid X_{i-1})$, then suddenly

$$P(X_i \mid X_{i-1}, \ldots, X_1) = P(X_i \mid X_{i-1}),$$

and now we only need K tables (each of $K-1$ free parameters) for the CPD of X_i instead of K^{i-1} tables before. We just saved $(K-1)K^{i-2}$ parameters.

The main purpose of the graphs in *probabilistic graphical models* (PGM) is as a structure that encodes such conditional independencies.

2.2 Common Distributions

2.2.1 Dirichlet Distribution

Dirichlet distribution is a prior over discrete distributions over a finite dimension.

Definition 2.2.1: Let $\beta = (\beta_1, \ldots, \beta_K)$, such that $\sum_k \beta_k = 1$ and $\beta_k \geq 0$ for all k. We say that $\beta \sim \text{Dirichlet}(\alpha_1, \ldots, \alpha_K)$, if

$$P(\beta) = \frac{1}{Z(\alpha_1, \ldots, \alpha_K)} \prod_k \beta_k^{\alpha_k - 1}.$$

∎

It turns out that there is a closed form formula for the normalization constant in this distribution, that will become very useful later on:

$$Z(\alpha_1, \ldots, \alpha_K) = \int_\beta \prod_{k=1}^{K} \beta_k^{\alpha_k - 1} d\beta = \frac{\prod_{k=1}^{K} \Gamma(\alpha_k)}{\Gamma(\sum_{k=1}^{K} \alpha_k)} \tag{2.8}$$

where the integral is taken over the simplex where $\sum_k \beta_k = 1$ and $\beta_k \geq 0$ for all k.

Corollary 2.2.2: If $\beta \sim \text{Dirichlet}(\alpha_1, \ldots, \alpha_K)$ then $E[\beta_j] = \frac{\alpha_j}{\sum_{k=1}^{K} \alpha_k}$.

Proof: Without loss of generality, assume $j = 1$:

$$\begin{aligned}
E[\beta_1] =& \frac{1}{Z(\alpha_1, \ldots, \alpha_K)} \int_\beta \beta_1 P(\beta) d\beta \\
=& \frac{1}{Z(\alpha_1, \ldots, \alpha_K)} \int_\beta \beta_1 \prod_{k=1}^{K} \beta_k^{\alpha_k - 1} d\beta \\
=& \frac{Z(\alpha_1 + 1, \alpha_2, \ldots, \alpha_K)}{Z(\alpha_1, \alpha_2, \ldots, \alpha_K)} \\
=& \frac{\Gamma(\alpha_1 + 1)}{\Gamma(\alpha_1)} \cdot \frac{\Gamma(\sum_k \alpha_k)}{\Gamma(1 + \sum_k \alpha_k)} \\
=& \frac{\alpha_1}{\sum_k \alpha_k} \blacksquare
\end{aligned}$$

Conjugacy Between The Dirichlet And Multinomial Distributions

Consider the case we draw samples $\mathbf{C} = (C_1 \ldots, C_N)$ from the discrete distribution β generated by a Dirichlet prior. The math in this case turns out to beautifully simplify the expressions for $P(\Theta \mid \mathbf{C})$ and $P(\mathbf{C})$.

Let

$$\begin{aligned}
\beta &\sim \text{Dirichlet}\,(\alpha_1, \ldots, \alpha_K) \\
C_i &\sim \text{multinomial}(\beta) \quad \forall i = 1 \ldots N
\end{aligned}$$

Proposition 2.2.3: Under the above model:

$$\beta \mid C_1, \ldots C_N \sim \text{Dirichlet}\,(\alpha_1 + N_1, \ldots, \alpha_K + N_K)$$

where $N_k = \sum_i \mathbb{1}\{C_i = k\}$.

Proof:

$$\begin{aligned}
P(\beta \mid \mathbf{C}) \propto& P(\beta, \mathbf{C}) = P(\mathbf{C} \mid \beta) P(\beta) \propto \prod_{k=1}^{K} \beta_k^{N_k} \cdot \prod_{k=1}^{K} \beta_k^{\alpha_k - 1} = \prod_{k=1}^{K} \beta_k^{\alpha_k + N_k - 1} \\
\propto& \text{Dirichlet}(\beta; \alpha_1 + N_1, \ldots, \alpha_K + N_K) \blacksquare
\end{aligned}$$

This result represents the *conjugacy* of the Dirichlet distribution and the multinomial distribution. Both the prior over β and the posterior over β, after observing any amount of samples, are members of the Dirichlet distribution over K dimensions. The conjugacy also allows us to easily integrate-out β when we compute the likelihood of the data:

Corollary 2.2.4:

$$P(C_i = j \mid \mathbf{C}_{-i}) = \frac{\alpha_j + N_{-i,j}}{N - 1 + \sum_{k=1}^{K} \alpha_k}$$

where $N_{-i,j} = \sum_{i':i' \neq i} \mathbb{1}\{C_{i'} = j\}$ and $\mathbf{C}_{-i} = \mathbf{C} \setminus \{C_i\}$.

Proof:

$$\begin{aligned} P(C_i = j \mid \mathbf{C}_{-i}) &= \int P(C_i = j \mid \beta) P(\beta \mid \mathbf{C}_{-i}) d\beta \\ &= \int \beta_j \cdot \text{Dirichlet}(\beta; \alpha_1 + N_{-i,1}, \ldots, \alpha_K + N_{-i,K}) d\beta \end{aligned}$$

The last term is exactly $E[\beta_j]$ under $\beta \sim \text{Dirichlet}(\alpha_1 + N_{-i,1}, \ldots, \alpha_K + N_{-i,K})$, which by Corollary 2.2.2 is QED. ∎

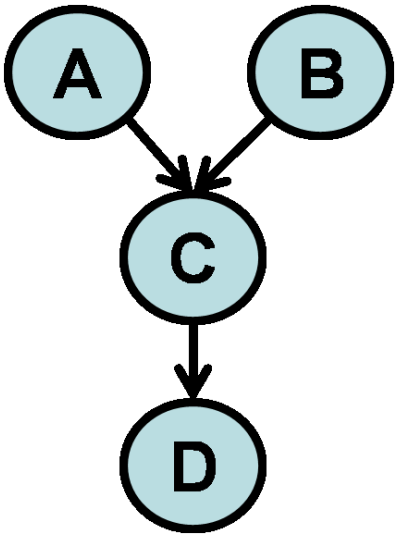
Using this result we can sample $\mathbf{C}$ without sampling β, by sampling the C_i variables one after the other using the chain rule:

$$P(\mathbf{C}) = P(C_1) P(C_2 \mid C_1) \cdots P(C_N \mid C_1, \ldots, C_{N-1}) \tag{2.9}$$

The likelihood of the resulting assignment is

$$P(\mathbf{C}) = \frac{\Gamma(\sum_k \alpha_k) \prod_k \Gamma(\alpha_k + N_k)}{\Gamma(N + \sum_k \alpha_k) \prod_k \Gamma(\alpha_k)} \tag{2.10}$$

and notice that it does not depend on the order of the variables.



