

An Introduction to Probabilistic Graphical Models

Michael I. Jordan
University of California, Berkeley

September 2, 2002

Chapter 2

Conditional Independence and Factorization

A graphical model can be thought of as a probabilistic database, a machine that can answer “queries” regarding the values of sets of random variables. We build up the database in pieces, using probability theory to ensure that the pieces have a consistent overall interpretation. Probability theory also justifies the inferential machinery that allows the pieces to be put together “on the fly” to answer queries.

Consider a set of random variables $\{X_1, X_2, \dots, X_n\}$ and let x_i represent the realization of random variable X_i . Each random variable may be scalar-valued or vector-valued. Thus x_i is in general a vector in a vector space. In this section, for concreteness, we assume that the random variables are discrete; in general, however, we make no such restriction. There are several kinds of query that we might be interested in making regarding such an ensemble. We might, for example, be interested in knowing whether one subset of variables is independent of another, or whether one subset of variables is conditionally independent of another subset of variables given a third subset. Or we might be interested in calculating conditional probabilities—the probabilities of one subset of variables given the values of another subset of variables. Still other kinds of queries will be described in later chapters. In principle all such queries can be answered if we have in hand the joint probability distribution, written $P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n)$. Questions regarding independence can be answered by factoring the joint probability distribution, and questions regarding conditional probabilities can be answered by appropriate marginalization and normalization operations.

To simplify our notation, we will generally express discrete probability distributions in terms of the probability mass function $p(x_1, x_2, \dots, x_n)$, defined as $p(x_1, x_2, \dots, x_n) \triangleq P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n)$. We also will often use X to stand for $\{X_1, \dots, X_n\}$, and x to stand for $\{x_1, \dots, x_n\}$, so that $P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n)$ can be written more succinctly as $P(X = x)$, or, more succinctly still, as $p(x)$. Note also that subsets of indices are allowed wherever single indices appear. Thus if $A = \{2, 4\}$ and $B = \{3\}$, then X_A is shorthand for $\{X_2, X_4\}$, X_B is shorthand for $\{X_3\}$, and $P(X_A = x_A | X_B = x_B)$ is shorthand for $P(X_2 = x_2, X_4 = x_4 | X_3 = x_3)$.

While it is in fact our goal to maintain and manipulate representations of joint probabilities, we must not be naive regarding the size of the representations. In the case of discrete random

variables, one way to represent the joint probability distribution is as an n -dimensional table, in which each cell contains the probability $p(x_1, x_2, \dots, x_n)$ for a specific setting of the variables $\{x_1, x_2, \dots, x_n\}$. If each variable x_i ranges over r values, we must store and manipulate r^n numbers, a quantity exponential in n . Given that we wish to consider models in which n is in the hundreds or thousands, such a naive tabular representation is out.

Graphical models represent joint probability distributions more economically, using a set of “local” relationships among variables. To define what we mean by “local” we avail ourselves of graph theory.

2.1 Directed graphs and joint probabilities

Let us begin by considering directed graphical representations. A directed graph is a pair $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where \mathcal{V} is a set of nodes and \mathcal{E} a set of (oriented) edges. We will assume that \mathcal{G} is acyclic.

Each node in the graph is associated with a random variable. Formally, we assume that there is a one-to-one mapping from nodes to random variables, and we say that the random variables are *indexed* by the nodes in the graph. Thus, for each $i \in \mathcal{V}$, there is an associated random variable X_i . Letting $\mathcal{V} = \{1, 2, \dots, n\}$, as we often do for convenience, the set of random variables associated with the graph is given by $\{X_1, X_2, \dots, X_n\}$.

Although nodes and random variables are rather different formal objects, we will find it convenient to ignore the distinction, letting the symbol “ X_i ” refer both to a node and to its associated random variable. Indeed, we will often gloss over the distinction between nodes and random variables altogether, using language such as “the marginal probability of node X_i .”

Note that we will also sometimes use lower-case letters—that is, the realization variables x_i —to label nodes, further blurring distinctions. Given the strict one-to-one correspondence that we enforce between the notation for random variables (X_i) and their realizations (x_i), however, this is unlikely to lead to confusion.

It would be rather inconvenient to be restricted to the symbol “ X ” for random variables, and we often use other symbols as well. Thus, we may consider examples in which sets such as $\{W, X, Y, Z\}$ or $\{X_1, X_2, X_3, Y_1, Y_2, Y_3\}$ denote the set of random variables associated with a graph. As long as it is clear which random variable is associated with which node, then formally the random variables are “indexed” by the nodes in the graph as required, even though the indexing is not necessarily made explicit in the notation.

Each node has a set of *parent nodes*, which can be the empty set. For each node $i \in \mathcal{V}$, we let π_i denote the set of parents of node i . We also refer to the set of random variables X_{π_i} as the “parents” of the random variable X_i , exploiting the one-to-one relationship between nodes and random variables.

We use the locality defined by the parent-child relationship to construct economical representations of joint probability distributions. To each node $i \in \mathcal{V}$ we associate a function $f_i(x_i, x_{\pi_i})$. These functions are assumed to have the properties of conditional probability distributions: that is, $f_i(x_i, x_{\pi_i})$ is nonnegative and sums to one with respect to x_i for each value of x_{π_i} . We impose no additional constraint on these functions; in particular, there is no assumption of any relationship between the functions at different nodes.

Let $\mathcal{V} = \{1, 2, \dots, n\}$. Given a set of functions $\{f_i(x_i, x_{\pi_i}) : i \in \mathcal{V}\}$, we define a joint probability distribution as follows:

$$p(x_1, x_2, \dots, x_n) \triangleq \prod_{i=1}^n f_i(x_i, x_{\pi_i}). \quad (2.1)$$

That is, we define the joint probability as a product of the local functions at the nodes of the graph. To verify that the definition obeys the constraints on a joint probability, we check: (1) the right-hand side is clearly nonnegative; and (2) the assumption that each factor $f_i(x_i, x_{\pi_i})$ sums to one with respect to x_i , together with the assumption that the graph is acyclic, implies that the right-hand side sums to one with respect to $\{x_1, x_2, \dots, x_n\}$. In particular, we can sum “backward” from the leaves of the graph, summing over the values of leaf nodes and removing the nodes from the graph, obtaining a value of one at each step.¹

By choosing specific numerical values for the functions $f_i(x_i, x_{\pi_i})$, we generate a specific joint probability distribution. Ranging over all possible numerical choices for these functions, we define a family of joint probability distributions associated with the graph \mathcal{G} . It will turn out that this family is a natural mathematical object. In particular, as we will see later in this chapter, this family can be characterized not only in terms of products of local functions, but also more “graph-theoretically” in terms of the patterns of edges in the graph. It is this relationship between the different ways to characterize the family of probability distributions associated with a graph that is the key to the underlying theory of probabilistic graphical models.

With a definition of joint probability in hand, we can begin to address the problem of calculating conditional probabilities under this joint. Suppose in particular that we calculate $p(x_i | x_{\pi_i})$ under the joint probability in Eq. (2.1). What, if any, is the relationship between this conditional probability and $f_i(x_i, x_{\pi_i})$, a function which has the properties of a conditional probability but is otherwise arbitrary? As we ask the reader to verify in Exercise ??, these functions are in fact one and the same. That is, under the definition of joint probability in Eq. (2.1), the function $f_i(x_i, x_{\pi_i})$ is necessarily the conditional probability of x_i given x_{π_i} . Put differently, we see that the functions $f_i(x_i, x_{\pi_i})$ must form a consistent set of conditional probabilities under a single joint probability. This is a pleasant and somewhat surprising fact given that we can define the functions $f_i(x_i, x_{\pi_i})$ arbitrarily.

Given that functions $f_i(x_i, x_{\pi_i})$ are in fact conditional probabilities, we henceforth drop the f_i notation and write the definition in terms of $p(x_i | x_{\pi_i})$:²

$$p(x_1, x_2, \dots, x_n) = \prod_{i=1}^n p(x_i | x_{\pi_i}). \quad \text{equal to 2.1} \quad (2.2)$$

¹If this point is not clear now, it will be clear later when we discuss inference algorithms.

²Eq. (2.2) is often used as the definition of the joint probability for a directed graphical model. Such a definition risks circularity, however, because it is not clear in advance that an arbitrary collection of conditional probabilities, $\{p(x_i | x_{\pi_i})\}$, are necessarily conditionals under the same joint probability. Moreover, it is not clear in advance that an arbitrary collection of conditional probabilities is consistent with each other. We thus prefer to treat Eq. (2.1) as the definition and view Eq. (2.2) as a consequence. Having made this cautionary note, however, for simplicity we refer to Eq. (2.2) as the “definition” of joint probability in the remainder of the chapter.

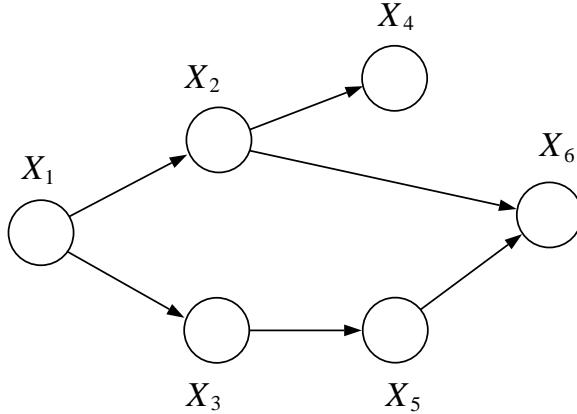


Figure 2.1: An example of a directed graphical model.

We refer to the conditional probabilities $p(x_i | x_{\pi_i})$ as the *local conditional probabilities* associated with the graph \mathcal{G} . These functions are the building blocks whereby we synthesize a joint distribution associated with the graph \mathcal{G} .

Figure 2.1 shows an example on six nodes. According to the definition, we obtain the joint probability as follows:

$$p(x_1, x_2, x_3, x_4, x_5, x_6) = p(x_1)p(x_2 | x_1)p(x_3 | x_1)p(x_4 | x_2)p(x_5 | x_3)p(x_6 | x_2, x_5), \quad (2.3)$$

by taking the product of the local conditional distributions.

Let us now return to the problem of representational economy. Are there computational advantages to representing a joint probability as a set of local conditional probabilities?

Each of the local conditional probabilities must be represented in some manner. In later chapters we will consider a number of possible representations for these probabilities; indeed, this representational issue is one of the principal topics of the book. For concreteness, however, let us make a simple choice here. For a discrete node X_i , we must represent the probability that node X_i takes on one of its possible values, for each combination of values for its parents. This can be done using a table. Thus, for example, the probability $p(x_1)$ can be represented using a one-dimensional table, and the probability $p(x_6 | x_2, x_5)$ can be represented using a three-dimensional table, one dimension for each of x_2, x_5 and x_6 . The entire set of tables for our example is shown in Figure 2.2, where for simplicity we have assumed that the nodes are binary-valued. Filling these tables with specific numerical values picks out a specific distribution in the family of distributions defined by Eq. (2.3).

In general, if m_i is the number of parents of node X_i , we can represent the conditional probability associated with node X_i with an $(m_i + 1)$ -dimensional table. If each node takes on r values, then we require a table of size r^{m_i+1} .

We have exchanged exponential growth in n , the number of variables in the domain, for exponential growth in m_i , the number of parents of individual nodes X_i (the “fan-in”). This is very often a happy exchange. Indeed, in many situations the maximum fan-in in a graphical model is relatively small and the reduction in complexity can be enormous. For example, in hidden Markov

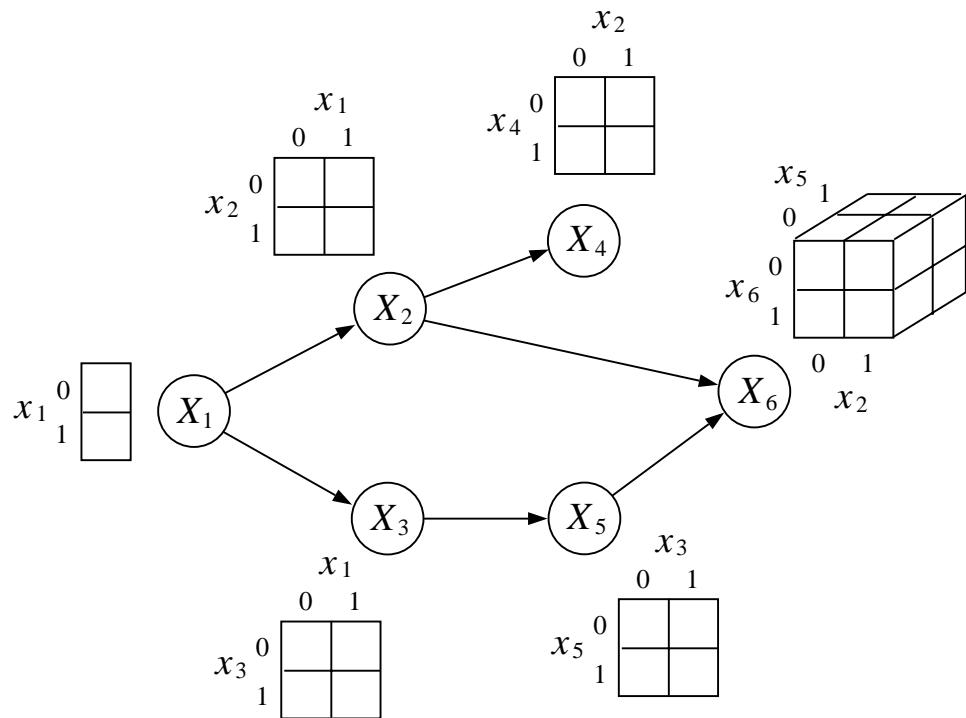


Figure 2.2: The local conditional probabilities represented as tables. Each of the nodes is assumed to be binary-valued. Each of these tables can be filled with arbitrary nonnegative numerical values, subject to the constraint that they sum to one for given fixed values of the parents of a node. Thus, each column in each table must sum to one.

models (see Chapter 12), each node has at most a single parent, while the number of nodes n can be in the thousands.

The fact that graphs provide economical representations of joint probability distributions is important, but it is only a first hint of the profound relationship between graphs and probabilities. As we show in the remainder of this chapter and in the following chapter, graphs provide much more than a data structure; in particular, they provide inferential machinery for answering questions about probability distributions.

2.1.1 Conditional independence

An important class of questions regarding probability distributions has to do with conditional independence relationships among random variables. We often want to know whether a set of variables is independent of another set, or perhaps conditionally independent of that set given a third set. Independence and conditional independence are important qualitative aspects of probability theory.

By definition, X_A and X_B are *independent*, written $X_A \perp\!\!\!\perp X_B$, if:

$$p(x_A, x_B) = p(x_A)p(x_B), \quad (2.4)$$

and X_A and X_C are *conditionally independent given X_B* , written $X_A \perp\!\!\!\perp X_C | X_B$, if:

$$p(x_A, x_C | x_B) = p(x_A | x_B)p(x_C | x_B), \quad (2.5)$$

or, alternatively,

$$p(x_A | x_B, x_C) = p(x_A | x_B), \quad (2.6)$$

for all x_B such that $p(x_B) > 0$. Thus, to establish independence or conditional independence we need to factor the joint probability distribution.

Graphical models provide an intuitively appealing, symbolic approach to factoring joint probability distributions. The basic idea is that representing a probability distribution within the graphical model formalism involves making certain independence assumptions, assumptions which are embedded in the structure of the graph. From the graphical structure other independence relations can be derived, reflecting the fact that certain factorizations of joint probability distributions imply other factorizations. The key advantage of the graphical approach is that such factorizations can be read off from the graph via simple graph search algorithms. We will describe such an algorithm in Section 2.1.2; for now let us try to see in general terms why graphical structure should encode conditional independence.

The *chain rule of probability theory* allows a probability mass function to be written in a general factored form, once a particular ordering for the variables is chosen. For example, a distribution on the variables $\{X_1, X_2, \dots, X_6\}$ can be written as:

$$\begin{aligned} & p(x_1, x_2, x_3, x_4, x_5, x_6) \\ &= p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2)p(x_4 | x_1, x_2, x_3)p(x_5 | x_1, x_2, x_3, x_4)p(x_6 | x_1, x_2, x_3, x_4, x_5), \end{aligned}$$

where we have chosen the usual arithmetic ordering of the nodes. In general, we have:

$$p(x_1, x_2, \dots, x_n) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1}). \quad (2.7)$$

Comparing this expansion, which is true for an arbitrary probability distribution, with the definition in Eq. (2.2), we see that our definition of joint probability involves dropping some of the conditioning variables in the chain rule. Inspecting Eq. (2.6), it seems natural to try to interpret these missing variables in terms of conditional independence. For example, the fact that $p(x_4 | x_2)$ appears in Eq. (2.3) in the place of $p(x_4 | x_1, x_2, x_3)$ suggests that we should expect to find that X_4 is independent of X_1 and X_3 given X_2 .

Taking this idea a step further, we might posit that *missing variables in the local conditional probability functions correspond to missing edges in the underlying graph*. Thus, $p(x_4 | x_2)$ appears as a factor in Eq. (2.3) because there are no edges from X_1 and X_3 to X_4 . Transferring the interpretation from missing variables to missing edges we obtain a probabilistic interpretation for the missing edges in the graph in terms of conditional independence. Let us formalize this interpretation.

Define an ordering I of the nodes in a graph \mathcal{G} to be *topological* if for every node $i \in \mathcal{V}$ the nodes in π_i appear before i in the ordering. For example, the ordering $I = (1, 2, 3, 4, 5, 6)$ is a topological ordering for the graph in Figure 2.1. Let ν_i denote the set of all nodes that appear earlier than i in the ordering I , excluding the parent nodes π_i . For example, $\nu_5 = \{1, 2, 4\}$ for the graph in Figure 2.1.

As we ask the reader to verify in Exercise ??, the set ν_i necessarily contains all *ancestors* of node i (excluding the parents π_i), and may contain other *nondescendant* nodes as well.

Given a topological ordering I for a graph \mathcal{G} we associate to the graph the following set of *basic conditional independence statements*:

$$\{X_i \perp\!\!\!\perp X_{\nu_i} | X_{\pi_i}\} \quad (2.8)$$

for $i \in \mathcal{V}$. Given the parents of a node, the node is independent of all earlier nodes in the ordering.