(a)

A	P(A)
0	0.7
1	0.3

B	P(B)
0	0.1
1	0.9

C	D	P(D|C)
0	0	0.3
0	1	0.7
1	0	0.98
1	1	0.02

A	B	C	P(C|A,B)
0	0	0	0.3
0	0	1	0.7
0	1	0	0.98
0	1	1	0.02
1	0	0	0.5
1	0	1	0.5
1	1	0	0.4
1	1	1	0.6

(b)

Figure 2.2: An example Bayesian network

2.3 Bayesian Networks

Perhaps the most well-known type of a probabilistic graphical model is the *Bayesian network* (BN). A BN is represented by a pair $(\mathcal{G}, \Theta)$, where $\mathcal{G}$ is a directed acyclic graph (DAG) with the nodes corresponding to the variables $\mathcal{X}$ that we wish to model. An example graph is depicted in Figure 2.2(a), modeling the four **binary** variables A, B, C, and D.

The DAG $\mathcal{G}$ visually illustrates the conditional independencies in the distribution represented by the BN. In order to describe these independencies, we must define two concepts. The *parents* of a node X_i (also written as $\mathbf{Pa}(X_i)$) in a BN are the nodes X_j that have an arrow beginning at X_j and ending at X_i. The *descendants* of a node X_i are all nodes X_j such that there is a directed path from X_i to X_j in $\mathcal{G}$. The graph structure $\mathcal{G}$ for a Bayesian network tells us that each variable X_i is conditionally independent of its non-descendants given its parents. More formally:

Definition 2.3.1: For a Bayesian network over the variables $\mathcal{X} = \{X_1, \ldots, X_N\}$, the

BN graph $\mathcal{G}$ encodes the set of *Markov independence assumptions*:

$$\forall X_i \in \mathcal{X} \quad (X_i \perp NonDesc(X_i) \mid \mathbf{Pa}(X_i))$$

∎

Definition 2.3.2: The set of variables $\mathcal{X} = \mathcal{X}_1, \ldots, \mathcal{X}_N$ is in a *topological ordering* with respect to a DAG $\mathcal{G}$, if for every i

$$\mathbf{Pa}(X_i) \subseteq \{X_1, \ldots, X_{i-1}\}.$$

∎

Corollary 2.3.3: If the order of the variables in $\mathcal{X}$ is topological with respect to $\mathcal{G}$, then:

$$P(X_i \mid X_1, \ldots, X_{i-1}) = P(X_i \mid \mathbf{Pa}(X_i))$$

This means that we can now replace each CPD in the chain-rule formulation of Eq. (2.7) with an equivalent, yet simpler, CPD:

$$P(\mathcal{X}) = \prod_i P(X_i \mid \mathbf{Pa}(X_i)) \tag{2.11}$$

In other words, the joint distribution is simply the product of CPDs, each variable conditioned only on its parents.

Corollary 2.3.4: If the order of the variables in $\mathcal{X}$ is topological with respect to $\mathcal{G}$, then for every $i = 1 \ldots N$:

$$P(X_i, \ldots, X_N \mid X_1 \ldots X_{i-1}) = \prod_{k=i}^{N} P(X_i \mid \mathbf{Pa}(X_i)) \tag{2.12}$$

The other component of a BN is the set of parameters Θ. This is simply the set of probability tables for each *conditional probability distribution* (CPD) in the above decomposition.

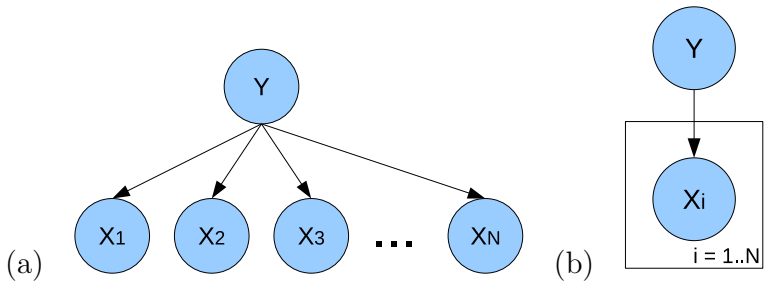


Figure 2.3: Two equivalent graph structures, the one in (b) uses a plate model.

Figure 2.2(b) shows some of the CPDs for our example network. In this example, the decomposition is $P(A, B, C, D) = P(A)P(B)P(C \mid A, B)P(D \mid C)$. In this case, representing the full joint distribution as a single table would take $2^4 - 1 = 15$ independent parameters. In our BN representation, however, we need 1 free parameter each for $P(A)$ and $P(B)$, 4 free parameters for $P(C \mid A, B)$, and 2 free parameters for $P(D \mid C)$, for a total of 8 free parameters. This more compact representation should require less data to learn accurately, and will allow many inference questions to be answered more easily.

2.3.1 Plate Models

Often times, our BN has many similar variables that share the same domain and CPD with respect to their parents. In such a case we can represent the BN graph more compactly by putting a plate around the reoccurring pattern, and simply listing how many times it repeats.

For example, if our data has N independent draws from a coin with an unknown bias, then we can model the data as a BN with a variable Y representing the probability of 'HEAD', and variables X_i for $i = 1 \ldots N$ for the result of the N draws (see Figure 2.3a). Since the X_i variables all have Y as parent, and since they all share the same CPD ('HEAD' with probability Y, 'TAIL' with probability $1 - Y$), we can represent the same graph more compactly by drawing a single X_i and putting a plate around it, as shown in Figure 2.3b).

2.3.2 Inference

We typically divide the random variables in the model to one of two kinds: observed and hidden. The *observed variables* are those we have values for, and the rest are *hidden*. *Inference* is the task of computing the probability over the hidden variables, or a subset of them, given the observed variables. More technically we might say that every time we compute a conditional probability of the type $P(\mathbf{Y} \mid \mathbf{X})$ we perform inference in our model.

In this work we use inference on a model given by a Bayesian network whose graph structure is a tree, and the observed variables are the leaves. Define the model variables $\mathcal{X} = \mathbf{X} \cup \mathbf{Y}$ where $\mathbf{X} = \{X_1, \ldots, X_M\}$ are the observed leaf variables (with values $\mathbf{x} = (x_1, \ldots, x_M)$), and $\mathbf{Y} = \{Y_0, Y_1, \ldots, Y_N\}$ are the hidden variables (Y_0 is the root). In such a case sampling $\mathbf{Y}$ from the conditional distribution $P(\mathbf{Y} \mid \mathbf{X} = \mathbf{x})$, or finding the assignment $\mathbf{y} \in \mathit{Val}(\mathbf{Y})$ that maximizes $P(\mathbf{Y} = \mathbf{y} \mid \mathbf{X} = \mathbf{x})$ (the *MAP assignment*), are both tasks that can be solved efficiently.

Notation: Let $\mathit{Desc}(Y_i) = (\mathbf{X}_d^i, \mathbf{Y}_d^i)$, where $\mathbf{X}_d^i \subseteq \mathbf{X}, \mathbf{Y}_d^i \subseteq \mathbf{Y}$, be the set of variables that are descendants of Y_i in the BN tree (including Y_i). Let $\mathit{NonDesc}(Y_i) = (\mathbf{X}_u^i, \mathbf{Y}_u^i)$ be the rest of the variables. Let $u(i)$ be the index of Y_i's parent (e.g.$\mathbf{Pa}(Y_i) = Y_{u(i)}$).

Define:

$$\phi_0(y) \triangleq \left(\prod_{j:\mathbf{Pa}(X_j)=Y_0} P(x_j \mid \mathbf{Pa}(X_j) = y) \right) P(Y_i = y) \tag{2.13}$$

$$\phi_i(y, y') \triangleq \left(\prod_{j:\mathbf{Pa}(X_j)=Y_i} P(x_j \mid \mathbf{Pa}(X_j) = y) \right) P(Y_i = y \mid \mathbf{Pa}(Y_i) = y') \quad [\forall i = 1 \ldots N] \tag{2.14}$$

Proposition 2.3.5:

$$\prod_{k:Y_k \in \mathbf{Y}_d^i} \phi_k(Y_k, Y_{u(k)}) = P(\mathbf{Y}_d^i, \mathbf{x}_d^i \mid Y_{u(i)}) \tag{2.15}$$

Proof: Consider the topological ordering that puts all variables in $\mathit{NonDesc}(Y_i)$

before $Desc(Y_i)$. By Eq. (2.12), we have that

$$P(\mathbf{Y}_d^i, \mathbf{x}_d^i \mid \mathbf{Y}_u^i, \mathbf{x}_u^i) = \prod_{k:Y_k \in \mathbf{Y}_d^i} \left(\underbrace{P(Y_k \mid \mathbf{Pa}(Y_k)) \prod_{j:\mathbf{Pa}(X_j)=Y_k} P(x_j \mid \mathbf{Pa}(X_j))}_{\phi_k(Y_k, Y_{u(k)})} \right) \tag{2.16}$$

$$= \prod_{k:Y_k \in \mathbf{Y}_d^i} \tag{2.17}$$

We notice that the only variable from $NonDesc(Y_i)$ that appears on the RHS of Eq. (2.16) is $Y_{u(i)}$ (this is true because our BN is a tree), and hence:

$$P(\mathbf{Y}_d^i, \mathbf{x}_d^i \mid \mathbf{Y}_u^i, \mathbf{x}_u^i) = P(\mathbf{Y}_d^i, \mathbf{x}_d^i \mid Y_{u(i)}) \tag{2.18}$$

∎

A standard message-passing algorithm is used to pass messages from the bottom towards the root. Each hidden node Y_i ($i = 1 \dots N$) sends a message $m_{i \to u(i)}(y_{u(i)})$ to its parent, defined recursively as follows. Let i be a node in the tree with a parent p and K children $v_1 \dots v_K$. Let $m_{v_k \to i}(y_i)$ be the messages sent to i from its children. Then:

$$\tau_{i \to p}(y_i, y_p) \triangleq \phi_i(y_i, y_p) \prod_{k=1}^{K} m_{v_k \to i}(y_i) \tag{2.19}$$

$$m_{i \to p}(y_p) \triangleq \sum_{y_i} \tau_{i \to p}(y_i, y_p) \tag{2.20}$$

Proposition 2.3.6: (easily shown via induction)

$$\tau_{i \to p}(Y_i, Y_p) = \sum_{\mathbf{Y}_d^i \setminus Y_i} \prod_{k:Y_k \in \mathbf{Y}_d^i} \phi_k(Y_k, Y_{u(k)}) \tag{2.21}$$

Corollary 2.3.7:

$$\tau_{i \to p}(Y_i, Y_p) = \sum_{\mathbf{Y}_d^i \setminus Y_i} P(\mathbf{Y}_d^i, \mathbf{x}_d^i \mid Y_p) = P(Y_i, \mathbf{x}_d^i \mid Y_p) \tag{2.22}$$

Proposition 2.3.8:

$$P(Y_i \mid Y_p, \mathbf{x}) = \frac{\tau_{i\to p}(Y_i, Y_p)}{m_{i\to p}(Y_p)} \tag{2.23}$$

Proof:

$$P(Y_i \mid Y_p, \mathbf{x}) = P(Y_i \mid Y_p, \mathbf{x}_d^i) \propto P(Y_i, \mathbf{x}_d^i \mid Y_p) = \tau_{i\to p}(Y_i, Y_p) \tag{2.24}$$

And hence:

$$P(Y_i \mid Y_p, \mathbf{x}) = \frac{\tau_{i\to p}(Y_i, Y_p)}{\sum_{Y_i} \tau_{i\to p}(Y_i, Y_p)} = \frac{\tau_{i\to p}(Y_i, Y_p)}{m_{i\to p}(Y_p)} \tag{2.25}$$

∎

As for the root itself, we compute the message from the root τ_0 like we would for any other node, except we use the unary factor ϕ_0:

$$\tau_0(y_0) \triangleq \phi_0(y_0) \prod_{k=1}^{K} m_{v_k\to i}(y_0)$$

and the same proof above shows that $\tau_0(y_0) \propto P(Y_0 \mid \mathbf{x})$.