For example, for the graph in Figure 2.1 we have the following set of basic conditional independencies:

$$X_1 \perp\!\!\!\perp \emptyset \quad | \quad \emptyset \quad (2.9)$$

$$X_2 \perp\!\!\!\perp \emptyset \quad | \quad X_1 \quad (2.10)$$

$$X_3 \perp\!\!\!\perp X_2 \quad | \quad X_1 \quad (2.11)$$

$$X_4 \perp\!\!\!\perp \{X_1, X_3\} \quad | \quad X_2 \quad (2.12)$$

$$X_5 \perp\!\!\!\perp \{X_1, X_2, X_4\} \quad | \quad X_3 \quad (2.13)$$

$$X_6 \perp\!\!\!\perp \{X_1, X_3, X_4\} \quad | \quad \{X_2, X_5\}, \quad (2.14)$$

where the first two statements are vacuous.

Is this interpretation of the missing edges in terms of conditional independence consistent with our definition of the joint probability in Eq. (2.2)? The answer to this important question is “yes,” although proof will be again postponed until later. Let us refer to our example, however, to provide a first indication of the basic issues.

Let us verify that X_1 and X_3 are independent of X_4 given X_2 by direct calculation from the

joint probability in Eq. (2.3). We first compute the marginal probability of $\{X_1, X_2, X_3, X_4\}$:

$$p(x_1, x_2, x_3, x_4) = \sum_{x_5} \sum_{x_6} p(x_1, x_2, x_3, x_4, x_5, x_6) \quad (2.15)$$

$$= \sum_{x_5} \sum_{x_6} p(x_1)p(x_2 | x_1)p(x_3 | x_1)p(x_4 | x_2)p(x_5 | x_3)p(x_6 | x_2, x_5) \quad (2.16)$$

$$= p(x_1)p(x_2 | x_1)p(x_3 | x_1)p(x_4 | x_2) \sum_{x_5} p(x_5 | x_3) \sum_{x_6} p(x_6 | x_2, x_5) \quad (2.17)$$

$$= p(x_1)p(x_2 | x_1)p(x_3 | x_1)p(x_4 | x_2), \quad (2.18)$$

and also compute the marginal probability of $\{X_1, X_2, X_3\}$:

$$p(x_1, x_2, x_3) = \sum_{x_4} p(x_1)p(x_2 | x_1)p(x_3 | x_1)p(x_4 | x_2) \quad (2.19)$$

$$= p(x_1)p(x_2 | x_1)p(x_3 | x_1). \quad (2.20)$$

Dividing these two marginals yields the desired conditional:

$$p(x_4 | x_1, x_2, x_3) = p(x_4 | x_2), \quad (2.21)$$

which demonstrates the conditional independence relationship $X_4 \perp\!\!\!\perp \{X_1, X_3\} | X_2$.

We can readily verify the other conditional independencies in Eq. (2.14), and indeed it is not hard to follow along the lines of the example to prove in general that the conditional independence statements in Eq. (2.8) follow from the definition of joint probability in Eq. (2.2). Thus we are licensed to interpret the missing edges in the graph in terms of a basic set of conditional independencies.

More interestingly, we might ask whether there are other conditional independence statements that are true of such joint probability distributions, and whether these statements also have a graphical interpretation.

For example, for the graph in Figure 2.1 it turns out that X_1 is independent of X_6 given $\{X_2, X_3\}$. This is not one of the basic conditional independencies in the list in Eq. (2.14), but it is *implied* by that list. We can verify this conditional independence by algebra. In general, however, such algebraic calculations can be tedious and it would be appealing to find a simpler method for checking conditional independencies. Moreover, we might wish to write down *all* of the conditional independencies that are implied by our basic set. Is there any way to do this other than by trying to factorize the joint distribution with respect to all possible triples of subsets of the variables?

A solution to the problem is suggested by examining the graph in Figure 2.3. We see that the nodes X_2 and X_3 *separate* X_1 from X_6 , in the sense that all paths between X_1 and X_6 pass through X_2 or X_3 . Moreover, returning to the list of basic conditional independencies in Eq. (2.14), we see that the parents X_{π_i} block all paths from the node X_i to the earlier nodes in a topological ordering. This suggests that the notion of *graph separation* can be used to derive a graphical algorithm for inferring conditional independence.

We will have to take some care, however, to make the notion of “blocking” precise. For example, X_2 is *not* necessarily independent of X_3 given X_1 and X_6 , as would be suggested by a naive interpretation of “blocking” in terms of graph separation.

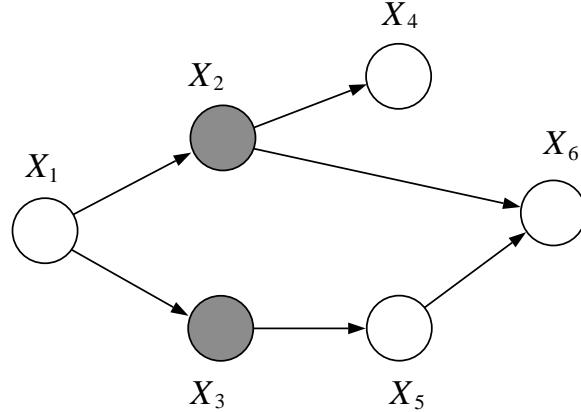


Figure 2.3: The nodes X_2 and X_3 separate X_1 from X_6 .

We will pursue the analysis of blocking and conditional independence in the following section, where we provide a general graph search algorithm to solve the problem of finding implied independencies.

Let us make a final remark on the definition of the set of basic conditional independence statements in Eq. (2.8). Note that this set is dependent on both the graph \mathcal{G} and on an ordering I . It is also possible to make an equivalent definition that is defined only in terms of the graph \mathcal{G} . In particular, recall that the set ν_i necessarily includes all ancestors of i (excluding the parents π_i). Note that the set of ancestors is independent of the ordering I . We thus might consider defining a basic set of independence statements that assert the conditional independence of a node from its ancestors, conditional on its parents. It turns out that the independence statements in this set hold if and only if the independence statements in Eq. (2.8) hold. As we ask the reader to verify in Exercise ??, this equivalence follows easily from the “Bayes ball” algorithm that we present in the following section.

The definition in Eq. (2.8) was chosen so as to be able to contrast the definition of the joint probability in Eq. (2.2) with the general chain rule in Eq. (2.7). An order-independent definition of the basic set of conditional independencies is, however, an arguably more elegant characterization of conditional independence in a graph, and it will take center stage in our more formal treatment of conditional independence and Markov properties in Chapter 16.

2.1.2 Conditional independence and the Bayes ball algorithm

The algorithm that we describe is called the *Bayes ball algorithm*, and it has the colorful interpretation of a ball bouncing around a graph. In essence it is a “reachability” algorithm, under a particular definition of “separation.”

Our approach will be to first discuss the conditional independence properties of three canonical, three-node graphs. We then embed these properties in a protocol for the bouncing ball; these are the local rules for a graph-search algorithm.

Two final remarks before we describe the algorithm. In our earlier discussion in Section 2.1.1,

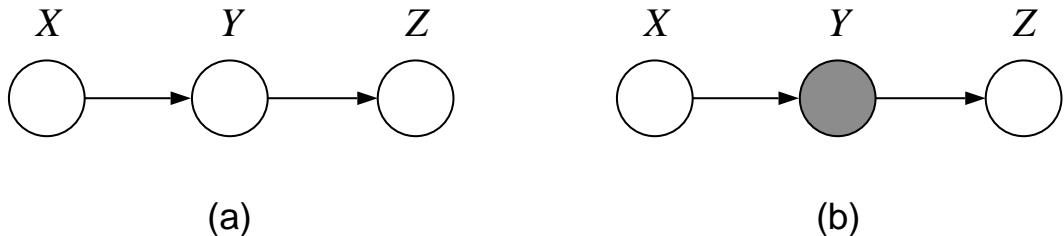


Figure 2.4: (a) The missing edge in this graph corresponds to the conditional independence statement $X \perp\!\!\!\perp Z | Y$. As suggested in (b), conditioning on Y has the graphical interpretation of blocking the path between X and Z .

and also in the current section, we presented conditional independence as being subservient to the basic definition in Eq. (2.2) of the joint probability. That is, we justified an assertion of conditional independence by factorizing Eq. (2.2) or one of its marginals. This is not the only point of view that we can take, however. Indeed it turns out that this relationship can be reversed, with Eq. (2.2) being derived from a characterization of conditional independence, and we will also introduce this point of view in this section. By the end of the current section we hope to have clarified what is meant by a “characterization of conditional independence.”

On a related note, let us recall a remark that was made earlier, which is that to each graph we associate a *family of joint probability distributions*. In terms of the definition of joint probability in Eq. (2.2), this family arises as we range over different choices of the numerical values of the local conditional probabilities $p(x_i | x_{\pi_i})$. Our work in the current section can be viewed as providing an alternative, more qualitative, characterization of a family of probability distributions associated to a given graph. In particular we can view the conditional independence statements generated by the Bayes ball algorithm as generating a list of constraints on probability distributions. Those joint probabilities that meet all of the constraints in this list are in the family, and those that fail to meet one or more constraints are out. It is then an interesting question as to the relationship between this characterization of a family of probability distributions in terms of conditional independence and the more numerical characterization of a family in terms of local conditional probabilities. This is the topic of Section 2.1.3.

Three canonical graphs

As we discussed in Section 2.1.1, the missing edges in a directed graphical model can be interpreted in terms of conditional independence. In this section, we flesh out this interpretation for three simple graphs.

Consider first the graph shown in Figure 2.4, in which X , Y , and Z are connected in a chain. There is a missing edge between X and Z , and we interpret this missing edge to mean that X and Z are conditionally independent given Y ; thus:

$$X \perp\!\!\!\perp Z \mid Y. \quad (2.22)$$

Moreover, we assert that there are no other conditional independencies associated with this graph.

Let us justify the first assertion, showing that $X \perp\!\!\!\perp Z | Y$ can be derived from the assumed form of the joint distribution for directed models Eq. (2.2). We have:

$$p(x, y, z) = p(x)p(y|x)p(z|y), \quad (2.23)$$

which implies:

$$p(z|x, y) = \frac{p(x, y, z)}{p(x, y)} \quad (2.24)$$

$$= \frac{p(x)p(y|x)p(z|y)}{p(x)p(y|x)} \quad (2.25)$$

$$= p(z|y), \quad (2.26)$$

which establishes the independence.

The second assertion needs some explanation. What do we mean when we say that “there are no other conditional independencies associated with this graph”? It is important to understand that this does *not* mean that no further conditional independencies can arise in any of the distributions in the family associated with this graph (that is, distributions that have the factorized form in Eq. (2.23)). There are certainly some distributions which exhibit additional independencies. For example, we are free to choose any local conditional probability $p(y|x)$; suppose that we choose a distribution in which the probability of y happens to be the same no matter the value of x . We can readily verify that with this particular choice of $p(y|x)$, we obtain $X \perp\!\!\!\perp Y$.

The key point, then, is that Figure 2.4 does not assert that X and Y are necessarily dependent (i.e., not independent). That is, edges that are present in a graph do not necessarily imply dependence (whereas edges that are missing do necessarily imply independence). But the “lack of independence” does have a specific interpretation: the general theory that we present in Chapter 16 will imply that if a statement of independence is not made, then there exists at least one distribution for which that independence relation does not hold. For example, it is easy to find distributions that factorize as in Eq. (2.23) and in which X is not independent of Y .

In essence, the issue comes down to a difference between universally quantified statements and existentially quantified statements, with respect to the family of distributions associated with a given graph. Asserted conditional independencies *always* hold for these distributions. Non-asserted conditional independencies *sometimes* fail to hold for the distributions associated with a given graph, but sometimes they do hold. This of course has important consequences for algorithm design. In particular, if we build an algorithm that is based on conditional independencies, the algorithm will be correct for *all* of the distributions associated with the graph. An algorithm based on the absence of conditional independencies will *sometimes* be correct, sometimes not.

For an intuitive interpretation of the graph in Figure 2.4, let X be the “past,” Y be the “present,” and Z be the “future.” Thus our conditional independence statement $X \perp\!\!\!\perp Z | Y$ translates into the statement that the past is independent of the future given the present, and we can interpret the graph as a simple classical Markov chain.

Our second canonical graph is shown in Figure 2.5. We associate to this graph the conditional independence statement:

$$X \perp\!\!\!\perp Z | Y, \quad (2.27)$$

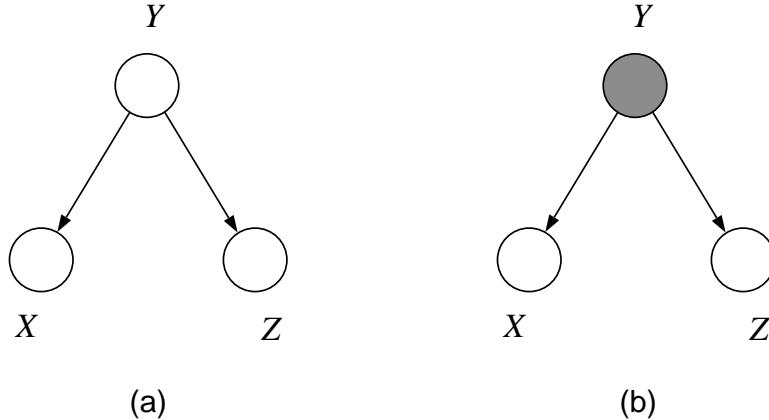


Figure 2.5: (a) The missing edge in this graph corresponds to the conditional independence statement $X \perp\!\!\!\perp Z | Y$. As suggested in (b), conditioning on Y has the graphical interpretation of blocking the path between X and Z .

and once again we assert that no other conditional independencies associated with this graph.

A justification of the conditional independence statement follows from the factorization rule. Thus:

$$p(x, y, z) = p(y)p(x \mid y)p(z \mid y) \quad (2.28)$$

implies:

$$p(x, z | y) = \frac{p(y)p(x | y)p(z | y)}{p(y)} \quad (2.29)$$

$$= p(x \mid y)p(z \mid y), \quad (2.30)$$

which means that X and Z are independent given Y .

An intuitive interpretation for this graph can be given in terms of a “hidden variable” scenario. Let X be the variable “shoe size,” and let Z be the variable “amount of gray hair.” In the general population, these variables are strongly dependent, because children tend to have small feet and no gray hair. But if we let Y be “chronological age”, then we might be willing to assert that $X \perp\!\!\!\perp Z | Y$; that is, given the age of a person, there is no further relationship between the size of their feet and the amount of gray hair that they have. The hidden variable Y “explains” all of the observed dependence between X and Z .

Note once again we are making no assertions of dependence based on Figure 2.5. In particular, we do not necessarily assume that X and Z are dependent because they both “depend” on the variable Y . (But we can assert that there are at least some distributions in which such dependencies are to be found).

Finally, the most interesting canonical graph is that shown in Figure 2.6. Here the conditional independence statement that we associate with the graph is actually a statement of marginal independence:

$$X \perp\!\!\!\perp Z, \quad (2.31)$$

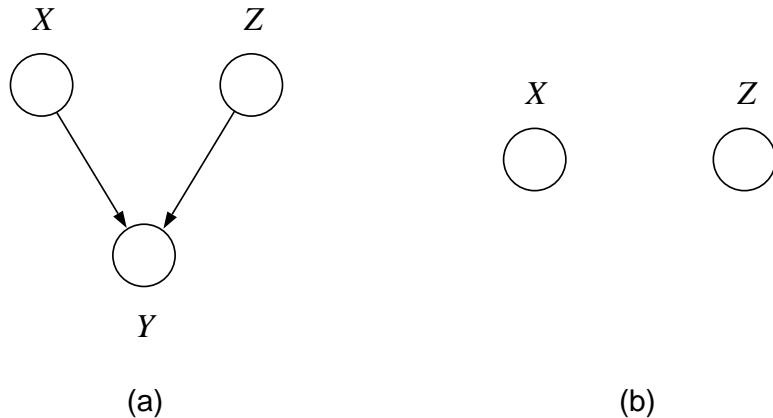


Figure 2.6: (a) The missing edge in this graph corresponds to the marginal independence statement $X \perp\!\!\!\perp Z$. As shown in (b), this is a statement about the subgraph defined on X and Z . Note moreover that conditioning on Y does not render X and Z independent, as would be expected from a naive characterization of conditional independence in terms of graph separation.

which we leave to the reader to verify in terms of the form of the joint probability. Once again, we assert that no other conditional independencies hold. In particular, note that we do not assert any conditional independence involving all three of the variables.

The fact that we do not assert that X is independent of Z given Y in Figure 2.6 is an important fact that is worthy of some discussion. Based on our earlier discussion, we should expect to be able to find scenarios in which a variable X is independent of another variable Z , given no other information, but once a third variable Y is observed these variables become dependent. Indeed, such a scenario is provided by a “multiple, competing explanation” interpretation of Figure 2.6.

Suppose that Bob is waiting for Alice for their noontime lunch date, and let $\{\text{late} = \text{"yes"}\}$ be the event that Alice does not arrive on time. One explanation of this event is that Alice has been abducted by aliens, which we encode as $\{\text{aliens} = \text{"yes"}\}$ (see Figure 2.7). Bob uses Bayes' theorem to calculate the probability $P(\text{aliens} = \text{"yes"} | \text{late} = \text{"yes"})$ and is dismayed to find that it is larger than the base rate $P(\text{aliens} = \text{"yes"})$. Alice has perhaps been abducted by aliens. Now let $\{\text{watch} = \text{"no"}\}$ denote the event that Bob forgot to set his watch to reflect daylight savings time. Bob now calculates $P(\text{aliens} = \text{"yes"} | \text{late} = \text{"yes"}, \text{watch} = \text{"no"})$ and is relieved to find that the probability of $\{\text{aliens} = \text{"yes"}\}$ has gone down again. The key point is that $P(\text{aliens} = \text{"yes"} | \text{late} = \text{"yes"}) \neq P(\text{aliens} = \text{"yes"} | \text{late} = \text{"yes"}, \text{watch} = \text{"no"})$, and thus **aliens** is not independent of **watch** given **late**.

On the other hand, it is reasonable to assume that **aliens** is *marginally* independent of **watch**; that is, Bob's watch-setting behavior and Alice's experiences with aliens are presumably unrelated and we would evaluate their probabilities independently, outside of the context of the missed lunch date.