Once all the messages have been delivered, we obtain through our τ messages a factorization of $P(\mathbf{Y}|\mathbf{x})$:

$$P(\mathbf{Y}|\mathbf{x}) = \prod_{i=0}^{N} P(Y_i \mid Y_0 \dots Y_{i-1}, \mathbf{x}) \tag{2.26}$$

$$= \prod_{i=0}^{N} P(Y_i \mid \mathbf{Pa}(Y_i), \mathbf{x}) \tag{2.27}$$

$$= \tau_0(Y_0) \prod_{i=1}^{N} \tau_{i\to u(i)}(Y_i, Y_{u(i))}) \tag{2.28}$$

where $\tau_0(y_0) \triangleq \phi_0(y_0) \prod_{k=1}^{K} m_{v_k\to i}(y_0)$ and the same proof above shows that $\tau_0(y_0) \propto P(Y_0 \mid \mathbf{x})$.

The second equality holds for a topological ordering, because the parent of each Y_i is included in its conditioned set, so the BN independencies allow us to remove from this set all the other Y_i's (all of which necessarily non-descendants). We can now quickly compute $P(\mathbf{Y} = \mathbf{y} \mid \mathbf{X} = \mathbf{x})$ for every assignment $\mathbf{y} \in \mathbf{Y}$. Moreover, we

can use the messages and *Eq.* (2.23) to **sample** the Y_i variables one by one from top to bottom, starting from the root Y_0.

2.3.3 Finding The MAP Assignment

If we replace the summation in Eq. (2.20) with a maximization, the **exact same inference algorithm** can be used to decode the MAP assignment to the $\mathbf{Y}$ variables (given $\mathbf{X} = \mathbf{x}$), which we will denote as $\tilde{\mathbf{y}} = (\tilde{y}_0, \tilde{y}_1, \ldots, \tilde{y}_N)$:

$$\tilde{\mathbf{y}} = \arg\max_{\mathbf{y}} P(\mathbf{y} \mid \mathbf{x})$$

In this case, the analogous derivation of Eq. (2.22) will show that

$$\tau_{i \to p}(y_i, y_p) = \max_{\mathbf{z} \in Val(\mathbf{Y}_d^i \setminus Y_i)} P(\mathbf{z}, y_i, \mathbf{x}_d^i \mid Y_p = y_p) \tag{2.29}$$

$$\propto \max_{\substack{\mathbf{z} \in Val(\mathbf{Y}_d^i \setminus Y_i) \\ \mathbf{w} \in Val(\mathbf{Y}_u^i \setminus Y_p)}} P(\mathbf{z}, \mathbf{w}, y_i, y_p \mid \mathbf{x}) \tag{2.30}$$

And for the root:

$$\tau_0(y_0) \propto \max_{\mathbf{z} \in Val(\mathbf{Y} \setminus Y_0)} P(\mathbf{z}, y_0 \mid \mathbf{x}) \tag{2.31}$$

we can easily see that $\tilde{y}_0 = \arg\max_{y_0} \tau_0(y_0)$. Armed with the value of $\tilde{y}_0$, we can compute the MAP assignment to the rest of the variables by walking down the tree, setting

$$\tilde{y}_i = \arg\max_{y_i} \quad \tau_{i \to p}(y_i, \tilde{y}_p) \tag{2.32}$$

2.4 Markov Chain Monte Carlo

In many cases the model we use cannot be reduced to a Bayesian Network that allows an exact inference solution like the one described in the previous section. In the general case, inference is an NP-hard problem [8]. In such cases we resort to other approximate inference methods. The method that we use heavily in this work is called Markov Chain Monte Carlo, or MCMC for short [14].

Define the model variables $\mathcal{X} = \mathbf{X} \cup \mathbf{Y}$ where $\mathbf{X}$ are the observed variables and $\mathbf{Y}$ are the hidden variables. The algorithm generates *particles*, each is a full assignment to all the hidden variables of the model. model. The algorithm starts with a random particle and iteratively starts spitting out new particles, each particle is a manipulation of the previous one. Given enough time, the algorithm is guaranteed to reach an asymptote in which it spits particles that are samples from the conditional distribution $P(\mathbf{Y} \mid \mathbf{X} = \mathbf{x})$. However, there is no way to know for sure when enough is "enough".

Assuming we waited enough time, we can use those samples to estimate various conditional probabilities. If we wanted to know the quantity $P(Y_2 = 1 \mid \mathbf{x})$, we can estimate it as the fraction of samples that have this assignment amongst all the samples that were drawn:

$$P(Y_2 = 1 \mid \mathbf{x}) \approx \frac{\sum_{m=1}^{M} \mathbb{1}\{y_2[m] = 1\}}{M}, \tag{2.33}$$

where M is the total number of samples drawn, and $y_i[m]$ is the assignment to variable Y_i in the m^{th} sample, and $\mathbb{1}\{\}$ is the "indicator" function which returns 1 when the argument is true, and 0 otherwise.

Define $\pi()$ as our target distribution:

Our target distribution is $P(\mathbf{Y} = \mathbf{y} \mid \mathbf{X} = \mathbf{x})$. Notice that we cannot compute it directly. However, we can compute a proxy $\pi(\mathbf{y})$ to this distribution, which is proportional to it:

$$\pi(\mathbf{y}) \triangleq P(\mathbf{y}, \mathbf{x}) \propto P(\mathbf{Y} = \mathbf{y} \mid \mathbf{X} = \mathbf{x})$$

The algorithm works as follows. We provide a transition probability $\mathcal{T}(\mathbf{y} \rightarrow \mathbf{y}')$ that is used to randomly generate the next particle $\mathbf{y}'$ given the current one $\mathbf{y}$. According to the *Metropolis-Hastings* framework, we should accept the transition to $\mathbf{y}'$ with probability

$$\min\left[1, \frac{\pi(\mathbf{y}')\mathcal{T}(\mathbf{y}' \rightarrow \mathbf{y})}{\pi(\mathbf{y})\mathcal{T}(\mathbf{y} \rightarrow \mathbf{y}')}\right]. \tag{2.34}$$

Definition 2.4.1: The set of *supported particles* with respect to a particle $\mathbf{y}$ and a transition $\mathcal{T}(\mathbf{y} \to \mathbf{y}')$ is those particles who get positive probability:

$$\mathcal{I}_{\mathbf{y}} = \{\mathbf{y}' \mid \mathcal{T}(\mathbf{y} \to \mathbf{y}') > 0\}$$

∎

Usually, the set $\mathcal{I}_{\mathbf{y}}$ for any transition distribution $\mathcal{T}$ is not the whole set of particles, but much smaller. Most MCMC algorithms prepare an arsenal of such distributions, often called "moves". As long as for any two particles there exist some combination of moves that can take you with a positive probability from one to the other, the distribution over the particles that the algorithm spits converges to $\pi(Y)$.

Notice that for the proposal to be accepted, we have to have a positive probability for the reverse transition as well (otherwise the acceptance probability will be zero). This kind of move is called a *reversible* move.

Definition 2.4.2: A transition $\mathcal{T}(\mathbf{y} \to \mathbf{y}')$ is reversible, if $\mathbf{y} \in \mathcal{I}_{\mathbf{y}'}$ for every $\mathbf{y}$ and $\mathbf{y}' \in \mathcal{I}_{\mathbf{y}}$. ∎

For reversible transitions, the computation of the acceptance probability becomes even easier if we set

$$\mathcal{T}(\mathbf{y} \to \mathbf{y}') = \frac{\pi(\mathbf{y}')}{Z_{\mathbf{y}}} \tag{2.35}$$

for each particle $\mathbf{y}' \in \mathcal{I}_{\mathbf{y}}$. That is, we first come up with a way to produce a small set of particles $\mathcal{I}_{\mathbf{y}}$ from any $\mathbf{y}$. Then, $\mathcal{T}$ gives each particle $\mathcal{I}_{\mathbf{y}}$ a weight proportional to its probability in $\pi()$. The quantity $Z_{\mathbf{y}} = \sum_{\mathbf{y}' \in \mathcal{I}_{\mathbf{y}}} \pi(\mathbf{y})$ normalizes those weights to sum to 1. Once we plug this into Eq. (2.34) the acceptance probability then becomes

$$\min\left[1, \frac{Z_{\mathbf{y}}}{Z_{\mathbf{y}'}}\right]. \tag{2.36}$$

Since the transition makes only a local change, the particles in $\mathcal{I}_{\mathbf{y}}$ will have many common factors that will eventually cancel out in the above computation and hence need not be computed.

If the set of particles supported by $\mathcal{T}$ does not change between $\mathbf{y}$ and $\mathbf{y}'$, e.g. $\mathcal{I}_{\mathbf{y}} = \mathcal{I}_{\mathbf{y}'}$, then $Z_{\mathbf{y}} = Z_{\mathbf{y}'}$ and the acceptance probability becomes 1. This is how *Gibbs sampling* is derived.

Notation: when the current particle is clear from context we might just write $\mathcal{T}(\mathbf{y}')$ instead of $\mathcal{T}(\mathbf{y} \to \mathbf{y}')$. Also, if the transition changes only a fixed subset of the particle variables, we might just write those variables instead of $\mathbf{y}'$.

2.4.1 Gibbs Sampling

Gibbs Sampling is a particular case of MCMC, in which the particles are changed one variable at a time. For each variable Y_i, there is a transition function $\mathcal{T}^i(\mathbf{y} \to \mathbf{y}')$ defined as:

$$\mathcal{T}^i\left((\mathbf{y}_{-i}, y_i) \to (\mathbf{y}_{-i}, y_i')\right) = \frac{\pi(\mathbf{y}_{-i}, y_i')}{Z_{\mathbf{y}}} = P(Y_i = y_i' \mid \mathbf{y}_{-i}, \mathbf{x}) \tag{2.37}$$

where the supported particles are only those that are identical to $\mathbf{y}$ in their assignment to $\mathbf{Y}_{-i}$ and $Z_{\mathbf{y}} = \sum_{y_i} \pi(\mathbf{y}_{-i}, y_i)$ is (as usual) a normalization constant. This means that $\mathcal{I}_{\mathbf{y}} = \mathcal{I}_{\mathbf{y}'}$, and as shown above, in this case the acceptance probability is 1.

The values for $P(y_i' \mid \mathbf{y}_{-i}, \mathbf{x})$ can usually be computed very efficiently, especially in a Bayesian network. In such a case the joint distribution is simply a product of CPDs, and the only CPDs that depend on the value of Y_i (other than the CPD of Y_i) are those of its children. Hence one only needs to evaluate those CPDs when calculating the weight of each particle, since the rest of the CPDs will cancel out anyway.

A typical implementation of Gibbs sampling goes over the Y_i variables in a random permutation and applies $\mathcal{T}^i$ to each one. This constitutes a single Gibbs iteration.

2.4.2 Using MCMC To Get The MAP Assignment

Although the goal of MCMC is to approximate the posterior distribution over the hidden variables through the samples it produces, the same framework is sometimes used for a different purpose - to find the MAP assignment. One way to do so, is by applying the Gibbs sampling algorithm in a greedy mode, in which instead of *sampling*

from the proposal distribution, one transitions to the particle that *maximizes* it. This algorithm is called *Iterated Conditional Modes* (ICM)[2].

One approach that transitions from Gibbs sampling to ICM is called *simulated annealing*. Here we gradually make the joint distribution more "peaky", by exponentiating it. When used in the context of Gibbs sampling, we sample Y_i by giving each potential value y_i' a weight of $P(y_i' \mid \mathbf{y}_{-i}, \mathbf{x})^T$, where T is the *temperature* parameter. When $T = 1$, this corresponds to standard Gibbs sampling, and as T goes to infinity the model behaves more like ICM. In between we use a policy the grows T as a function of the current iteration.

2.4.3 MCMC with collapsed particles

In a collapsed particle, not all the hidden model variables receive an assignment. The MCMC algorithm works the same way as before, except the joint distribution over $(\mathbf{X}, \mathbf{Y})$ is now replaced with one that marginalizes over the missing variables.

In other words, if our collapsed particle does not include an assignment to Y_i, we simply use $\pi(\mathbf{y}_{-i}) = \sum_{y_i} \pi(\mathbf{y})$ instead of $\pi(\mathbf{y})$ in the Metropolis-Hastings equations. In the cases where there is a closed-form formula for the marginalization, using the collapsed particle is often a more efficient move, as it covers larger regions of the probabilistic space by not being tied to a specific value of Y_i.