This kind of scenario is known as “explaining-away” and it is commonplace in real-life situations. Moreover, there are other such scenarios (e.g., those involving multiple, synergistic explanations)

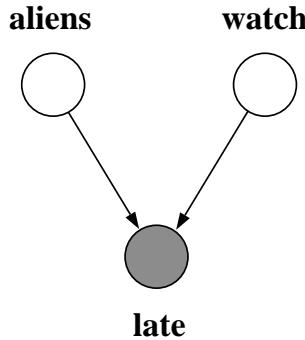


Figure 2.7: A graph representing the fact that Alice is late for lunch with Bob, with two possible explanations—that she has been abducted by aliens and that Bob has forgotten to set his watch to reflect daylight savings time.

in which variables that are marginally independent become dependent when a third variable is observed. We clearly do not want to assume in general that X is independent of Z given Y in Figure 2.6.

Graph separation

We would like to forge a general link between graph separation and assertions of conditional independence. Doing so would allow us to use a graph-search algorithm to answer queries regarding conditional independence.

Happily, the graphs in Figure 2.4 and Figure 2.5 exhibit situations in which naive graph separation corresponds directly to conditional independence. Thus, as shown in Figure 2.4(b), shading the Y node blocks the path from X to Z , and this can be interpreted in terms of the conditional independence of X and Z given Y . Similarly, in Figure 2.5(b), the shaded Y node blocks the path from X to Z , and this can be interpreted in terms of the conditional independence of X and Z given Y .

On the other hand, the graph in Figure 2.6 involves a case in which naive graph separation and conditional independence are opposed. It is when the node Y is unshaded that X and Z are independent; when Y is shaded they become dependent. If we are going to use graph-theoretic ideas to answer queries about conditional independence, we need to pay particular attention to this case.

The solution is straightforward. Rather than relying on “naive” separation, we define a new notion of separation that is more appropriate to our purposes. This notion is known as *d-separation*, for “directed separation.” We provide a formal discussion of d-separation in Chapter 16; in the current chapter we provide a simple operational definition of d-separation in terms of the Bayes ball algorithm.

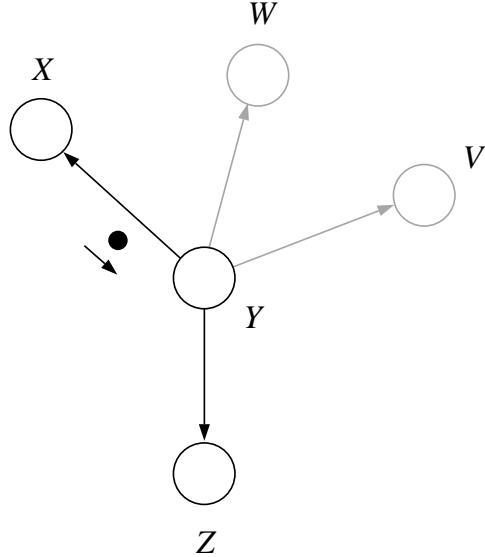


Figure 2.8: We develop a set of rules to specify what happens when a ball arrives from a node X at a node Y , en route to a node Z .

The Bayes ball algorithm

The problem that we wish to solve is to decide whether a given conditional independence statement, $X_A \perp\!\!\!\perp X_B | X_C$, is true for a directed graph \mathcal{G} . Formally this means that the statement holds for every distribution that factors according to \mathcal{G} , but let us not worry about formal issues for now, and let our intuition—aided by the three canonical graphs that we have already studied—help us to define an algorithm to decide the question.

The algorithm is a “reachability” algorithm: we shade the nodes X_C , place a ball at each of the nodes X_A , let the balls bounce around the graph according to a set of rules, and ask whether any of the balls reach one of the nodes in X_B . If none of the balls reach X_B , then we assert that $X_A \perp\!\!\!\perp X_B | X_C$ is true. If a ball reaches X_B then we assert that $X_A \perp\!\!\!\perp X_B | X_C$ is not true.

The basic problem is to specify what happens when a ball arrives at a node Y from a node X , en route to a node Z (see Figure 2.8). Note that we focus on a particular candidate destination node Z , ignoring the other neighbors that Y may have. (We will be trying all possible neighbors, but we focus on one at a time). Note also that the balls are allowed to travel in either direction along the edges of the graph.

We specify these rules by making reference to our three canonical graphs. In particular, referring to Figure 2.4, suppose that ball arrives at Y from X along an arrow oriented from X to Y , and we are considering whether to allow the ball to proceed to Z along an arrow oriented from Y to Z . Clearly, if the node Y is shaded, we do not want the ball to be able to reach Z , because $X \perp\!\!\!\perp Z | Y$ for this graph. Thus we require the ball to be “blocked” in this case. Similarly, if a ball arrives at Y from Z , we do not allow the ball to proceed to X ; again the ball is blocked. We summarize these rules with the diagram in Figure 2.9(a).

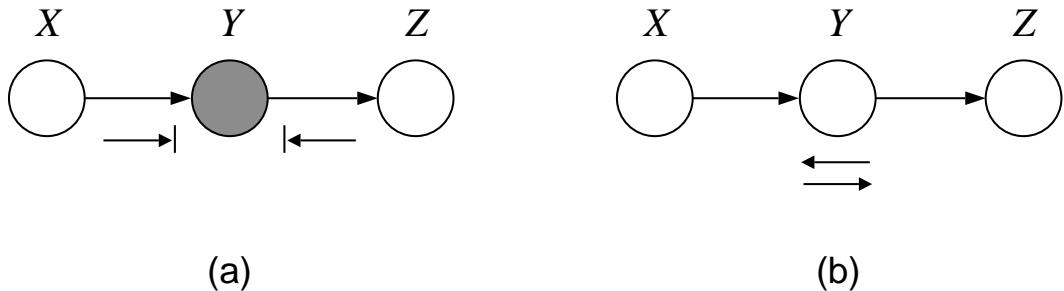


Figure 2.9: The rules for the case of one incoming arrow and one outgoing arrow. (a) When the middle node is shaded, the ball is blocked. (b) When the middle node is unshaded, the ball passes through.

On the other hand, if Y is not shaded, then we want to allow the ball to reach Z from X (and similarly X from Z), because we do not want to assert conditional independence in this case. Thus we have the diagram in Figure 2.9(b), which shows the ball “passing through” when Y is not shaded.

Similar considerations apply to the graph in Figure 2.5, where the arrows are oriented outward from the node Y . Once again, if Y is shaded we do not want the ball to pass between X and Z , thus we require it to be blocked at Y . On the other hand, if Y is unshaded we allow the ball to pass through. These rules are summarized in Figure 2.10.

Finally, we consider the graph in Figure 2.6 in which both of the arrows are oriented towards node Y (this is often referred to as a “v-structure”). Here we simply reverse the rules. Thus, if Y is not shaded we require the ball to be blocked, reflecting the fact that X and Z are marginally independent. On the other hand, if Y is shaded we allow the ball to pass through, reflecting the fact that we do not assert that X and Z are conditionally independent given Y . The rules for this graph are given in Figure 2.11.

We also intend for these rules to apply to the case in which the source node and the destination node (X and Z , respectively) are the same. That is, when a ball arrives at a node, we consider each possible outgoing edge in turn, including the edge the ball arrives on.

Consider first the case in which the ball arrives along an edge that is oriented from X to Y . In this case, the situation is effectively one in which a ball arrives on the head of an arrow and departs on the head of an arrow. This situation is covered by Figure 2.11. We see that the ball should be blocked if the node is unshaded and should “pass through” if the node is shaded, a result that is summarized in Figure 2.12. Note that the action of “passing through” is better described in this case as “bouncing back.”

The remaining situation is the one in which the ball arrives along an edge that is oriented from Y to X . The ball arrives on the tail of an arrow and departs on the tail of an arrow, a situation which is covered by Figure 2.10. We see that the ball should be blocked if the node is shaded and should bounce back if the node is unshaded, a result that is summarized in Figure 2.13.

Let us consider some examples. Figure 2.14 shows a chain-structured graphical model (a Markov chain) on a set of nodes $\{X_1, X_2, \dots, X_n\}$. The basic conditional independencies for this graph (cf.

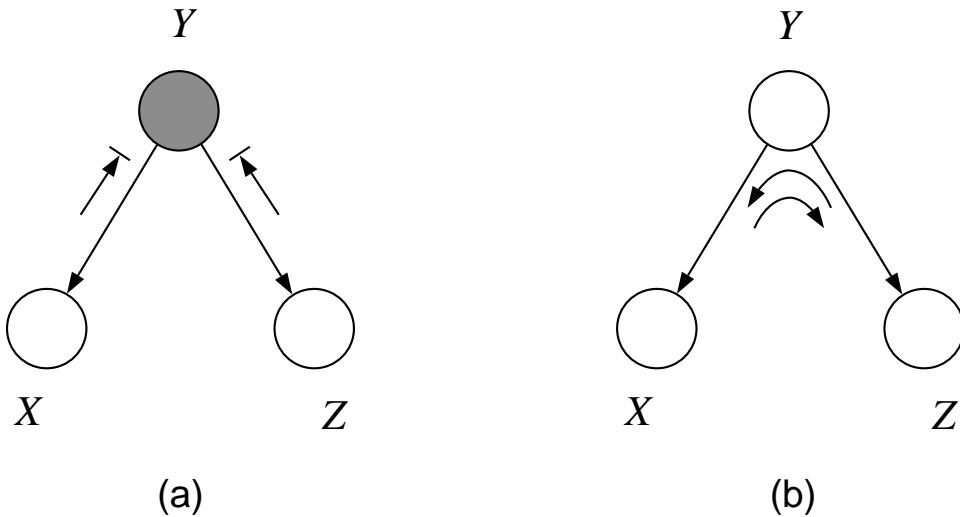


Figure 2.10: The rules for the case of two outgoing arrows. (a) When the middle node is shaded, the ball is blocked. (b) When the middle node is unshaded, the ball passes through.

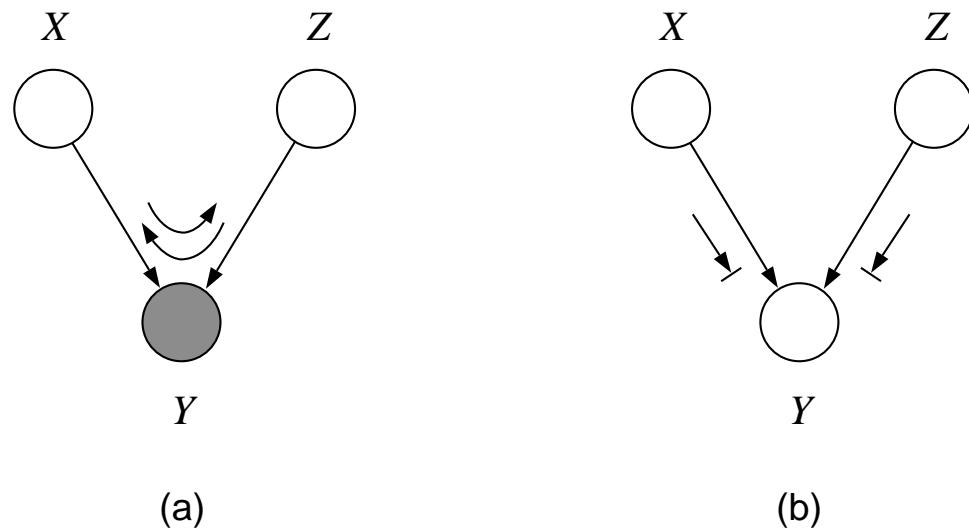


Figure 2.11: The rules for the case of two outgoing arrows. (a) When the middle node is shaded, the ball passes through. (b) When the middle node is unshaded, the ball is blocked.



Figure 2.12: The rules for this case follow from the rules in Figure 2.11. (a) When the ball arrives at an unshaded node, the ball is blocked. (b) When the ball arrives at a shaded node, the ball “passes through,” which effectively means that it bounces back.



Figure 2.13: The rules for this case follow from the rules in Figure 2.10. (a) When the ball arrives at an unshaded node, the ball “passes through,” which effectively means that it bounces back. (b) When the ball arrives at a shaded node, the ball is blocked.

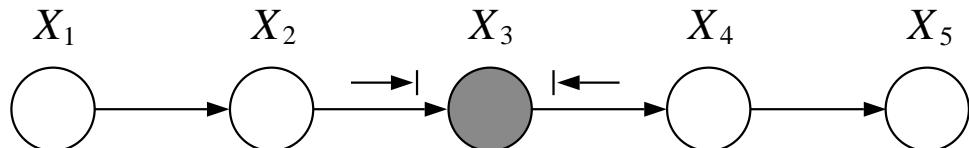


Figure 2.14: The separation of X_3 from X_1 , given its parent, X_2 , is a basic independence statement for this graph. But conditioning on X_3 also separates any subset of X_1, X_2 from any subset of X_4, X_5 , and all of these separations also correspond to conditional independencies.

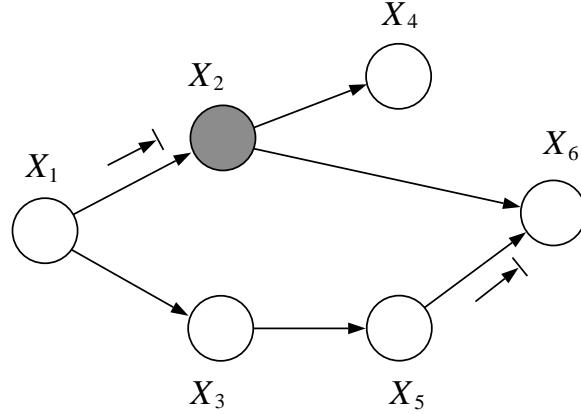


Figure 2.15: A ball arriving at X_2 from X_1 is blocked from continuing on to X_4 . Also, a ball arriving at X_6 from X_5 is blocked from continuing on to X_2 .

Eq. (2.8)) are the conditional independencies:

$$X_{i+1} \perp\!\!\!\perp \{X_1, X_2, \dots, X_{i-1}\} \mid X_i. \quad (2.32)$$

There are, however, many other conditional independencies that are implied by this basic set, such as:

$$X_1 \perp\!\!\!\perp X_5 \mid X_4, \quad X_1 \perp\!\!\!\perp X_5 \mid X_2, \quad X_1 \perp\!\!\!\perp X_5 \mid \{X_2, X_4\}, \quad (2.33)$$

each of which can be established from algebraic manipulations starting from the definition of the joint probability. Indeed, in general we can obtain the conditional independence of any subset of “future” nodes from any subset of “past” nodes given any subset of nodes that separates these subsets. This is clearly the set of conditional independence statements picked out by the Bayes ball algorithm; the ball is blocked when it arrives at X_3 from either the left or the right.

Consider the graph in Figure 2.1 and consider the conditional independence $X_4 \perp\!\!\!\perp \{X_1, X_3\} \mid X_2$ which we demonstrated to hold for this graph (this is one of the basic set of conditional independencies for this graph; recall Eqs. 2.9 through eq:example-set-of-basic-CI). Using the Bayes ball approach, let us consider whether it is possible for a ball to arrive at node X_4 from either node X_1 or node X_3 , given that X_2 is shaded (see Figure 2.15). To arrive at X_4 , the ball must pass through X_2 . One possibility is to arrive at X_2 from X_1 , but the path through to X_4 is blocked because of Figure 2.9(a). The other possibility is to arrive at X_2 via X_6 . However, any ball arriving at X_6 must do so via X_5 , and such a ball is blocked at X_6 because of Figure 2.11(b).

Note that balls can also bounce back at X_2 and X_6 , but this provides no help with respect to arriving at X_4 .

We claimed in Section 2.1.1 that $X_1 \perp\!\!\!\perp X_6 \mid \{X_2, X_3\}$, a conditional independence that is not in the basic set. Consider a ball starting at X_1 and traveling to X_3 (see Figure 2.16). Such a ball cannot pass through to X_5 because of Figure 2.9(a). Similarly, a ball cannot pass from X_1 through X_2 (to either X_4 or X_6) because of Figure 2.9(a).

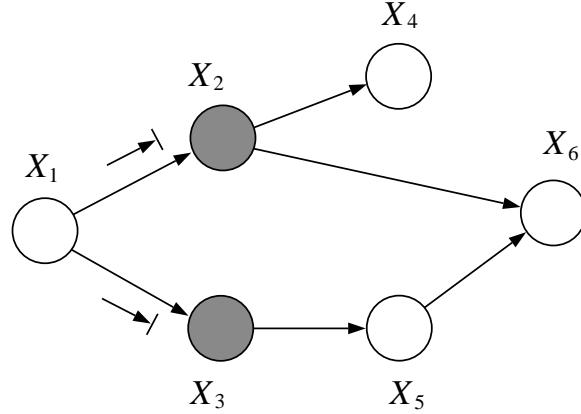


Figure 2.16: A ball cannot pass through X_2 to X_6 nor through X_3 .

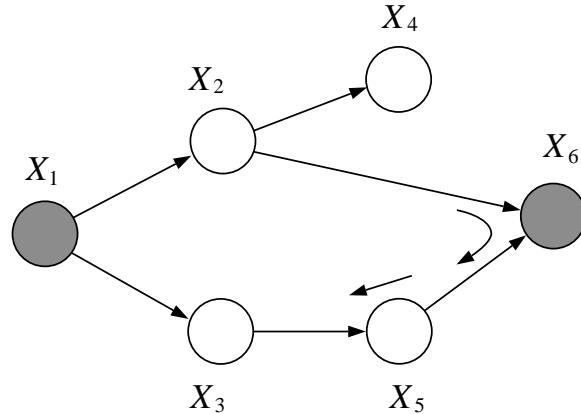


Figure 2.17: A ball can pass from X_2 through X_6 to X_5 , and thence to X_3 .

We also claimed in Section 2.1.1 that it is not the case that $X_2 \perp\!\!\!\perp X_3 \mid \{X_1, X_6\}$. To establish this claim we note that a ball can pass through X_2 to X_6 because of Figure 2.9(b), and (see Figure 2.17) can then pass from through X_6 to X_5 , on the basis of Figure 2.11(a). The ball then passes through X_5 and arrives at X_3 . Intuitively (and loosely), the observation of X_6 implies the possibility of an “explaining-away” dependency between X_2 and X_5 . Clearly X_5 and X_3 are dependent, and thus X_2 and X_3 are dependent.