2.4.4 MCMC With Auxiliary Variables

In Section 2.4.3 we showed how we can remove some of the model variables from our particles. We can also do the opposite. Sometimes we need to use auxiliary variables in order to decide which transition $\mathcal{T}$ to use. If we want to add such a variable R, the trick is to pretend that it was always part of our probabilistic model. In other words, we now have $\pi(\mathbf{y}, r) = \pi(\mathbf{y})P(R = r \mid \mathbf{y}, \mathbf{x})$. We are free to define $P(r \mid \mathbf{y}, \mathbf{x})$ in whatever way we want. We can now sample r from this distribution, perform a MH step over the augmented particle $(\mathbf{y}, r)$, and after we are done we can forget the value we gave to R, until the next time we need it.

2.4.5 Slice Sampling

One example for using an auxiliary variable in this way is done in a move termed *slice sampling* [22]. Let $\mathbf{y}$ be the current particle. Choose a threshold r uniformly from the interval $(0, \pi(\mathbf{y}))$. Then, use a transition probability $\mathcal{T}$ to choose the next particle $\mathbf{y}'$, however restrict the supported particles to the set $\mathcal{I}_r = \{\mathbf{y}' \mid \pi(\mathbf{y}') \geq r\}$. i.e., the move "slices" the surface of the probabilistic space at level r and only considers particles above the slice.

Under this move,

$$\begin{aligned}\pi(\mathbf{y}, r) =& \pi(\mathbf{y}) \cdot P(r \mid \mathbf{y}, \mathbf{x}) \\ =& \pi(\mathbf{y}) \cdot \frac{1}{\pi(\mathbf{y})} = 1.\end{aligned}$$

Plugging this into Eq. (2.34) (using the augmented particle) we see that as long as

$$\mathcal{T}\left((\mathbf{y}, r) \to (\mathbf{y}', r)\right) = \mathcal{T}\left((\mathbf{y}', r) \to (\mathbf{y}, r)\right), \tag{2.38}$$

we can guarantee a MH acceptance probability of 1.

We can use slice sampling to speed up Gibbs sampling. Recall that in Gibbs sampling we want our set of supported particles to differ only in their assignment to single variable $Y_i \in \mathbf{Y}$. Normally we will need to evaluate $P(Y_i = y_i \mid \mathbf{x}, \mathbf{y}_{-i})$ for each $y_i \in \mathit{Val}(Y_i)$. If $\mathit{Val}(Y_i)$ is large (or harder still, if Y_i is continuous), this can be computationally expensive.

Instead, we use slice sampling. We choose the threshold r as described above. Then, we choose the next y' using the following procedure[1]:

1. Choose a threshold $r \sim \text{Uniform}\left([0, \pi(y_i, \mathbf{y}_{-i})\right)$

2. Set $\alpha = \min \mathit{Val}(Y_i); \beta = \max \mathit{Val}(Y_i)$;

[1] In the case Y_i is continuous, the same procedure holds with minor changes, namely y_i' is sampled uniformly from the open interval (α, β), and in 4a α and β are simply set to y_i'.

3. Sample $y'_i \sim \text{Uniform}(\{\alpha, \alpha+1, \ldots, \beta\})$

4. *while* $\pi(y'_i, \mathbf{y}_{-i}) < r$

   (a) *if* $y'_i < y_i$ set $\alpha = y'_i + 1$ *else* set $\beta = y'_i - 1$

   (b) Sample $y'_i \sim \text{Uniform}(\{\alpha, \alpha+1, \ldots, \beta\})$

This procedure satisfies Eq. (2.38)[2] , and hence we can simply accept the transition and set $Y_i = y'_i$. Like in Gibbs sampling, we can save some time computing π in lines 1 and 4 by only computing those (few) factors that depend on Y_i. However, unlike Gibbs sampling, the expected number of such computations is now $\log_2 K$ instead of K, where $K = |\mathit{Val}(Y_i)|$.

This sampling technique is most efficient when we believe close values of Y_i should induce similar likelihoods. In such a case the move can be as efficient in converging towards the target distribution as Gibbs sampling, at a fraction (well, log) of the computational cost.

2.4.6 Example: Bayesian Clustering

In the standard mixture model, we generate samples $\mathbf{X} = \{X_1, X_2, \ldots, X_N\}$ from K different classes. First, the class assignments $\mathbf{C} = \{C_1, C_2, \ldots, C_N\}$ are drawn from a multinomial vector $\beta = (\beta_1, \ldots, \beta_k)$. Then, each class generates all its samples from a distribution associated with that class. The K distributions are all members of the family $\mathcal{F}$ parametrized by θ, so that $\mathcal{F}(\theta_k)$ is the distribution that generated all the samples from class k, where $\Theta = \{\theta_1, \ldots, \theta_K\}$ are the class parameters. For example, in a Mixture of Gaussians model, $\mathcal{F}$ is the set of all m-dimensional Gaussians with θ encoding their mean and variance. Our goal is to cluster the samples - find out which samples belong to the same class and what are the parameters associated with that class. One way to do so is to find the MAP of $P(C, \Theta|X)$. Since exact inference in this case is hard, one resorts to methods of approximate inference. In this case we focus on MCMC using Gibbs sampling.

[2] [22] gives a number of other efficient transitions that satisfy this condition.