Finally, consider again the scenario with Alice and Bob, and suppose that Bob does not actually observe that Alice fails to show at the hour that he expects her. Suppose instead that Bob is an important executive and there is a security guard for Bob’s building who reports to Bob whether a guest has arrived or not. We augment the model to include a node **report** for the security guard’s report and, as shown in Figure 2.18, we hang this node off of the node **late**. Now observation of **report** is essentially as good as observation of **late**, particularly if we believe that the security guard is reliable. That is, we should still have **aliens** $\perp\!\!\!\perp$ **watch**, and moreover we should not assert

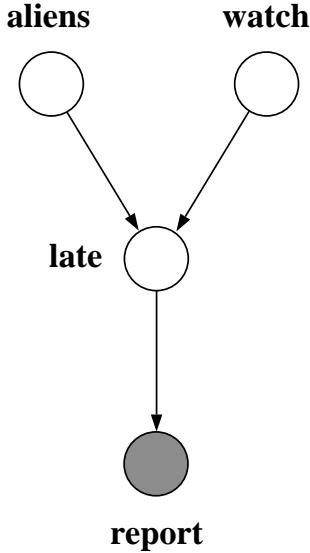


Figure 2.18: An extended graphical model for the Bob-Alice scenario, including a node **report** for the security guard's report.

aliens $\perp\!\!\!\perp$ watch | report. That is, if the security guard reports that Alice has not arrived, then Bob worries about aliens and subsequently has his worries alleviated when he realizes that he has forgotten about daylight savings time.

This pattern is what the Bayes ball algorithm delivers. Consider first the marginal independence **aliens $\perp\!\!\!\perp$ watch**. As can be verified from Figure 2.19(a), a ball that starts at **aliens** is blocked from passing through **late** directly to **watch**. Moreover, although a ball can pass through **late** to **report**, such a ball dies at **report**. Thus the ball cannot arrive at **watch**.

Consider now the situation when **report** is observed (Figure 2.19(b)). As before a ball that starts at **aliens** is blocked from passing through **late** directly to **watch**; however, a ball can pass through **late** to **report**. At this point Figure 2.12(b) implies that the ball bounces back at **report**. The ball can then pass through **late** on the path from **report** to **watch**. Thus we cannot conclude independence of **aliens** and **watch** in the case that **report** is observed.

Some further thought will show that it suffices for any descendant of **late** to be observed in order to enable the explaining-away mechanism and render **aliens** and **watch** dependent.

Remarks

We hope that the reader agrees that the Bayes ball algorithm is a simple, intuitively-appealing algorithm for answering conditional independence queries. Of course, we have not yet provided a fully-specified algorithm, because there are many implementational details to work out, including how to represent multiple balls when X_A and X_B are not singleton sets, how to make sure that the algorithm considers all possible paths in an efficient way, how to make sure that the algorithm doesn't loop, etc. But these details are just that—details—and with a modicum of effort the reader

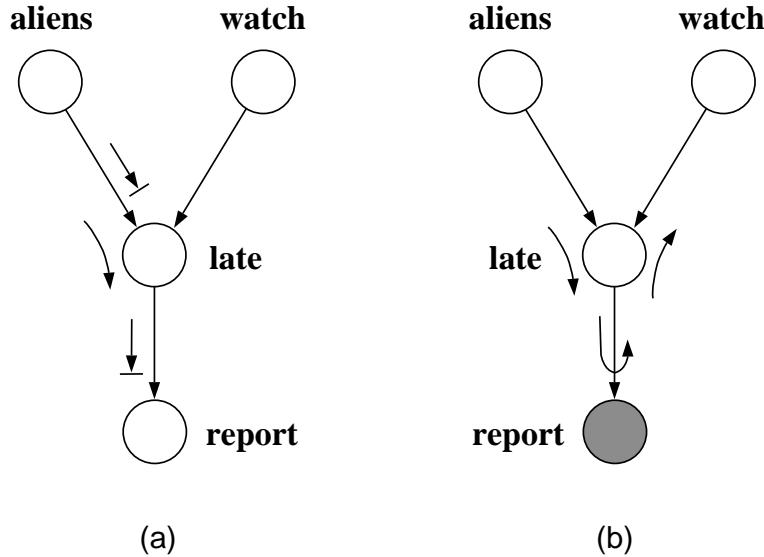


Figure 2.19: (a) A ball cannot pass from **aliens** to **watch** when no observations are made on **late** or **report**. (b) A ball can pass from **aliens** to **watch** when **report** is observed.

can work out such an implementation. Our main interest in the Bayes ball algorithm is to provide a handy tool for quick evaluation of conditional independence queries, and to provide concrete support for the more formal discussion of conditional independence that we undertake in the next section.

2.1.3 Characterization of directed graphical models

A key idea that has emerged in this chapter is that graphical model is associated with a *family* of probability distributions. Moreover, as we now discuss, this family can be characterized in two equivalent ways.

Let us define two families and then show that they are equivalent. Actually we defer the proof of equivalence until Chapter 16, but we state the theorem here and discuss its consequences.

The first family is defined via the definition of joint probability for directed graphs, which we repeat here for convenience. Thus for a directed graph \mathcal{G} , we have:

$$p(x_1, x_2, \dots, x_n) \triangleq \prod_{i=1}^n p(x_i \mid x_{\pi_i}). \quad (2.34)$$

Let us now consider ranging over all possible numerical values for the local conditional probabilities $\{p(x_i | x_{\pi_i})\}$, imposing only the restriction that these functions are nonnegative and normalized. For discrete variables this would involve ranging over all possible real-valued tables on nodes x_i and their parents. While in practice, we often want to choose simplified parameterizations instead of these tables, for the general theory we must range over all possible tables.

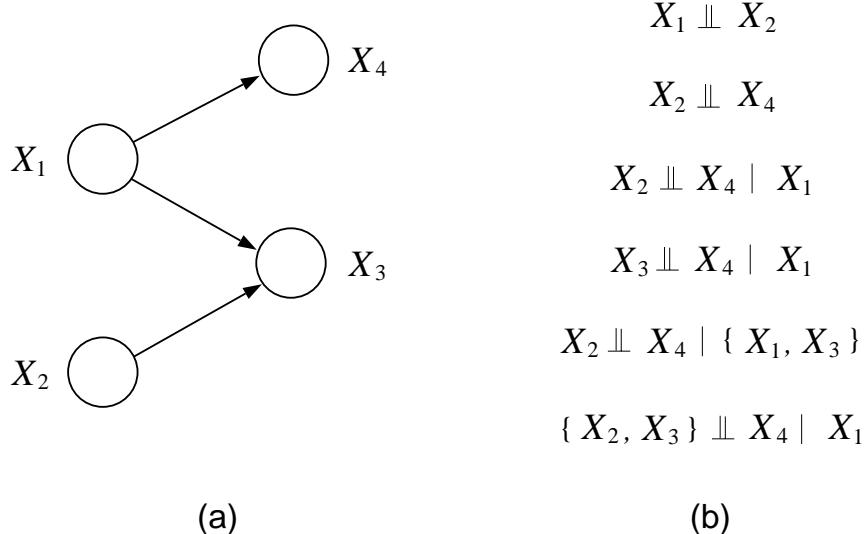


Figure 2.20: The list in (b) shows all of the conditional independencies that hold for the graph in (a).

For each choice of numerical values for the local conditional probabilities we obtain a particular probability distribution $p(x_1, \dots, x_n)$. Ranging over all such choices we obtain a family of distributions that we refer to as \mathcal{D}_1 .

Let us now consider an alternative way to generate a family of probability distributions associated with a graph \mathcal{G} . In this approach we will make no use of the numerical parameterization of the joint probability in Eq. (2.34)—this approach will be more “qualitative.”

Given a graph \mathcal{G} we can imagine making a list of all of the conditional independence statements that characterize the graph. To do this, imagine running the Bayes ball algorithm for all triples of subsets of nodes in the graph. For any given triple X_A , X_B and X_C , the Bayes ball algorithm tells us whether or not $X_A \perp\!\!\!\perp X_B \mid X_C$ should be included in the list associated with the graph.

For example, Figure 2.20 shows a graph, and all of its associated conditional independence statements. In general such lists can be significantly longer than the list in this example, but they are always finite.

Now consider all possible joint probability distributions $p(x_1, \dots, x_n)$, where we make no restrictions at all. Thus, for discrete variables, we consider all possible n -dimensional tables. For each such distribution, imagine testing the distribution against the list of conditional independencies associated with the graph \mathcal{G} . Thus, for each conditional independence statement in the list, we test whether the distribution factorizes as required. If it does, move to the next statement. If it does not, throw out this distribution and try a new distribution. If a distribution passes all of the tests in the list, we include that distribution in a family that we denote as \mathcal{D}_2 .

In Chapter 16, we state and prove a theorem that shows that the two families \mathcal{D}_1 and \mathcal{D}_2 are the same family. This theorem, and an analogous theorem for undirected graphs, provide a strong and important link between graph theory and probability theory and are at the core of the graphical

model formalism. They show that the characterizations of probability distributions via numerical parameterization and conditional independence statements are one and the same, and allow us to use these characterizations interchangeably in analyzing models and defining algorithms.

2.2 Undirected graphical models

The world of graphical models divides into two major classes—those based on directed graphs and those based on undirected graphs.³ In this section we discuss undirected graphical models, also known as *Markov random fields*, and carry out a development that parallels our discussion of the directed case. Thus we will present a factorized parameterization for undirected graphs, a conditional independence semantics, and an algorithm for answering conditional independence queries. There are many similarities to the directed case—and much of our earlier work on directed graphs carries over—but there are interesting and important differences as well.

An undirected graphical model is a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where \mathcal{V} is a set of nodes that are in one-to-one correspondence with a set of random variables, and where \mathcal{E} is a set of undirected edges. The random variables can be scalar-valued or vector-valued, discrete or continuous. Thus we will be concerned with graphical representations of a joint probability distribution, $p(x_1, x_2, \dots, x_n)$ —a mass function in the discrete case and a density function in the continuous case.

2.2.1 Conditional independence

As we saw in Section 2.1.3, there are two equivalent characterizations of the class of joint probability distributions associated with a directed graph. Our presentation of directed graphical models began (in Section 2.1) with the factorized parameterization and subsequently motivated the conditional independence characterization. We could, however, have turned this discussion around and started with a set of conditional independence axioms, subsequently deriving the parameterization. In the case of undirected graphs, indeed, this latter approach is the one that we will take. For undirected graphs, the conditional independence semantics is the more intuitive and straightforward of the two (equivalent) characterizations.

To specify the conditional independence properties of a graph, we must be able to say whether $X_A \perp\!\!\!\perp X_C | X_B$ is true for the graph, for arbitrary index subsets A , B , and C . For directed graphs we defined the conditional independence properties operationally, via the Bayes ball algorithm (we provide a corresponding declarative definition in Chapter 16). For undirected graphs we go straight to the declarative definition.

We say that X_A is independent of X_C given X_B if the set of nodes X_B separates the nodes X_A from the nodes X_C , where by “separation” we mean naive graph-theoretic separation (see Figure 2.21). Thus, if every path from a node in X_A to a node in X_C includes at least one node in X_B , then we assert that $X_A \perp\!\!\!\perp X_C | X_B$ holds; otherwise we assert that $X_A \perp\!\!\!\perp X_C | X_B$ does not hold.

³There is also a generalization known as *chain graphs* that subsumes both classes. We will discuss chain graphs in Chapter ??.

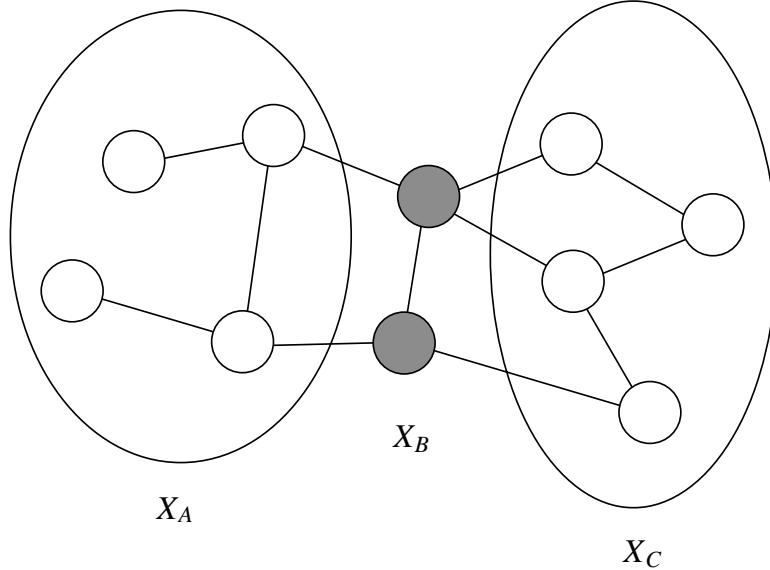


Figure 2.21: The set X_B separates X_A from X_C . All paths from X_A to X_C pass through X_B .

As before, the meaning of the statement “ $X_A \perp\!\!\!\perp X_C | X_B$ holds for a graph \mathcal{G} ” is that every member of the family of probability distributions associated with \mathcal{G} exhibits that conditional independence. On the other hand, the statement “ $X_A \perp\!\!\!\perp X_C | X_B$ does not hold for a graph \mathcal{G} ” means—in its strong form—that some distributions in the family associated with \mathcal{G} do not exhibit that conditional independence.

Given this definition, it is straightforward to develop an algorithm for answering conditional independence queries for undirected graphs. We simply remove the nodes X_B from the graph and ask whether there are any paths from X_A to X_C . This is a “reachability” problem in graph theory, for which standard search algorithms provide a solution.

Comparative semantics

Is it possible to reduce undirected models to directed models, or vice versa? To see that this is not possible in general, consider Figure 2.22.

In Figure 2.22(a) we have an undirected model that is characterized by the conditional independence statements $X \perp\!\!\!\perp Y | \{W, Z\}$ and $W \perp\!\!\!\perp Z | \{X, Y\}$. If we try to represent this model in a directed graph on the same four nodes, we find that we must have at least one node in which the arrows are inward-pointing (a “v-structure”). (Recall that our graphs are acyclic). Suppose without loss of generality that this node is Z . By the conditional independence semantics of directed graphs, we have $X \perp\!\!\!\perp Y | W$, and we do not have $X \perp\!\!\!\perp Y | \{W, Z\}$. We are unable to represent both conditional independence statements, $X \perp\!\!\!\perp Y | \{W, Z\}$ and $W \perp\!\!\!\perp Z | \{X, Y\}$, in the directed formalism.

On the other hand, in Figure 2.22(b) we have a directed graph characterized by the singleton independence statement $X \perp\!\!\!\perp Y$. No undirected graph on three nodes is characterized by this

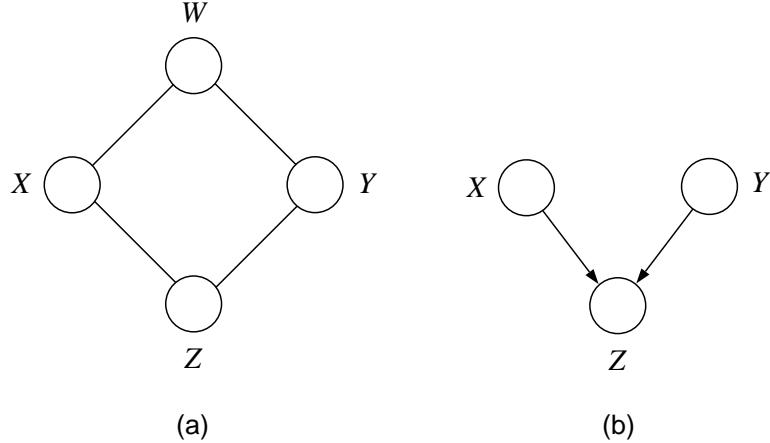


Figure 2.22: (a) An undirected graph whose conditional independence semantics cannot be captured by a directed graph on the same nodes. (b) A directed graph whose conditional independence semantics cannot be captured by an undirected graph on the same nodes.

singleton set. A missing edge in an undirected graph only between X and Y captures $X \perp\!\!\!\perp Y | Z$, not $X \perp\!\!\!\perp Y$. An additional missing edge between X and Z captures $X \perp\!\!\!\perp Y$, but implies $X \perp\!\!\!\perp Z$.

We will show in Chapter 16 that there are some families of probability distributions that can be represented with either directed or undirected graphs. There is no good reason to restrict ourselves to these families, however. In general, directed models and undirected models are different modeling tools, and have different strengths and weaknesses. The two together provide modeling power beyond that which could be provided by either alone.

2.2.2 Parameterization

As in the case of directed graphs, we would like to obtain a “local” parameterization for undirected graphical models. For directed graphs the parameterization was based on local conditional probabilities, where “local” had the interpretation of a set $\{i, \pi_i\}$ consisting of a node and its parents. The definition of the joint probability as a product of such local probabilities was motivated via the chain rule of probability theory.

In the undirected case it is rather more difficult to utilize conditional probabilities to represent the joint. One possibility would be to associate to each node the conditional probability of the node given its neighbors. This approach falls prey to a major consistency problem, however—it is hard to ensure that the conditional probabilities at different nodes are consistent with each other and thus with a single joint distribution. We are not able to choose these functions independently and arbitrarily, and this poses problems both in theory and in practice.

A better approach turns out to be to abandon conditional probabilities altogether. By so doing we will lose the ability to give a local probabilistic interpretation to the functions used to represent the joint probability, but we will retain the ability to choose these functions independently and arbitrarily, and we will retain the all-important representation of the joint as a *product* of local

functions.

A key problem is to decide the domain of the local functions; in essence, to decide the meaning of “local” for undirected graphs. It is here that the discussion of conditional independence in the previous section is helpful. In particular, consider a pair of nodes X_i and X_j that are not linked in the graph. The conditional independence semantics imply that these two nodes are conditionally independent given all of the other nodes in the graph (because upon removing this latter set there can be no paths from X_i to X_j). Thus it must be possible to obtain a factorization of the joint probability that places x_i and x_j in different factors. This implies that we can have no local function that depends on both x_i and x_j in our representation of the joint. Such a local function, say $\psi(x_i, x_j, x_k)$, would not factorize with respect to x_i and x_j in general—recall that we are assuming that the local functions can be chosen arbitrarily.