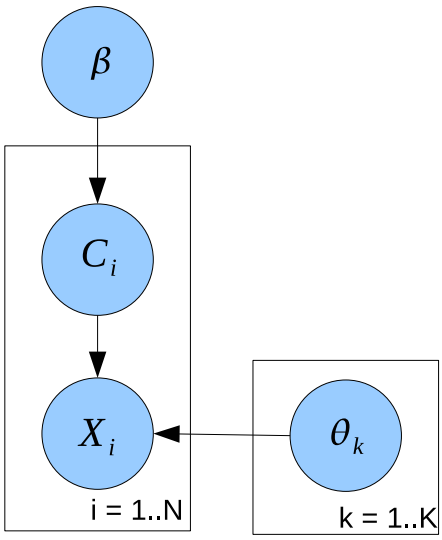


Figure 2.4: A Bayesian network of a finite mixture model

The above scenario can be modeled using the graphical model in Figure 2.4, formally defined as:

$$\begin{aligned}
\theta_k &\sim \mathcal{H} && \forall k = 1 \dots K \\
\beta &\sim \text{Dirichlet}\left(\frac{\alpha}{K}, \dots, \frac{\alpha}{K}\right) \\
C_i &\sim \text{multinomial}(\beta) && \forall i = 1 \dots N \\
X_i &\sim \mathcal{F}(\theta_{C_i}) && \forall i = 1 \dots N
\end{aligned}$$

Notice that we do not observe the values for the class parameters Θ and the class prior β, and hence we have to put a prior on them. The prior for Θ can be any distribution $\mathcal{H}$. The prior on β should be symmetric (= invariant to permutations of the vector elements), as we are indifferent regarding the usage of the class labels. A suitable prior is the symmetric prior Dirichlet $\left(\frac{\alpha}{K}, \dots, \frac{\alpha}{K}\right)$. A large value for α reflects a belief that samples should be partitioned evenly among all K classes, while a small value prefers peakier distributions, in which many classes could have a near-zero probability.

Applying Collapsed Gibbs

Our observed variables are $\mathbf{X}$, our hidden variables are $\mathbf{C}, \Theta, \beta$, and we have α as a fixed parameter. Our goal is to approximate the MAP of $P(C, \Theta|X)$. Using collapsed Gibbs, we will generate particles from $P(C, \Theta|X)$, and use simulated annealing in the hope to obtain the MAP assignment to C, θ. In other words, our particles assign values to C and Θ, and integrate over β. To do Gibbs, we should sample each θ_k (for $k = 1 \dots K$) and each C_i (for $i = 1 \dots N$) given the rest of the variables in C and Θ. This evaluates to the following equations:

$$P(\theta_k|X, K, \theta_{-k}) \propto \mathcal{H}(\theta_k) \prod_{i:C_i=k} \mathcal{F}(X_i; \theta_k) \tag{2.39}$$

$$\begin{aligned} P(C_i = k|X, \mathbf{C}_{-i}, \Theta) &\propto P(C_i = k \mid \mathbf{C}_{-i}) \mathcal{F}(X_i; \theta_k) \\ &\propto \left(N_{-i,k} + \frac{\alpha}{K}\right) \mathcal{F}(X_i; \theta_k) \end{aligned} \tag{2.40}$$

where $N_{-i,k} = \sum_{j:j\neq i} \mathbb{1}\{C_j = k\}$.

We used Eq. (2.8) to get the closed form formula for $P(C_i|\mathbf{C}_{-i})$. To sample from Eq. (2.40), one needs to compute the K possible values for the RHS, and sample from them multinomially (after normalization). To use annealing, simply exponentiate each value by the current temperature. The complexity of a full Gibbs iteration over the C_i variables is therefore $O(NK)$.

Sampling from Eq. (2.39) might be more complicated, depending on the choice of $\mathcal{H}$ and the cardinality of θ_k. Notice that the RHS only depends on the samples currently assigned to cluster k. For some pairs of $\mathcal{F}$ and $\mathcal{H}$ we might find that the posterior of θ_k can be represented as a simple distribution that we can easily sample from. Otherwise, we can use slice sampling or other MH moves to make proposals that can be computed efficiently.

2.5 Non-Parametric Models

In this section we discuss a class of distributions called *non-parametric*. Each distribution in the class actually has an infinite number of parameters. However, those

distributions are built so that only a finite number of those parameters is used when creating a finite amount of data. The power of these distributions is that each has the capacity to explain the same under different choices for the number of parameters. In other words, the complexity of the model, measured by the number of parameters used to explain the data, is now part of the Bayesian game, and thus inference in such model would tailor its complexity to the complexity of the data. To best explain how we can come to such a distribution, I would develop further the Bayesian clustering example illustrated in Section 2.4.6, which supported only a fixed number of clusters, and extend it an unbounded number.

2.5.1 Example: Bayesian Clustering - Continued

A typical problem in the clustering scenario of Section 2.4.6 is that the number of classes used, K, has to be fixed in advance, even though it might be unknown. To cope with this problem, one could try experimenting with different values for K, and use some evaluation criteria to decide which K is optimal for the scenario. An alternative way is to increase K to a very large value, perhaps to the size of the data, N. However, this incurs a heavy computational cost on the algorithm, as the complexity of each iteration grows linearly with K. Moreover, most of this computation seems like a waste - with such a large K many of the classes are likely to remain empty, why should we waste time sampling their θ_k?

The solution is to use collapsed Gibbs, reduce the size of each particle, and thus narrow down the number of necessary sampling moves. First, we can include in our particles only θ_k parameters of non-empty classes. Second, recall that our prior β is symmetric, and that is because we don't really care about the numerical values for the C_i, but rather the *partition* of the samples ("who is with whom"). Indeed, if we permute the labels of the clusters we get a particle that has exactly the same probability. This means that as long as we are blind to those numerical values, our particles can at anytime replace the "true assignment" with any arbitrary assignment consistent with the same partition, and it would not effect the Markov Chain. For example, we can always use a "de-fragmented" assignment where the non-empty

clusters are numbered $1..L$ (for partitions with L clusters). In other words, our particles no longer store the current assignment of the samples. They store their partition.

So, to summarize, what we did here is reduced the dimensionality of each particle, by marginalizing over unimportant variables and irrelevant aspects of our full probabilistic space. We can still apply MCMC and Gibbs sampling in this reduced space!