Recall that a *clique* of a graph is a fully-connected subset of nodes. Our argument thus far has suggested that the local functions should not be defined on domains of nodes that extend beyond the boundaries of cliques. That is, if X_i and X_j are not directly connected, they do not appear together in any clique, and correspondingly there should be no local function that refers to both nodes. We now consider the flip side of the coin. Should we allow arbitrary functions that are defined on all of the cliques? Indeed, an interpretation of the edges that are present in the graph in terms of “dependence” suggests that we should. We have not defined dependence, but heuristically, dependence is the “absence of independence” in one or more of the distributions associated with a graph. If X_i and X_j are linked, and thus appear together in a clique, we can achieve dependence between them by defining a function on that clique.

The *maximal cliques* of a graph are the cliques that cannot be extended to include additional nodes without losing the property of being fully connected. Given that all cliques are subsets of one or more maximal cliques, we can restrict ourselves to maximal cliques without loss of generality. Thus, if X_1 , X_2 , and X_3 form a maximal clique, then an arbitrary function $\psi(x_1, x_2, x_3)$ already captures all possible dependencies on these three nodes; we gain no generality by also defining functions on sub-cliques such as $\{X_1, X_2\}$ or $\{X_2, X_3\}$.⁴

In summary, our arguments suggest that the meaning of “local” for undirected graphs should be “maximal clique.” More precisely, the conditional independence properties of undirected graphs imply a representation of the joint probability as a product of local functions defined on the maximal cliques of the graph. This argument is in fact correct, and we will establish it rigorously in Chapter 16. Let us proceed to make the definition and explore some of its consequences.

Let C be a set of indices of a maximal clique in an undirected graph G , and let \mathcal{C} be the set of all such C . A *potential function*, $\psi_{X_C}(x_C)$, is a function on the possible realizations x_C of the maximal clique X_C .

Potential functions are assumed to be nonnegative, real-valued functions, but are otherwise arbitrary. This arbitrariness is convenient, indeed necessary, for our general theory to go through, but it also presents a problem. There is no reason for a product of arbitrary functions to be

⁴While there is no need to consider non-maximal cliques in developing the general theory relating conditional independence and factorization—our topic in this section—in practice it is often convenient to work with potentials on non-maximal cliques. This issue will return in Section 2.3 and in later chapters. Let us define joint probabilities in terms of maximal cliques for now, but let us be prepared to relax this definition later.

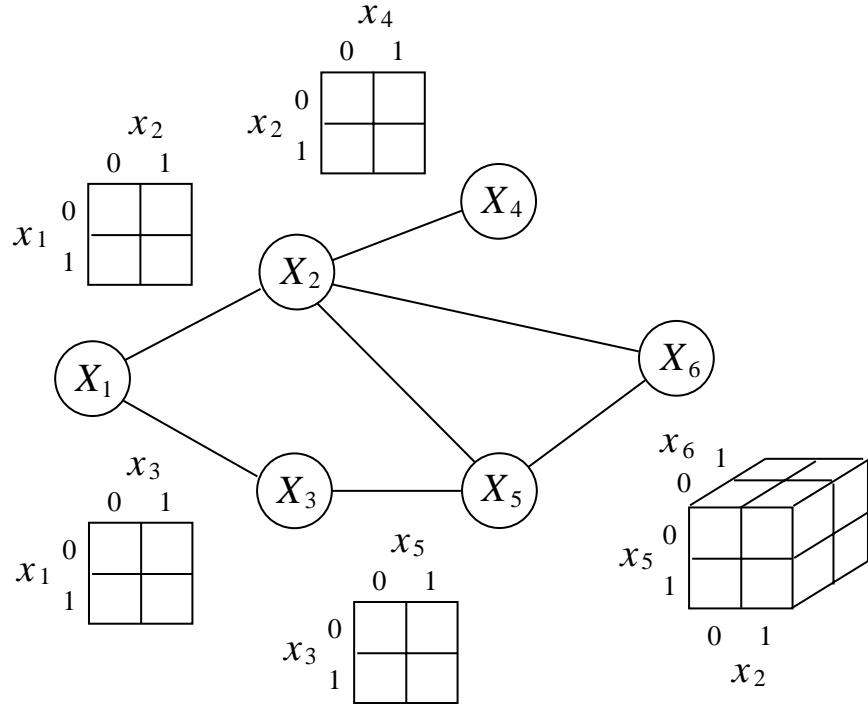


Figure 2.23: The maximal cliques in this graph are $\{X_1, X_2\}$, $\{X_1, X_3\}$, $\{X_2, X_4\}$, $\{X_3, X_5\}$, and $\{X_2, X_5, X_6\}$. Letting all nodes be binary, we represent a joint distribution on the graph via the potential tables that are displayed.

normalized and thus define a joint probability distribution. This is a bullet which we simply have to bite if we are to achieve the desired properties of arbitrary, independent potentials and a product representation for the joint.

Thus we define:

$$p(x) \triangleq \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_{X_C}(x_C), \quad (2.35)$$

where Z is the normalization factor:

$$Z \triangleq \sum_x \prod_{C \in \mathcal{C}} \psi_{X_C}(x_C), \quad (2.36)$$

obtained by summing the product in Eq. (2.35) over all assignments of values to the nodes X .

An example is shown in Figure 2.23. The nodes in this example are assumed discrete, and thus tables can be used to represent the potential functions. An overall configuration x picks out subvectors x_C , which determine particular cells in each of the potential tables. Taking the product of the numbers in these cells yields an unnormalized representation of the joint probability $p(x)$.

The normalization factor Z is obtained by summing over all configurations x . There are an exponential number of such configurations and it is unrealistic to try to perform such a sum by

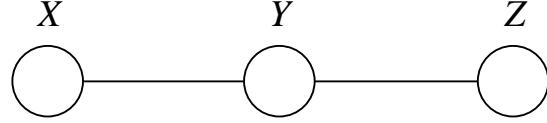


Figure 2.24: An undirected representation of a three-node Markov chain. The conditional independence associated with this graph is $X \perp\!\!\!\perp Z | Y$.

naively enumerating all of the summands. Note, however, that the expression being summed over is a factored expression, in which each factor refers to a local set of variables, and thus we can exploit the distributive law. This is an issue that is best discussed in the context of the more general discussion of probabilistic inference, and we return to it in Chapter 3.

Note, however, that we do not necessarily have to calculate Z . In particular, recall that a conditional probability is a ratio of two marginal probabilities. The factor Z appears in both of the marginal probabilities, and cancels when we take the ratio. Thus we calculate conditionals by summing across unnormalized probabilities—the numerator in Eq. (2.35)—and taking the ratio of these sums.

The interpretation of potential functions

Although local conditional probabilities do not provide a satisfactory approach to the parameterization of undirected models, it might be thought that marginal probabilities could be used instead. Thus, why not replace the potential functions $\psi_{X_C}(x_C)$ in Eq. (2.35) with marginal probabilities $p(x_C)$?

An example will readily show that this approach is infeasible. Consider the model shown in Figure 2.24. The conditional independence that is associated with this graph is $X \perp\!\!\!\perp Z | Y$. This independence statement implies (by definition) that the joint must factorize as:

$$p(x, y, z) = p(y)p(x | y)p(z | y). \quad (2.37)$$

The cliques in Figure 2.24 are $\{X, Y\}$ and $\{Y, Z\}$. We can multiply the first two factors in Eq. (2.37) together to obtain a potential function $p(x, y)$ on the first clique, leaving $p(z | y)$ as the potential function on the second clique. Alternatively, we can multiply $p(z | y)$ by $p(y)$ to yield a potential $p(y, z)$ on the second clique, leaving $p(x | y)$ as the potential on the first clique. Thus we can obtain a factorization in which one of the potentials is a marginal probability, and the other is a conditional probability. But we are unable to obtain a representation in which both potentials are marginal probabilities. That is:

$$p(x, y, z) \neq p(x, y)p(y, z). \quad (2.38)$$

In fact, it is not hard to see that $p(x, y, z) = p(x, y)p(y, z)$ implies $p(y) = 0$ or $p(y) = 1$, and that this representation is thus a rather limited and unnatural one.

In general, potential functions are neither conditional probabilities nor marginal probabilities, and in this sense they do not have a local probabilistic interpretation. On the other hand, potential functions do often have a natural interpretation in terms of pre-probabilistic notions such

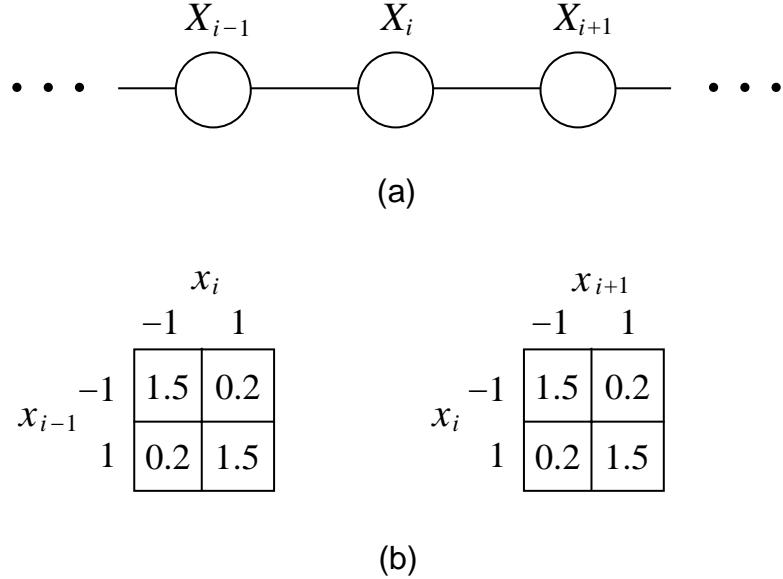


Figure 2.25: (a) A chain of binary random variables X_i , where $X_i \in \{-1, 1\}$. (b) A set of potential tables that encode a preference for neighboring variables to have the same values.

as “agreement,” “constraint,” or “energy,” and such interpretations are often useful in choosing an undirected model to represent a real-life domain. The basic idea is that a potential function favors certain local configurations of variables by assigning them a larger value. The global configurations that have high probability are, roughly, those that satisfy as many of the favored local configurations as possible.

Consider a set of binary random variables, $X_i \in \{-1, 1\}, i = 0, \dots, n$, arrayed on a one-dimensional lattice as shown in Figure 2.25(a). In physics, such lattices are used to model magnetic behavior of crystals, where the binary variables have an interpretation as magnetic “spins.” All else being equal, if a given spin X_i is “up”; that is, if $X_i = 1$, then its neighbors X_{i-1} and X_{i+1} are likely to be “up” as well. We can easily encode this in a potential function, as shown in Figure 2.25(b). Thus, if two neighboring spins agree, that is, if $X_i = 1$ and $X_{i-1} = 1$, or if $X_i = -1$ and $X_{i-1} = -1$, we obtain a large value for the potential on the clique $\{X_{i-1}, X_i\}$. If the spins disagree we obtain a small value.

The fact that potentials must be nonnegative can be inconvenient, and it is common to exploit the fact that the exponential function, $f(x) = \exp(x)$, is a nonnegative function, to represent potentials in an unconstrained form. We let:

$$\psi_{X_C}(x_C) = \exp\{-H_C(x_C)\}, \quad (2.39)$$

for a real-valued function $H_C(x_C)$, where the negative sign is a standard convention. Thus if we range over arbitrary $H_C(x_C)$, we can range over legal potentials.

The exponential representation has another useful feature. In particular, products of exponen-

tials behave nicely, and from Eq. (2.35) we obtain:

$$p(x) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \exp\{-H_C(x_C)\} \quad (2.40)$$

$$= \frac{1}{Z} \exp\left\{-\sum_{C \in \mathcal{C}} H_C(x_C)\right\} \quad (2.41)$$

as an equivalent representation of the joint probability for undirected models. The sum in the latter expression is generally referred to as the “energy”:

$$H(x) \triangleq \sum_{C \in \mathcal{C}} H_C(x_C) \quad (2.42)$$

and we have represented the joint probability of an undirected graphical model as a Boltzmann distribution:

$$p(x) = \frac{1}{Z} \exp\{-H(x)\}. \quad (2.43)$$

Without going too far astray into the origins of the Boltzmann representation in statistical physics, let us nonetheless note that the representation of a model in terms of energy, and in particular the representation of the total energy as a sum over local contributions to the energy, is exceedingly useful. Many physical theories are specified in terms of energy, and the Boltzmann distribution provides a translation from energies into probabilities.

Quite apart from any connection to physics, the undirected graphical model formalism is often quite useful in domains in which global constraints on probabilities are naturally decomposable into sets of local constraints, and the undirected representation is apt at capturing such situations.

2.2.3 Characterization of undirected graphical models

In Section 2.2.3 we discussed a theorem that shows that the two different characterizations of the family of probability distributions associated with a directed graphical model—one based on local conditional probabilities and the other based on conditional independence assertions—were the same. A formally identical theorem holds for undirected graphs.

For a given undirected graph \mathcal{G} , we define a family of probability distributions, \mathcal{U}_1 , by ranging over all possible choices of positive potential functions on the maximal cliques of the graph.

We define a second family of probability distributions, \mathcal{U}_2 , via the conditional independence assertions associated with \mathcal{G} . Concretely, we make a list of all of the conditional independence statements, $X_A \perp\!\!\!\perp X_B | X_C$, asserted by the graph, by assessing whether the subset of nodes X_A is separated from X_B when the nodes X_C are removed from the graph. A probability distribution is in \mathcal{U}_2 if it satisfies all such conditional independence statements, otherwise it is not.

In Chapter 16 we state and prove a theorem, the Hammersley-Clifford theorem, that shows that \mathcal{U}_1 and \mathcal{U}_2 are identical. Thus the characterization of probability distributions in terms of potentials on cliques and conditional independence are equivalent. As in the directed case, this is an important and profound link between probability theory and graph theory.

2.3 Parameterizations

We have introduced two kinds of graphical model representations in this chapter—directed graphical models and undirected graphical models. In each of these cases we have defined conditional independence semantics and corresponding parameterizations. Thus, in the directed case, we have:

$$p(x) \triangleq \prod_{i=1}^n p(x_i | x_{\pi_i}), \quad (2.44)$$

and in the undirected case, we have:

$$p(x) \triangleq \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_{X_C}(x_C). \quad (2.45)$$

By ranging over all possible conditional probabilities in Eq. (2.44) or all possible potential functions in Eq. (2.45) we obtain certain families of probability distributions, in particular exactly those distributions which respect the conditional independence statements associated with a given graph.

Conditional independence is an exceedingly useful constraint to impose on a joint probability distribution. In practical settings conditional independence can sometimes be assessed by domain experts, and in such cases it provides a powerful way to embed qualitative knowledge about the relationships among random variables into a model. Moreover, as we will discuss in the following chapter, the relationship between conditional independence and factorization allows the development of powerful general inference algorithms that use graph-theoretic ideas to compute marginal probabilities of interest. We often impose conditional independence as a rough, tentative assumption in a domain so as to be able to exploit the efficient inference algorithms and begin to learn something about the domain.

On the other hand, conditional independence is by no means the only kind of constraint that one can impose on a probabilistic model. Another large class of constraints arise from assumptions about the algebraic structure of the conditional probabilities or potential functions that define a model. In particular, rather than ranging over all possible conditional probabilities or potential functions, we may wish to range over a proper subset of these functions, thus defining a proper subset of the family of probability distributions associated with a graph. Thus, in practice we often work with *reduced parameterizations* that impose constraints on probability distributions beyond the structural constraints imposed by conditional independence.

We will present many examples of reduced parameterizations in later chapters. Let us briefly consider two such examples in the remainder of this section to obtain a basic appreciation of some of the issues that arise.

Directed graphical models require conditional probabilities, and if the number of parents of a given node is large, then the specification of the conditional probability can be problematic. Consider in particular the graph shown in Figure 2.26(a), where all of the variables are assumed binary (for simplicity), and where the number of parents of Y is assumed large. In particular, if Y has 50 parents, then ranging over “all possible conditional probabilities” means specifying 2^{50} numbers, one probability for each configuration of the parents. Clearly such a specification cannot

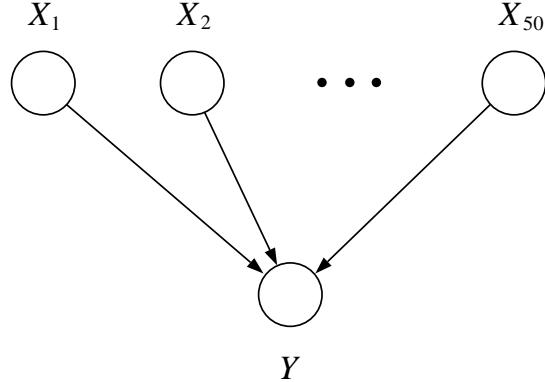


Figure 2.26: An example in which a node has many parents. In such a graph, a general specification of the local conditional probability distribution requires an impractically large number of parameters.

be stored on a computer, and, equally problematically, it would be impossible to collect enough data to be able to estimate these numbers with any degree of precision. We are forced to consider “reduced parameterizations.” One such parameterization, discussed in detail in Chapter 8, is the following:

$$p(Y = 1 | x) = f(\theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_m x_m), \quad (2.46)$$

for a given function $f(\cdot)$ whose range is the interval $(0, 1)$ (we will provide examples of such functions in Chapter 8). Here, we need only specify the 50 numbers θ_i to specify a distribution.

In general, we can consider directed graphical models in which each node is parameterized as shown in Eq. (2.46). The family of probability distributions associated with the model as a whole is that obtained by ranging over all possible values of θ_i in the defining conditional probabilities. This is a proper sub-family of the family of distributions associated with the graph.

If practical considerations often force us to work with reduced parameterizations, of what value is the general definition of “the family of distributions associated with a graph”? As we will see in Chapter 4 and Chapter 17, given a graph, efficient probabilistic inference algorithms can be defined that operate on the graph. These algorithms are based solely on the graph structure and are correct for any distribution that respects the conditional independencies encoded by the graph. Thus such algorithms are correct for any distribution in the family of distributions associated with a graph, including those in any proper sub-family associated with a reduced parameterization.

Similar issues arise in undirected models. Consider in particular the graph shown in Figure 2.27(a). From the point of view of independence, there is little to say—there are no independence assertions associated with this graph. Equivalently, the family of probability distributions associated with the graph is the set of all possible probability distributions on the three variables, obtained by ranging over all possible potential functions $\psi(x_1, x_2, x_3)$. Suppose, however, that we are interested in models in which the potential function is defined algebraically as a product of

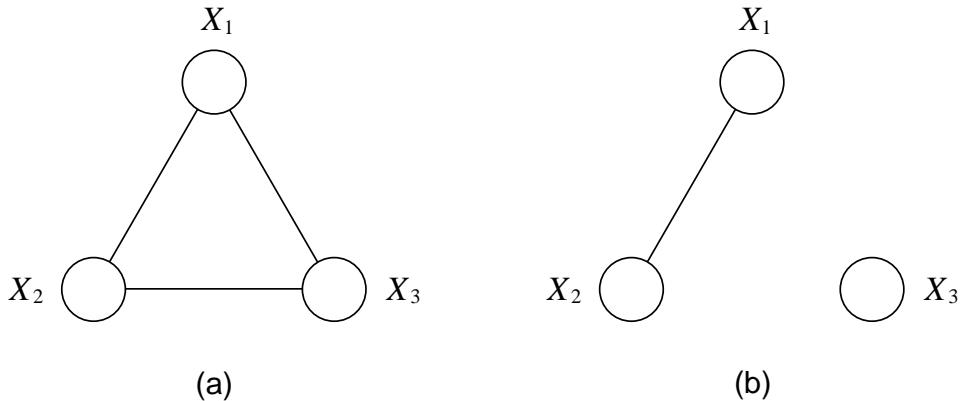


Figure 2.27: (a) An undirected graph which makes no independence assertions. (b) An undirected graph which asserts $X_3 \perp\!\!\!\perp \{X_1, X_2\}$.

factors on smaller subsets of variables. Thus, we might let:

$$\psi(x_1, x_2, x_3) = f(x_1, x_2)g(x_3), \quad (2.47)$$

or let:

$$\psi(x_1, x_2, x_3) = r(x_1, x_2)s(x_2, x_3)t(x_1, x_3), \quad (2.48)$$

for given functions f , g , r , s and t . Ranging over all possible choices of these functions, we obtain potentials that are necessarily members of the family associated with the graph in Figure 2.27(a)—because all such potentials that respect the (vacuous) conditional independence requirement. The potential in Eq. (2.47), however, also respects the (non-vacuous) conditional independence requirement of the graph in Figure 2.27(b). We would normally use this latter graph if we decide (a priori) to restrict our parameterization to the form given in Eq. (2.47). On the other hand, the potential given in Eq. (2.48) is problematic in this regard—there is no smaller graph that represents this class of potentials. Any graph with a missing edge makes an independence assertion regarding one or more pairs of variables, and $\psi(x_1, x_2, x_3) = r(x_1, x_2)s(x_2, x_3)t(x_1, x_3)$ does not respect such an assertion, when we range over all functions r , s and t .

Thus we see that “factorization” is a richer concept than “conditional independence.” There are families of probability distributions that can be defined in terms of certain factorizations of the joint probability that cannot be captured solely within the undirected or directed graphical model formalism. From the point of view of designing inference algorithms, this might not be viewed as a problem, because an algorithm that is correct for the graph is correct for a distribution in any sub-family defined on the graph. However, by ignoring the algebraic structure of the potential, we may be missing opportunities for simplifying the algebraic operations of inference.

In Chapter 4 we introduce *factor graphs*, a graphical representation of probability distributions in which such reduced parameterizations are made explicit. Factor graphs allow a more fine-grained representation of probability distributions than is provided by either the directed or the undirected graphical formalism, and in particular allow the factorization of the potential in Eq. (2.48) to be

represented explicitly in the graph. While factor graphs provide nothing new in terms of representing and exploiting conditional independence relationships—the main theme of the current chapter—they do provide a way to represent and exploit algebraic relationships, an issue that will return in Chapter 4.

2.4 Summary

In this chapter we have presented some of the basic definitions and basic issues that arise when one associates probability distributions with graphs. A key idea that we have emphasized is that a graphical model is a representation of a *family* of probability distributions. This family is characterized in one of two equivalent ways—either in terms of a numerical parameterization or in terms of a set of conditional independencies. Both of these characterizations are important and useful, and it is the interplay between these characterizations that gives the graphical models formalism much of its distinctive flavor.

Directed graphs and undirected graphs have different parameterizations and different conditional independence semantics, but the key concept of using graph theory to capture the notion of a joint probability distribution being constructed from a set of “local” pieces is the same in the two cases.

We have also introduced simple algorithms that help make the problem of understanding conditional independence in graphical models more concrete. The reader should be able to utilize the Bayes ball algorithm to read off conditional independence statements from directed graphs. Similarly, for undirected graphs the reader should understand that naive graph separation encodes conditional independence. Conditional independence assertions in undirected graphs can be assessed via a graph reachability algorithm.

2.5 Historical remarks and bibliography

Chapter 3

The Elimination Algorithm

In this chapter we discuss the problem of computing conditional and marginal probabilities in graphical models—the problem of *probabilistic inference*. Building on the ideas in Chapter ??, we show how the conditional independencies encoded in a graph can be exploited for efficient computation of conditional and marginal probabilities.

We take a very concrete approach in the current chapter, basing the presentation on a simple “elimination algorithm” for probabilistic inference. This algorithm applies equally well to directed and undirected graphs. It requires little formal machinery to describe and to analyze. On the other hand, the algorithm has its limitations, and is not our final word on the inference problem. But it is a good place to start.

3.1 Probabilistic inference

Let E and F be disjoint subsets of the node indices of a graphical model, such that X_E and X_F are disjoint subsets of the random variables in the domain. Our goal is to calculate $p(x_F | x_E)$ for arbitrary subsets E and F . This is the general *probabilistic inference problem* for graphical models (directed or undirected).

We begin by focusing on directed graphs. Almost all of our work, however, will transfer to undirected graphs with little or no change. Our subsequent treatment of undirected models in Section 3.1.3 will be short and sweet.

Throughout the chapter we limit ourselves to the probability of calculating the conditional probability of a single node X_F —which we refer to as the “query node”—given an arbitrary set of nodes X_E . This is a limitation of the simple elimination algorithm that we discuss in this chapter, and is not a limitation of the more general algorithms that we discuss in later chapters.

Graphically we indicate the set of conditioning variables by shading the corresponding nodes in the graph. Thus, the dark shading in Figure 3.1 indicates the nodes (indexed by E) on which we condition. We will often refer to these nodes as the *evidence nodes*. The unshaded nodes (indexed by F) are the nodes for which we wish to compute conditional probabilities. Finally, the lightly shaded nodes, indexed by $R = V \setminus (E \cup F)$, are the nodes that must be marginalized out of the joint probability so that we can focus on the conditional, $p(x_F | x_E)$, of interest. Thus, symbolically, we

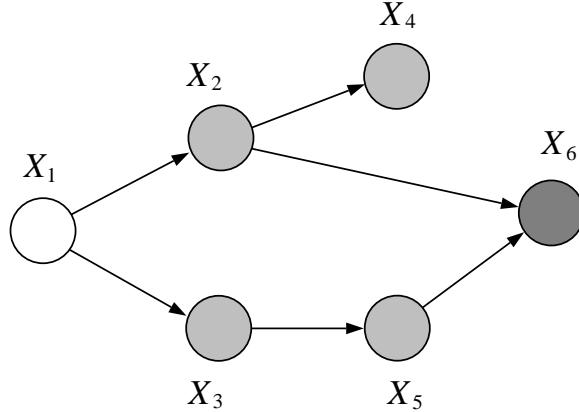


Figure 3.1: The dark shaded node, X_6 , is the node on which we condition, the lightly shaded nodes, $\{X_2, X_3, X_4, X_5\}$, are nodes that are marginalized over, and the unshaded node, X_1 is the node for which we wish to calculate conditional probabilities. Thus, for this example, we have $E = \{6\}$, $F = \{1\}$, and $R = \{2, 3, 4, 5\}$.

must compute the marginal:

$$p(x_E, x_F) = \sum_{x_R} p(x_E, x_F, x_R), \quad (3.1)$$

which can be further marginalized to yield $p(E)$:

$$p(x_E) = \sum_{x_F} p(x_E, x_F), \quad (3.2)$$

from which we obtain the conditional probability:

$$p(x_F | x_E) = \frac{p(x_E, x_F)}{p(x_E)}. \quad (3.3)$$

We will be interested in finding effective computational methods for making these calculations.

A special case of the general problem is worth noting. Consider the case of just two nodes, X and Y , as shown in Figure 3.2(a). This model is specified in terms of the distributions $p(x)$ and $p(y|x)$, reflecting the arrow from X to Y . Suppose that we condition on X , as shown in Figure 3.2(b), and wish to calculate the probability of Y . This “calculation” is simply a table lookup using $p(y|x)$. On the other hand, suppose that we condition on Y and wish to calculate the probability of X , as indicated in Figure 3.2(c). This is achieved via an application of Bayes rule:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}. \quad (3.4)$$

where the denominator is calculated as follows:

$$p(y) = \sum_x p(y|x)p(x). \quad (3.5)$$

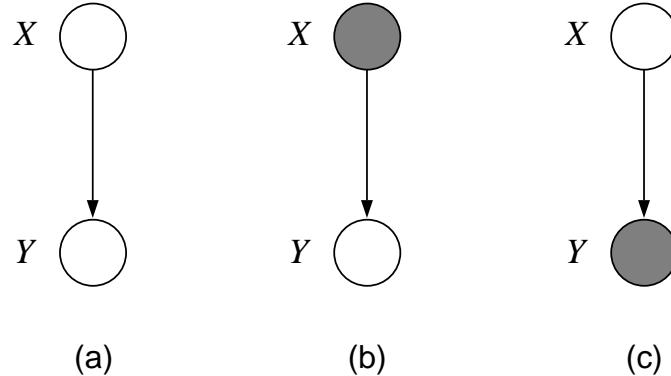


Figure 3.2: (a) A two-node model. (b) Conditioning on X involves a simple evaluation of $p(y|x)$. (c) Conditioning on Y requires the use of Bayes rule.

Can we find an efficient extension of these familiar ideas to general graphs?

The summations in Eq. (3.1) and Eq. (3.2) should give us pause. The summation \sum_{x_R} expands into a sequence of summations, one for each of the random variables indexed by R . If each such random variable can take on k values, and there are $|R|$ variables, we obtain $k^{|R|}$ terms in our summation. A similar statement applies to the summation \sum_{x_F} . With $|F|$ and $|R|$ in the dozens or hundreds in typical cases, naive summation is infeasible.

We need to take advantage of the factorization offered by the definition of the joint probability. If we do not take advantage of the factorization we will be in trouble performing even a single summation, much less a sequence of summations. Consider summing $p(x_1, x_2, \dots, x_6)$ with respect to x_6 , where we naively represent the joint probability as a table of size k^6 . (Recall that k is the number of values that each variable x_i can take on, assumed independent of i for simplicity.) Given that we must perform the sum for each value of the variables $\{x_1, x_2, \dots, x_5\}$, we see that we must perform $O(k^6)$ operations to do a single sum (essentially, we must touch each entry in the table). To reduce the computational complexity let us instead represent the joint probability in its factored form (cf. Eq. (2.3)) and exploit the distributive law:

$$p(x_1, x_2, \dots, x_5) = \sum_{x_6} p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2)p(x_5|x_3)p(x_6|x_2, x_5) \quad (3.6)$$

$$= p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2)p(x_5|x_3)\sum_{x_6} p(x_6|x_2, x_5). \quad (3.7)$$

The summation over x_6 is now applied to $p(x_6|x_2, x_5)$, a table of size k^3 . We have reduced the operation count from $O(k^6)$ to $O(k^3)$, a significant improvement.¹

Successive summations also take advantage of the factorization. A summation over, say, x_5 , can also be moved along the chain of factors until it encounters a factor involving x_5 . If each such summation is of reduced complexity, say $O(k^r)$ for some r , then the result is an algorithm that

¹Of course this sum is unity by the definition of conditional probability, and thus we don't actually have to perform any operations at all, but let us pretend not to know that.

scales as $O(nk^r)$ instead of $O(k^n)$. Of course, the summations create intermediate factors that may link variables, making it not entirely clear whether or not we can keep r small. It is here that graphical methods are helpful. We can determine the parameter r by a graph-theoretic algorithm.

Let us introduce the basic ideas in the context of an example. Referring to the graph in Figure 3.1, let us condition on the event $\{X_6 = x_6\}$ and calculate the conditional probability $p(x_1 | x_6)$.

A point to note at the outset is that x_6 is a fixed constant in this calculation and does not contribute to the computational complexity of the calculation. Thus, while the table representing $p(x_6 | x_2, x_5)$ is nominally a three-dimensional table, the observation of X_6 involves taking a two-dimensional slice of this table. Unfortunately our notation is ambiguous in this regard; we have been using “ x_6 ” as a variable that ranges over the possible values of X_6 . In particular it is meaningful to sum over “ x_6 .” In the remainder of this section, to avoid confusion, we refer to a particular fixed value of X_6 as “ \bar{x}_6 .” Thus, we wish to compute $p(x_1 | \bar{x}_6)$, for any x_1 and for a particular \bar{x}_6 .

We begin by computing the probability $p(x_1, \bar{x}_6)$ by summing over $\{x_2, x_3, x_4, x_5\}$. We introduce some notation to refer to intermediate factors that arise when performing these sums. In particular, let $m_i(x_{S_i})$ denote the expression that arises from performing the sum \sum_{x_i} , where x_{S_i} are the variables, other than x_i , that appear in the summand. Thus we have:

$$p(x_1, \bar{x}_6) = \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} p(x_1) p(x_2 | x_1) p(x_3 | x_1) p(x_4 | x_2) p(x_5 | x_3) p(\bar{x}_6 | x_2, x_5) \quad (3.8)$$

$$= p(x_1) \sum_{x_2} p(x_2 | x_1) \sum_{x_3} p(x_3 | x_1) \sum_{x_4} p(x_4 | x_2) \sum_{x_5} p(x_5 | x_3) p(\bar{x}_6 | x_2, x_5) \quad (3.9)$$

$$= p(x_1) \sum_{x_2} p(x_2 | x_1) \sum_{x_3} p(x_3 | x_1) \sum_{x_4} p(x_4 | x_2) m_5(x_2, x_3) \quad (3.10)$$

where we define $m_5(x_2, x_3) \triangleq \sum_{x_5} p(x_5 | x_3) p(\bar{x}_6 | x_2, x_5)$. (Note that to simplify notation we do not indicate explicitly the dependence of this term on the constant \bar{x}_6). Computing $m_5(x_2, x_3)$ has eliminated X_5 from further consideration in the computation. As we will see later, this algebraic notion of “elimination” corresponds to a graphical notion of elimination in which the node X_5 is removed from the graph. We continue the derivation:

$$p(x_1, \bar{x}_6) = p(x_1) \sum_{x_2} p(x_2 | x_1) \sum_{x_3} p(x_3 | x_1) m_5(x_2, x_3) \sum_{x_4} p(x_4 | x_2) \quad (3.11)$$

$$= p(x_1) \sum_{x_2} p(x_2 | x_1) m_4(x_2) \sum_{x_3} p(x_3 | x_1) m_5(x_2, x_3). \quad (3.12)$$

Of course, $m_4(x_2) \triangleq \sum_{x_4} p(x_4 | x_2)$ is equal to one by definition, and in practice we would not do this sum, but let us be systematic and keep the term in our calculations. Finally, we have:

$$p(x_1, \bar{x}_6) = p(x_1) \sum_{x_2} p(x_2 | x_1) m_4(x_2) m_3(x_1, x_2) \quad (3.13)$$

$$= p(x_1) m_2(x_1). \quad (3.14)$$

From this result we can also obtain the probability $p(\bar{x}_6)$ by taking an additional sum over x_1 :

$$p(\bar{x}_6) = \sum_{x_1} p(x_1)m_2(x_1), \quad (3.15)$$

and the desired conditional is obtained by dividing Eq. (3.14) into Eq. (3.15):

$$p(x_1 | \bar{x}_6) = \frac{p(x_1)m_2(x_1)}{\sum_{x_1} p(x_1)m_2(x_1)}. \quad (3.16)$$

Alternatively we can view $p(x_1, \bar{x}_6)$ in Eq. (3.14) as an unnormalized representation of the conditional probability $p(x_1 | \bar{x}_6)$ —recall once again that \bar{x}_6 is a fixed constant. Thus we obtain the conditional by normalization, where the normalization constant is given by Eq. (3.15).

Lying behind this flurry of algebra is a simple general algorithm for computing marginal probabilities. We present this algorithm in Section 3.1.2. First, however, we set the stage for the general algorithm with some preparatory remarks on conditioning.

3.1.1 Conditioning

To provide a simple exposition of the general elimination algorithm in Section 3.1.2, and also to simplify our exposition of inference algorithms presented in later chapters, it is useful to make use of a notational trick in which conditioning is viewed as a summation. This trick will allow us to treat marginalization and conditioning as formally equivalent, and will make it easier to bring the key operations of the inference algorithms into focus.

Let X_i be an evidence node whose observed value is \bar{x}_i . To capture the fact that X_i is fixed at the value \bar{x}_i , we define an *evidence potential*, $\delta(x_i, \bar{x}_i)$, a function whose value is one if $x_i = \bar{x}_i$ and zero otherwise. The evidence potential allows us to turn evaluations into sums: To evaluate a function $g(x_i)$ at \bar{x}_i we multiply $g(x_i)$ by $\delta(x_i, \bar{x}_i)$ and sum over x_i :

$$g(\bar{x}_i) = \sum_{x_i} g(x_i)\delta(x_i, \bar{x}_i), \quad (3.17)$$

a trick that also extends to multivariate functions with x_i as one of the arguments. In particular, returning to the example from the previous section, we can express the evaluation of $p(x_6 | x_2, x_5)$ at \bar{x}_6 as follows:

$$m_6(x_2, x_5) = \sum_{x_6} p(x_6 | x_2, x_5)\delta(x_6, \bar{x}_6), \quad (3.18)$$

where $m_6(x_2, x_5)$ is nothing but $p(\bar{x}_6 | x_2, x_5)$.