Assume that in our particle, $\mathbf{C}$ partitions the samples into L (out of K possible) clusters numbered $1..L$, and let $\Theta' = \{\theta_k\}_{k=1}^{L}$. We can sample each $\theta_k \in \Theta'$ using Eq. (2.39), as before (and keep $\Theta \setminus \Theta'$ unobserved). As for the assignment $\mathbf{C}$, we notice that changing C_i, while keeping the other assignments fixed, induces $L+1$ possible partitions: one for each existing cluster $1..L$ to which i can join, and one for opening a new cluster. For that latter partition, there are $K-L$ currently-empty clusters that we can assign C_i to, that will all yield that partition. Hence, we will combine these equivalent cases to a single case which we denote as $C_i = \text{new}$. We also need to integrate over the unknown θ value for those currently-empty clusters. This evaluates to the following conditional probability:

$$P(C_i = \text{new} \mid \mathbf{X}, \mathbf{C}_{-i}, \Theta') \propto (K-L)\frac{\alpha}{K}\int \mathcal{H}(\theta)\mathcal{F}(X_i;\theta)d\theta \tag{2.41}$$

As for the existing clusters, we use the same equation as Eq. (2.40).

$$P(C_i = k \mid X, \mathbf{C}_{-i}, \Theta') \propto \left(N_{-i,k} + \frac{\alpha}{K}\right)\mathcal{F}(X_i;\theta_k) \tag{2.42}$$

Collapsing equivalent assignments allowed us to compute only $L+1$ values for this i, and the complexity of the Gibbs iteration was reduced to $O(NL)$. This complexity does not depend on the number of classes K. If this is the case, why not increase K even further? Let's consider $K \to \infty$. The update equation for θ_k remains the same,

and the one for $\mathbf{C}$ becomes:

$$P(C_i = k \mid \mathbf{X}, \mathbf{C}_{-i}, \Theta') \propto N_{-i,k} \cdot \mathcal{F}(X_i; \theta_k) \tag{2.43}$$

$$P(C_i = \text{new} \mid \mathbf{X}, \mathbf{C}_{-i}, \Theta') \propto \alpha \cdot \int \mathcal{H}(\theta)\mathcal{F}(X_i; \theta)d\theta \tag{2.44}$$

If we sampled $C_i = \text{new}$ we simply set $C_i = L+1$, i.e. obtain an assignment consistent with the new partition. If as a result of the change in C_i one of the clusters becomes empty, we can (for implementation purposes) renumber the cluster labels so there would not be any "holes". The MCMC scheme illustrated here essentially lifts the bound on the number of classes, and induces a prior that allows any possible partition of the samples. The number of parameters used to generate the samples, L, is not known in advance, changes dynamically between the different particles, and generally grows (logarithmically) with the size of the data. This is how a non-parametric model behaves.

2.5.2 The Chinese Restaurant Process

So what did we do in Section 2.5.1? We started with

$$\begin{aligned} \beta &\sim \text{Dirichlet}\left(\frac{\alpha}{K}, \ldots, \frac{\alpha}{K}\right) \\ C_i &\sim \text{multinomial}(\beta) \quad \forall i = 1 \ldots N \\ \theta_k &\sim \mathcal{H} \qquad\qquad\quad \forall k = 1 \ldots K \end{aligned}$$

and took K to infinity. On the flip side, instead of dealing with the infinite space of assignments we collapsed all equivalent assignments so a single representative $\mathbf{C}$. We saw that under this model we have for $\mathbf{C}$:

$$P(C_i = k \mid \mathbf{C}_{-i}) \propto N_{-i,k} \tag{2.45}$$

$$P(C_i = \text{new} \mid \mathbf{C}_{-i}) \propto \alpha \tag{2.46}$$

We can use these equations to sample $\mathbf{C} = (C_1, C_2, \ldots, C_N)$ one variable after the other, without observing β, in the same way we did so under the Dirichlet distribution

in Eq. (2.9). This sampling procedure for a partition $\mathbf{C}$ gained a culinary narrative called *the Chinese restaurant process*. The guests (samples) enter the restaurant one by one. Each guest chooses between the non-empty tables (clusters) with probability proportional to the number of guests already sitting at that table, or it sits at a new table, with probability proportional to α.

The difference compared to Eq. (2.9) is that here we are sampling a *partition*, not an assignment, i.e. the "actual" assignment could be any one consistent with the partition. Fortunately, the partition is all we need.

Definition 2.5.1: We write $\mathbf{C} \sim \mathrm{CRP}(\alpha, N)$ if $\mathbf{C} = (C_1, \ldots, C_N)$ is sampled from the Chinese restaurant process with positive parameter α. ∎

Corollary 2.5.2: (without proof) If $\mathbf{C} \sim \mathrm{CRP}(\alpha, N)$, then

$$P(\mathbf{C}) = \frac{\alpha^L \Gamma(\alpha) \prod_{k=1}^{L} (N_k - 1)!}{\Gamma(N + \alpha)} \tag{2.47}$$

assuming $\mathbf{C}$ is a de-fragmented representation of the partition with the non-empty clusters labeled $1 \ldots L$. Notice that the sampling order does not effect the probability of the partition.

2.5.3 The Dirichlet Process

Our Dirichlet prior over β cannot be easily extended to K at infinity. Fortunately, it turns out that there is a well defined prior over β that results in the same behavior. This prior, in yet another metaphor, is called *the stick-breaking prior (SBP)*. The SBP, parametrized by α, generates an infinite vector β whose entries sum to one (see [28] for its definition).

It follows that if

$$\begin{aligned} \beta &\sim \mathrm{SBP}(\alpha) \\ C_i &\sim \mathrm{multinomial}(\beta) \quad \forall i = 1 \ldots N \end{aligned}$$

then $\mathbf{C} \sim CRP(\alpha, N)$.

Definition 2.5.3: [Dirichlet Process]. Let α be a positive scalar, and $\mathcal{H}$ a distribution. Let G be a distribution over the same space as $\mathcal{H}$. We say that $G \sim \mathrm{DP}(\mathcal{H}, \alpha)$ if

$$G(\theta) = \sum_{k=1}^{\infty} \beta_k \mathbb{1}\{\theta = \theta_k\},$$

where $\beta \sim \mathrm{SBP}(\alpha)$, and $\theta_k \sim \mathcal{H}$ for $k = 1 \ldots \infty$. ∎

Hence G is a **random distribution**. Notice that by construction, G is discrete with probability 1. $\mathcal{H}$ is called the *base distribution* and the real number α is the *concentration parameter*.

Consider the finite mixture model described in Section 2.4.6. We have already showed two equivalent extensions of it to an infinite mixture. The first by sampling $\mathbf{C}$ directly using a CRP. The second by sampling β using a SBP, and then each C_i independently from β. We can now show a third formulation using a DP:

$$\begin{aligned} G \sim & \mathrm{DP}(\mathcal{H}, \alpha) & \\ \phi_i \sim & G & \forall i = 1 \ldots N \\ X_i \sim & \mathcal{F}(\phi_i) & \forall i = 1 \ldots N \end{aligned}$$

Intuitively, draws from a $DP(H, \alpha)$ can be viewed as discrete approximations to the base distribution H (that need not be discrete), where α controls the degree of approximation (the higher the alpha, the closer we are to H).