In general, let E be the set of nodes whose values are to be conditioned on. That is, for a specific configuration \bar{x}_E , we wish to compute $p(x_F | \bar{x}_E)$. Formally, we achieve this as follows. Define the *total evidence potential*:

$$\delta(x_E, \bar{x}_E) \triangleq \prod_{i \in E} \delta(x_i, \bar{x}_i), \quad (3.19)$$

a function that is equal to one if $x_E = \bar{x}_E$ and is equal to zero otherwise. Using this potential, we can obtain both the numerator and the denominator of the conditional probability $p(x_F | \bar{x}_E)$ by summation. Thus:

$$p(x_F, \bar{x}_E) = \sum_{x_E} p(x_F, x_E) \delta(x_E, \bar{x}_E) \quad (3.20)$$

and:

$$p(\bar{x}_E) = \sum_{x_F} \sum_{x_E} p(x_F, x_E) \delta(x_E, \bar{x}_E). \quad (3.21)$$

This suggests that it may be useful to define:

$$p^E(x) \triangleq p(x) \delta(x_E, \bar{x}_E) \quad (3.22)$$

as a generalized measure that represents conditional probability with respect to E . By formally “marginalizing” this measure with respect to x_E , we evaluate $p(x)$ at $X_E = \bar{x}_E$, and obtain $p(x_F, \bar{x}_E)$, an unnormalized version of the conditional probability $p(x_F | \bar{x}_E)$. Moreover, by marginalizing over x , we obtain the total “mass” $p(\bar{x}_E)$.

This tactic is particularly natural in the case of undirected graphs, where multiplication by an evidence potential $\delta(x_i, \bar{x}_i)$ can be implemented by simply redefining the local potentials $\psi(x_i)$ for $i \in E$. Thus, we define:

$$\psi_i^E(x_i) \triangleq \psi_i(x_i) \delta(x_i, \bar{x}_i), \quad (3.23)$$

for $i \in E$. Leaving all other clique potentials unchanged, that is, letting $\psi_C^E(x_C) \triangleq \psi_C(x_C)$, for $C \notin \{\{i\} : i \in E\}$, we obtain the desired unnormalized representation:

$$p^E(x) \triangleq \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_{X_C}^E(x_C). \quad (3.24)$$

Moreover, since we are working with an unnormalized representation, we may as well drop the $1/Z$ factor, and simply work with $\prod_{C \in \mathcal{C}} \psi_{X_C}^E(x_C)$ as an unnormalized representation of conditional probability.

It should be clear that the use of evidence potentials is merely a piece of formal trickery that will (turn out to) simplify our description of various inference algorithms. In practice we would not actually perform the sum over a function that we know to be zero over most of the sample space, but rather we would take “slices” of the appropriate probabilities or potentials. Thus, in evaluating $p(x_6 | x_2, x_5)$ at $X_6 = \bar{x}_6$, while formally we can view ourselves as multiplying by $\delta(x_6, \bar{x}_6)$ and summing over x_6 , algorithmically we would simply take the appropriate two-dimensional slice of the three-dimensional table representing $p(x_6 | x_2, x_5)$.

3.1.2 Elimination and directed graphs

In this section we describe a general algorithm for performing probabilistic inference in directed graphical models.

At each step of the algorithm, we perform a sum over a product of functions. The functions that can appear in such sums include the original local conditional probabilities, $p(x_i | x_{\pi_i})$, the

evidence potentials, $\delta(x_i, \bar{x}_i)$, and the intermediate factors, $m_i(x_{S_i})$, generated by previous sums. All of these functions are defined on local subsets of nodes, and we use the generic term “potential” to refer to all of them.² Thus our algorithm will involve taking sums over products of potential functions.

The algorithm works as follows (see Figure 3.3 for a summary). Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, an evidence set E , and a query node F , we first choose an elimination ordering I such that F appears last in the ordering.³ Throughout the algorithm we maintain an *active list* of potential functions. The active list is initialized to hold the local conditional probabilities, $p(x_i | x_{\pi_i})$, for $i \in \mathcal{V}$, and the evidence potentials, $\delta(x_i, \bar{x}_i)$, for $i \in E$. At each step of the algorithm, we find all those potentials on the active list that reference the next node (call it X_i) in the elimination ordering I . These potential functions are removed from the active list. We take the product of these functions and sum this product with respect to x_i . This defines a new intermediate factor, $m_i(x_{S_i})$, where x_{S_i} are the variables (other than x_i) that appear in the summand. This intermediate factor is added to the active list. We then proceed to the next node in the elimination ordering.

Note that we have introduced the notation $T_i = \{i\} \cup S_i$ in the description of the algorithm in Figure 3.3. The subset T_i indexes the set of all variables that appear in the summand of the operator \sum_{x_i} . We give a graph-theoretic interpretation of T_i later in the chapter.

The algorithm terminates when we arrive at the final node in the elimination ordering, the query node X_F . The product of potentials on the active list at this point defines the (unnormalized) conditional probability, $p(x_F, \bar{x}_E)$. Summing this product over x_F yields the normalization factor $p(\bar{x}_E)$.

Let us now return to the example in Section 12.4 and show how the steps of ELIMINATE correspond to the steps in the algebraic calculation in that section. The evidence node in this example is X_6 and the query node is X_1 . We choose the elimination ordering $I = (6, 5, 4, 3, 2, 1)$, in which the query node appears last.

We begin by placing the local conditional probabilities, $\{p(x_1), \dots, p(x_6 | x_2, x_5)\}$, on the active list. We also place $\delta(x_6, \bar{x}_6)$ on the active list.

We first eliminate node X_6 . The potential functions on the active list that reference x_6 are $p(x_6 | x_2, x_5)$ and $\delta(x_6, \bar{x}_6)$. Thus we have $\phi_6(x_2, x_5, x_6) = p(x_6 | x_2, x_5)\delta(x_6, \bar{x}_6)$. Summing this expression with respect to x_6 yields $m_6(x_2, x_5) = p(\bar{x}_6 | x_2, x_5)$. We place this potential on the active list, having removed $p(x_6 | x_2, x_5)$ and $\delta(x_6, \bar{x}_6)$. We have simply evaluated $p(x_6 | x_2, x_5)$ at \bar{x}_6 .

We now eliminate X_5 . The potentials on the active list that reference x_5 are $p(x_5 | x_3)$

²The reader may be concerned that we are using the term “potential” somewhat loosely here. In particular we are using it in the context of directed graphs and in the context of subsets that may not be cliques; this usage clashes with the definition of “potential” in Chapter ???. We hope that the reader will forgive the seeming abuse of terminology. It is worth noting, however, that the “potentials” discussed in this section are in fact honest-to-goodness potentials, but not with respect to G . Rather they are potentials on the cliques of a different graph, a graph known as the *moral graph* G^m . This point will be clarified in Section ?? below.

³We will not discuss the choice of elimination ordering in this chapter, but instead will defer this (non-trivial) problem until Chapter 17, where it will arise in a more general way in the context of the junction tree algorithm. For now, let the ordering I be arbitrary, under the constraint that F appears last. We might encourage the reader, however, to start to ponder how to characterize good elimination orderings. Some useful food for thought in this regard will be provided in Section ?? below.

```

ELIMINATE( $\mathcal{G}, E, F$ )
  INITIALIZE( $\mathcal{G}, F$ )
  EVIDENCE( $E$ )
  UPDATE( $\mathcal{G}$ )
  NORMALIZE( $F$ )

INITIALIZE( $\mathcal{G}, F$ )
  choose an ordering  $I$  such that  $F$  appears last
  for each node  $X_i$  in  $\mathcal{V}$ 
    place  $p(x_i | x_{\pi_i})$  on the active list
  end

EVIDENCE( $E$ )
  for each  $i$  in  $E$ 
    place  $\delta(x_i, \bar{x}_i)$  on the active list
  end

UPDATE( $\mathcal{G}$ )
  for each  $i$  in  $I$ 
    find all potentials from the active list that reference  $x_i$  and remove them from the active list
    let  $\phi_i(x_{T_i})$  denote the product of these potentials
    let  $m_i(x_{S_i}) = \sum_{x_i} \phi_i(x_{T_i})$ 
    place  $m_i(x_{S_i})$  on the active list
  end

NORMALIZE( $F$ )
   $p(x_F | \bar{x}_E) \leftarrow \phi_F(x_F) / \sum_{x_F} \phi_F(x_F)$ 

```

Figure 3.3: The ELIMINATE algorithm for probabilistic inference on directed graphs.

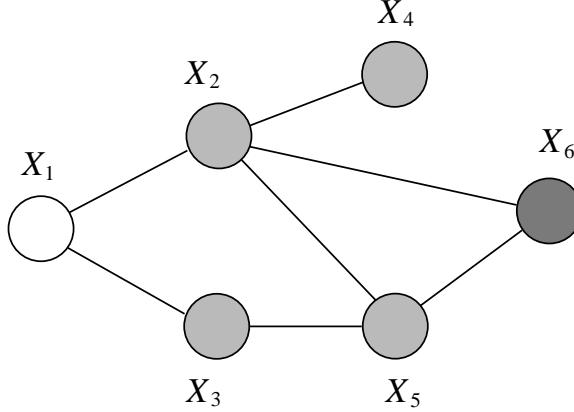


Figure 3.4: The dark shaded node, X_6 , is the node on which we condition, the lightly shaded nodes, $\{X_2, X_3, X_4, X_5\}$, are the nodes that are marginalized over, and the unshaded node, X_1 , is the node for which we wish to calculate conditional probabilities.

and $m_6(x_2, x_5)$. We remove them, and define the product $\phi_5(x_2, x_3, x_5)$. Summing over x_5 yields $m_5(x_2, x_3)$ (cf. Eq. (3.11)).

The only potential that references X_4 is $p(x_4 | x_2)$. The elimination of X_4 thus involves summing $p(x_4 | x_2)$ with respect to x_4 to obtain the factor $m_4(x_2)$. This factor is identically one and in practice we would not bother computing it.

Eliminating X_3 involves taking the sum over $\phi_3(x_1, x_2, x_3) = p(x_3 | x_1)m_5(x_2, x_3)$ to yield $m_3(x_1, x_2)$ and we are now at Eq. (3.13) in the earlier derivation.

We now eliminate X_2 to obtain $\phi_1(x_1) = p(x_1)m_2(x_1)$, which is the “unnormalized conditional probability,” $p(x_1, \bar{x}_6)$. Eliminating X_1 yields $m_1 = \sum_{x_1} \phi_1(x_1)$, which is the normalization factor, $p(\bar{x}_6)$.

3.1.3 Elimination and undirected graphs

In the case of directed models, we have shown that the problem of calculating conditional probabilities can be usefully viewed in terms of a simple elimination algorithm. The same perspective applies to undirected models, and indeed the entire ELIMINATE algorithm from Figure 3.3 goes through without essential change to the undirected case.

The only change needed to handle the undirected case occurs in the INITIALIZE procedure, where instead of using local conditional probabilities we initialize the active list to contain the potentials $\{\psi_{X_C}(x_C)\}$.

Let us briefly consider an example. Paralleling the example from Section 3.1.2 we calculate the probability $p(x_1 | \bar{x}_6)$ for the graph in Figure 3.4. We represent the joint probability on the graph via potential functions on the cliques $\{X_1, X_2\}$, $\{X_1, X_3\}$, $\{X_2, X_4\}$, $\{X_3, X_5\}$, and $\{X_2, X_5, X_6\}$.

We first calculate the probability $p(x_1, \bar{x}_6)$. To simplify the presentation we drop the subscript in the $\psi_{X_C}(x_C)$ notation, relying on the argument to the function to make it clear which potential function is being referred to. Also we again make use of the notation $m_i(x_{S_i})$ to denote the

intermediate factor that results from the summation over x_i . We have:

$$\begin{aligned}
p(x_1, \bar{x}_6) &= \frac{1}{Z} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} \sum_{x_6} \psi(x_1, x_2) \psi(x_1, x_3) \psi(x_2, x_4) \psi(x_3, x_5) \psi(x_2, x_5, x_6) \delta(x_6, \bar{x}_6) \\
&= \frac{1}{Z} \sum_{x_2} \psi(x_1, x_2) \sum_{x_3} \psi(x_1, x_3) \sum_{x_4} \psi(x_2, x_4) \sum_{x_5} \psi(x_3, x_5) \sum_{x_6} \psi(x_2, x_5, x_6) \delta(x_6, \bar{x}_6) \\
&= \frac{1}{Z} \sum_{x_2} \psi(x_1, x_2) \sum_{x_3} \psi(x_1, x_3) \sum_{x_4} \psi(x_2, x_4) \sum_{x_5} \psi(x_3, x_5) m_6(x_2, x_5) \\
&= \frac{1}{Z} \sum_{x_2} \psi(x_1, x_2) \sum_{x_3} \psi(x_1, x_3) m_5(x_2, x_3) \sum_{x_4} \psi(x_2, x_4) \\
&= \frac{1}{Z} \sum_{x_2} \psi(x_1, x_2) m_4(x_2) \sum_{x_3} \psi(x_1, x_3) m_5(x_2, x_3) \\
&= \frac{1}{Z} \sum_{x_2} \psi(x_1, x_2) m_4(x_2) m_3(x_1, x_2) \\
&= \frac{1}{Z} m_2(x_1).
\end{aligned} \tag{3.25}$$

Marginalizing further over x_1 yields:

$$p(\bar{x}_6) = \frac{1}{Z} \sum_{x_1} m_2(x_1), \tag{3.26}$$

and we calculate the desired conditional as:

$$p(x_1 | \bar{x}_6) = \frac{m_2(x_1)}{\sum_{x_1} m_2(x_1)}, \tag{3.27}$$

where the normalization factor Z cancels.

Note that the calculation in the example is formally identical to the corresponding calculation for directed graphs. Note, however, that the sum $m_4(x_2)$, which earlier could be omitted, no longer necessarily sums to one and must be explicitly carried along in the calculation.

Finally, a remark on the computation of marginal probabilities $p(x_i)$. For a marginal probability the normalization factor Z does not cancel, and must be calculated explicitly. Just as in the other calculations in this section, however, the calculation of Z is a summation over the unnormalized representation of the joint probability, and indeed it is simply a summation over *all* of the variables. To obtain the marginal $p(x_i)$, we would define an elimination ordering in which x_i is the final variable, and then normalize the result to calculate Z and obtain the marginal.

In the directed case, a variable that is parentless has its marginal represented explicitly in the parameterization of the graphical model and no calculation is needed. In general, nodes that are downstream from a target node can simply be deleted, and marginalization involves an inference calculation involving the ancestors of the node. The worst case is a leaf node. In the undirected case, there is no notion of “ancestor,” and essentially all nodes are worst case. On the other hand, once Z is calculated from a particular elimination ordering, it can be used to normalize other marginal probabilities.

```

UNDIRECTEDGRAPHELIMINATE( $\mathcal{G}, I$ )
  for each node  $X_i$  in  $I$ 
    connect all of the remaining neighbors of  $X_i$ 
    remove  $X_i$  from the graph
  end

```

Figure 3.5: A simple greedy algorithm for eliminating nodes in an undirected graph \mathcal{G} .

3.2 Graph elimination

The simple ELIMINATE algorithm captures the key algorithmic operation underlying probabilistic inference—that of taking a sum over a product of potential functions. What can we say about the overall computational complexity of the algorithm? In particular, how can we control the “size” of the summands that appear in the sequence of summation operations?

In this section, we show that questions regarding the computational complexity of the ELIMINATE algorithm can be reduced to purely graph-theoretic considerations. This graphical interpretation will also provide hints about how to design improved inference algorithms that overcome the limitations of ELIMINATE.

3.2.1 A graph elimination algorithm

Let us put aside marginalization and probabilistic inference for a moment, and concentrate solely on graph-theoretic manipulations. We describe a simple procedure that eliminates nodes in a graph. As will become clear, this procedure captures the graph-theoretic operations underlying ELIMINATE.

We begin by describing a node-elimination algorithm for undirected graphs, making the link to directed graphs shortly.

Assume that we are given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and an elimination ordering I . We describe a simple graph-theoretic algorithm that successively eliminates the nodes of \mathcal{G} . In particular, at each step, the algorithm eliminates the next node in the ordering I , where “eliminate” means removing the node from the graph and connecting the (remaining) neighbors of the node. The algorithm, which we refer to as UNDIRECTEDGRAPHELIMINATE, is presented in Figure 3.5.

We will be interested in the *reconstituted graph*; the graph $\tilde{\mathcal{G}} = (\mathcal{V}, \tilde{\mathcal{E}})$, whose edge set $\tilde{\mathcal{E}}$ is a superset of \mathcal{E} , incorporating all of the original edges \mathcal{E} , as well as any new edges created during a run of UNDIRECTEDGRAPHELIMINATE.

Consider in particular the graph in Figure 3.6(a) and the elimination ordering $(6, 5, 4, 3, 2, 1)$. Let us run through the graphical elimination procedure. Starting with node X_6 we first connect its neighbors, adding an edge between X_2 and X_5 , as shown in Figure 3.6(b). We then remove X_6 , which yields Figure 3.6(c). Moving to X_5 , we connect its neighbors, X_2 and X_3 , and remove X_5 , which yields Figure 3.6(d). The procedure continues in Figure 3.6(e)–(g), culminating in a graph with the single node X_1 , which is then removed yielding the empty graph.

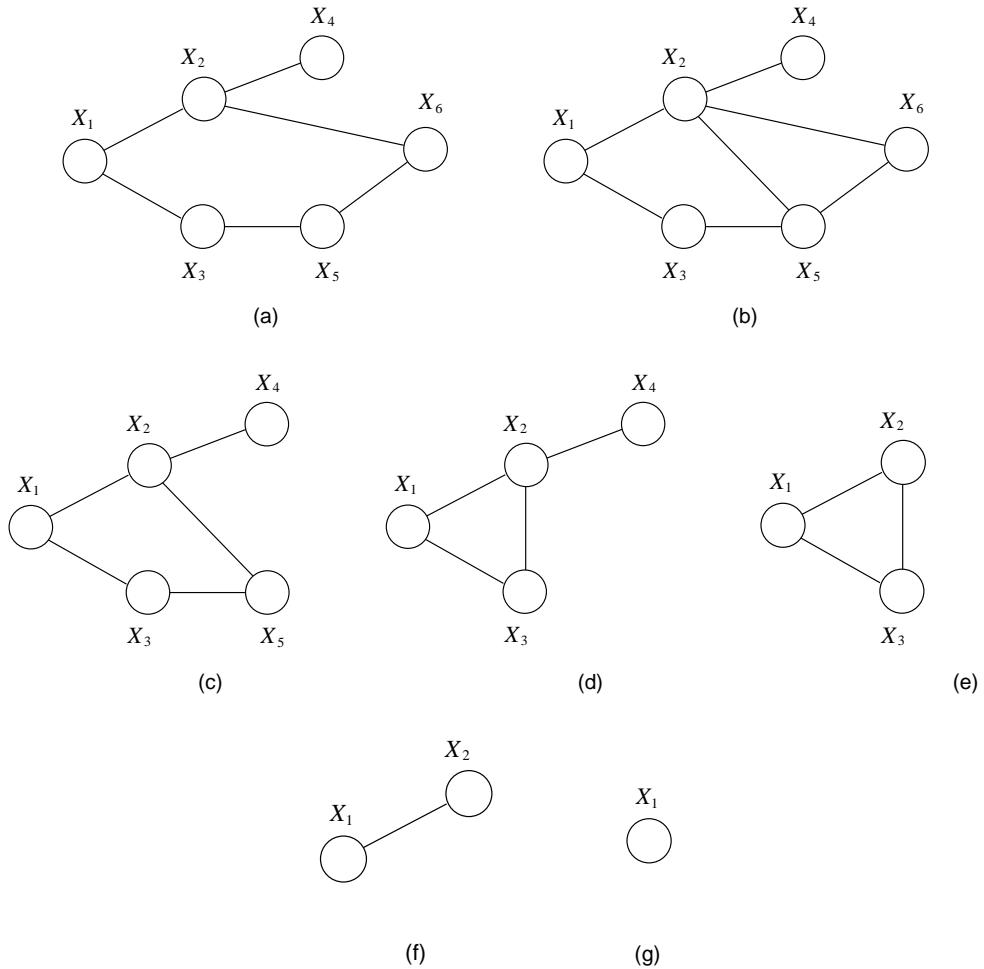


Figure 3.6: A run of the elimination algorithm under the elimination ordering $(6, 5, 4, 3, 2, 1)$. The original graph is shown in (a).

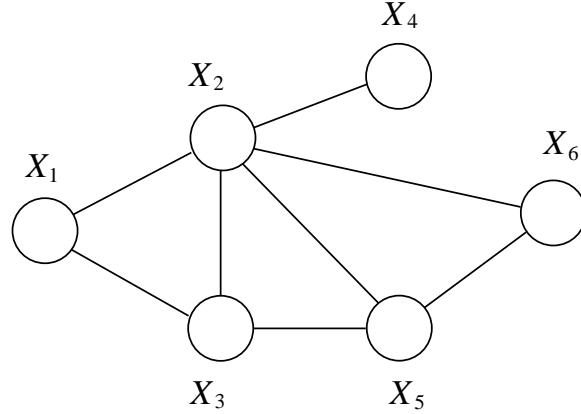


Figure 3.7: The reconstituted graph, showing the edges that were added during the elimination process.

Figure 3.7 shows the reconstituted graph, where we have retained the edges that were created during the elimination procedure (in particular, the edges between X_2 and X_3 and between X_2 and X_5). This graph turns out to have some important graph-theoretic properties, properties which underly the comprehensive theory of inference that will be the subject of Chapter 17.⁴ For current purposes, however, the relevant properties of the graph can be captured by recording the *elimination cliques* of the graph. In particular, each time we remove a node X_i in the second step of the algorithm, let us record the collection of nodes that are the neighbors of X_i at that moment, including X_i itself. These nodes form a fully-connected subset of nodes by virtue of the first step of the algorithm; that is, they form a clique. We denote the set of indices of the nodes in this clique as T_i . Thus, in our example, $T_6 = \{2, 5, 6\}$ and $T_5 = \{2, 3, 5\}$. (Note that index 6 does not appear in T_5 because X_6 has already been eliminated when we process node X_5).

3.2.2 Graph elimination and marginalization

When we perform a marginalization operation, removing a random variable from a joint distribution, we perform a sum over the product of all factors that depend on that random variable. This couples together all of the other random variables that appear in those factors. Thus, for example, summing the product $\psi(x_3, x_5)m_6(x_2, x_5)$ with respect to x_5 creates an intermediate factor that involves x_2 and x_3 . This new factor does not in general factorize with respect to x_2 and x_3 ; thus, we have an *induced dependency* between x_2 and x_3 . Subsequent operations will have to treat x_2 and x_3 together. `UNDIRECTEDGRAPHELIMINATE` makes this coupling explicit, by linking the neighbors of the node being summed over.

In general, as we now show, the elimination cliques in `UNDIRECTEDGRAPHELIMINATE` are the graph-theoretic counterparts of the sets of variables on which summations operate in probabilistic

⁴For readers who cannot bear the wait, the key property of the reconstituted graph is that it is a *triangulated graph*; indeed, our elimination procedure is a simple algorithm for triangulating a graph.

inference using `ELIMINATE`.

Consider the argument x_{T_i} to the function $\phi_i(x_{T_i})$ in `ELIMINATE`, the function which is the summand for the operator \sum_{x_i} . As our notation suggests, the variables referenced by ϕ_i are exactly those in the elimination clique created by `UNDIRECTEDGRAPHELIMINATE` upon elimination of X_i . To see this, note that any potential removed from the active list by `ELIMINATE` (when summing over x_i) must reference x_i . Now consider any other variable x_j referenced by $\phi_i(x_{T_i})$. We want to show that X_i and X_j must be neighbors in the graph constructed by `UNDIRECTEDGRAPHELIMINATE`. There are two cases to consider, corresponding to the two kinds of potentials that can link variables: (1) If the potential is one of the original potentials $\psi_C(x_C)$, then X_j is necessarily linked to X_i , because C is a clique (by definition). (2) If x_i and x_j appear together in an intermediate factor $m_k(x_{S_k})$, then this term was created by the elimination of an earlier node X_k . At the moment of eliminating X_k , `UNDIRECTEDGRAPHELIMINATE` must have linked the nodes X_i and X_j . Thus, in either case, X_j is a neighbor of X_i , and these nodes must appear together in the elimination clique X_{T_i} .

3.2.3 Computational complexity

Let us now consider the computational complexity of `ELIMINATE`. At each step we must sum over a variable x_i for all configurations of the variables in the summand $\phi_i(x_{T_i})$. Assuming that there is no special algebraic structure in this summand that can be exploited, the time and space complexities are exponential in the number of variables in the subset T_i . That is, the overall complexity of the algorithm is determined by the number of variables in the largest elimination clique. Thus, the question of the computational complexity of `ELIMINATE` can be reduced to the purely graph-theoretic question of the size of the largest elimination clique created by `UNDIRECTEDGRAPHELIMINATE`.

The problem of obtaining a largest elimination clique that is as small as possible, under all possible elimination orderings, is a well-studied problem in graph theory. The problem is generally expressed in terms of a parameter k known as the *treewidth*, which is one less than the smallest achievable value of the cardinality of the largest elimination clique, where we range over all possible elimination orderings.

Consider for example, the star graph on n nodes shown in Figure 3.8(a). If we were to eliminate the central node first, we would immediately link all other nodes, creating an elimination clique of size n . On the other hand, if we eliminate all of the leaf nodes first we never create a clique of cardinality greater than two. Indeed, the treewidth of this graph is equal to one.

Figure 3.8(b) shows a second example, in which it can be verified that it is possible to eliminate the nodes in such a way that the largest clique created is of size three. The treewidth is thus equal to two.

The general problem of finding the best elimination ordering of a graph—an elimination ordering that achieves the treewidth—turns out to be NP-hard. We discuss this hardness result, and its consequences for probabilistic inference, in more detail in Chapter 17. Indeed, in that chapter we discuss an inference algorithm (the junction tree algorithm) that generalizes `ELIMINATE` and necessitates a deeper discussion of the treewidth problem and methods for tackling it. As we will show, there are a number of useful heuristics for finding good elimination orders, and these can provide viable solutions in practical problems.

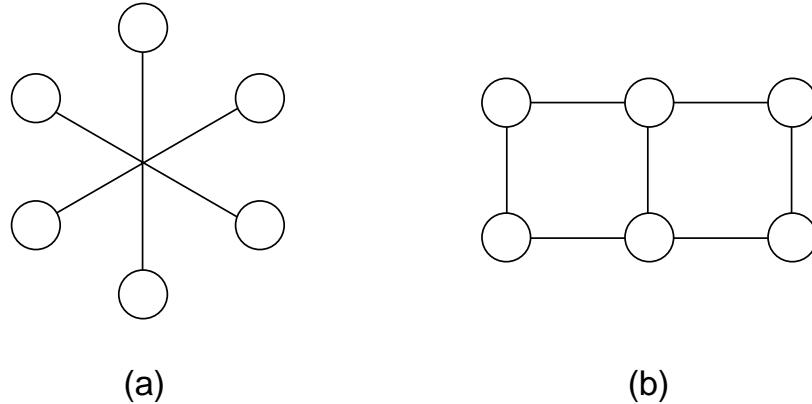


Figure 3.8: (a) A graph whose treewidth is equal to one. (b) A graph whose treewidth is equal to two.

In the meantime, all of the graphs that we study in intervening chapters will turn out to involve graphs that have “obvious” optimal elimination orderings.

The NP-hardness result should be taken as injecting a cautionary note into our study of elimination methods, suggesting that we should not expect `ELIMINATE` to provide a computationally-efficient solution to the general problem of probabilistic inference. On the other hand, we clearly should never have expected any such general solution from `ELIMINATE`. The fully-connected graph, for example, yields a single clique containing all of the nodes, under all possible elimination orderings, and thus has no graph-theoretic structure that `ELIMINATE` can exploit. To have any hope for efficient probabilistic inference in such a graph, we need to hope that other structural features of probability theory can be brought to bear.⁵

We can also take the NP-hardness result as providing a crisp statement of the computational bottleneck that arises in `ELIMINATE`. Indeed, note that `UNDIRECTEDGRAPHELIMINATE` provides a practically useful tool for assessing the severity of this bottleneck. For a given elimination ordering, we can obtain a cheap assessment of the predicted running time of `ELIMINATE` by running `UNDIRECTEDGRAPHELIMINATE`. If `UNDIRECTEDGRAPHELIMINATE` yields elimination cliques of reasonably small cardinality, then we know that it is viable to run `ELIMINATE`.

3.2.4 Graph elimination and directed graphs

All of the considerations of the previous three sections also apply to directed graphs. There is, however, a minor idiosyncracy of directed graphical models that must be addressed if we are to use the concept of “elimination clique” to analyze the directed version of `ELIMINATE`.

The functions that are used to initialize the active list in the directed case are conditional probabilities, $p(x_i | x_{\pi_i})$. Note that a pair of variables X_j and X_k that are parents of X_i are linked

⁵That is, there may be special algebraic structure in the potentials, or symmetries, or simplifications brought about by laws of large numbers. These issues will return in our consideration of approximate inference algorithms, in Chapter 20 and Chapter 21.

```

DIRECTEDGRAPHELIMINATE( $G, I$ )
 $G^m = \text{MORALIZE}(G)$ 
UNDIRECTEDGRAPHELIMINATE( $G^m, I$ )

MORALIZE( $G$ )
for each node  $X_i$  in  $I$ 
    connect all of the parents of  $X_i$ 
end drop the orientation of all edges
return  $G$ 

```

Figure 3.9: An algorithm for eliminating nodes in an directed graph G .

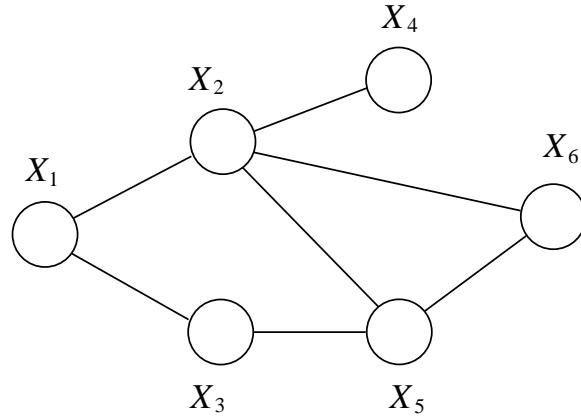


Figure 3.10: The moral graph corresponding to the directed graph in Figure 2.1.

functionally by their presence in the function $p(x_i | x_{\pi_i})$, but they are not necessarily neighbors in the graph \mathcal{G} (e.g., X_2 and X_5 are not linked in Figure 3.1). This breaks the relationship between elimination cliques and sets of arguments that we established in the previous section for undirected graphs.

There is a simple fix. To define the elimination cliques for a directed graph, first connect all of the parents of each node—this captures the basic fact that factors involving all variables X_{π_i} will necessarily appear in our calculations. Then drop the orientation of all of the edges in the graph, converting the graph to an undirected graph. This procedure, of “marrying” the parents of the nodes in a directed graph and converting to an undirected graph, is referred to as *moralization*. The resulting graph is referred to as a *moral graph*. We use moralization as a subroutine in the algorithm, DIRECTEDGRAPHELIMINATE, for eliminating directed graphs (see Figure 3.9).

The graph in Figure 3.10 is the moral graph corresponding to the directed graph in Figure 2.1. Note that (in the elimination order that we have been using in our example) X_6 is eliminated before

its parents X_2 and X_5 , and the elimination step already adds a link between these two nodes. That is, in this case, we do not need to moralize; elimination does it for us. On the other hand, if a parent of X_6 appears before X_6 in the elimination order, we need to moralize explicitly. Consider in particular an elimination ordering in which X_5 is eliminated first. The other parent, X_2 , is not a neighbor of X_5 when the latter node is eliminated, and thus is not included within the elimination clique of X_5 . This fails to capture the fact that summing over X_5 creates an intermediate factor that refers to X_2 and the other neighbors of X_5 . In general we need to moralize in the directed graphical setting if we want the elimination cliques to capture all such dependencies.

The considerations in this section may help to explain the important role that undirected graphical models play in designing and analyzing inference algorithms, a role that we will see again in later chapters, even when the original graphical model is directed. In a directed model, the basic factors that appear in the joint probability are *conditional* probabilities. There is of course a great difference between the appearance of a variable on the left-hand or right-hand side of a conditional probability. From the point of view of ELIMINATE, however, this difference is irrelevant. When we sum over x_k , the factor $p(x_i | x_j, x_k)$ and the factor $p(x_j | x_i, x_k)$ both create an intermediate factor linking x_i and x_j . Thus, to understand the computational complexity of inference, we need to ignore the directionality associated with a conditional probability. The undirected graphical formalism, which treats such a probability as a general potential, $\psi(x_i, x_j, x_k)$, does this automatically.

3.3 Discussion

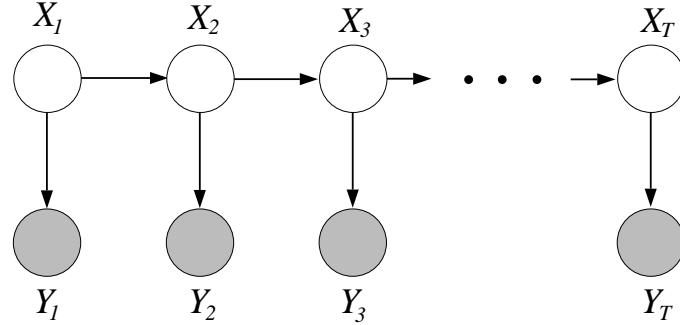
Our presentation of the elimination algorithm for probabilistic inference raises a number of questions:

- Can we prove that it works?
- What are its limitations?
- Can it be generalized?

Detailed answers to these questions will emerge in later chapters, but let us try to provide some short answers here.

It is not difficult to prove that the algorithm that we have presented is correct. Indeed, we ask the reader to provide a proof by induction in Exercise ??, and we present a proof by induction of the correctness of a closely-related algorithm (the SUM-PRODUCT algorithm) in Chapter 4. Moreover, in Chapter 17 we prove the correctness of the general junction tree algorithm, an algorithm that generalizes both ELIMINATE and SUM-PRODUCT.

The ELIMINATE algorithm has a number of limitations, some which are easily corrected and others which are not. In particular, taking ELIMINATE seriously as an algorithm to be implemented on a computer reveals a number of inefficiencies. Most importantly, the use of a single “active list” as a data structure requires an inefficient traversal of the entire list every time the algorithm eliminates a node. This can be fixed by maintaining a separate list, or “bucket,” for each node. Whenever the algorithm creates a new intermediate factor, $m_i(x_{S_i})$, it scans the elimination ordering I , and



combining intermediate factors. Rather than focusing on elimination orderings, the junction tree algorithm focuses on the relationships between intermediate factors, or “messages.” The same idea is present in simpler form in the SUM-PRODUCT algorithm, to which we now turn.

3.4 Historical remarks and bibliography

Chapter 4

Probability Propagation and Factor Graphs

In this chapter we describe an algorithm for probabilistic inference known as the *sum-product*, or *belief propagation*, algorithm. The algorithm is closely related to the elimination algorithm, and indeed we will derive it from the perspective of elimination. The algorithm goes significantly beyond the elimination algorithm, however, in that it can compute all single-node marginals (for certain classes of graphs) rather than only a single marginal.

It is important to be clear that we are also taking a step backward in this chapter—while the elimination algorithm is applicable to arbitrary graphs, the sum-product algorithm is designed to work only in trees (or in the various “tree-like” graphs that we discuss in this chapter). Despite this step backward, there are at least three reasons why the sum-product algorithm overall represents significant progress: (1) Trees are important graphs. Indeed, a significant fraction of the classical literature on graphical models was entirely restricted to trees, and many of these classical applications require the ability to compute all singleton marginals. Examples include the hidden Markov model of Chapter 12 and the state-space model of Chapter 15. (2) The sum-product algorithm provides new insights into the inference problem, insights which will eventually allow us to provide a general solution to the exact inference problem (the *junction tree algorithm* of Chapter 17). The sum-product algorithm essentially involves an efficient “calculus of intermediate factors,” which recognizes that many of the same intermediate factors are used in different elimination orderings. The junction tree algorithm extends this calculus to general graphs, by essentially combining the key ideas of the sum-product algorithm and the elimination algorithm. (3) While our focus in the current chapter is exact inference, the sum-product algorithm also provides the basis of a class of *approximate* inference algorithms for general graphs, as we discuss in Chapter 20.

Another goal of the current chapter is to introduce *factor graphs*, an alternative graphical representation of probabilities that is of particular value in the context of the sum-product algorithm. In particular, we will show that the factor graph approach provides an elegant way to handle various general “tree-like” graphs, including “polytrees,” a class of directed graphical models in which nodes have multiple parents.

Finally, we also broaden our agenda in the current chapter, moving beyond the problem of com-

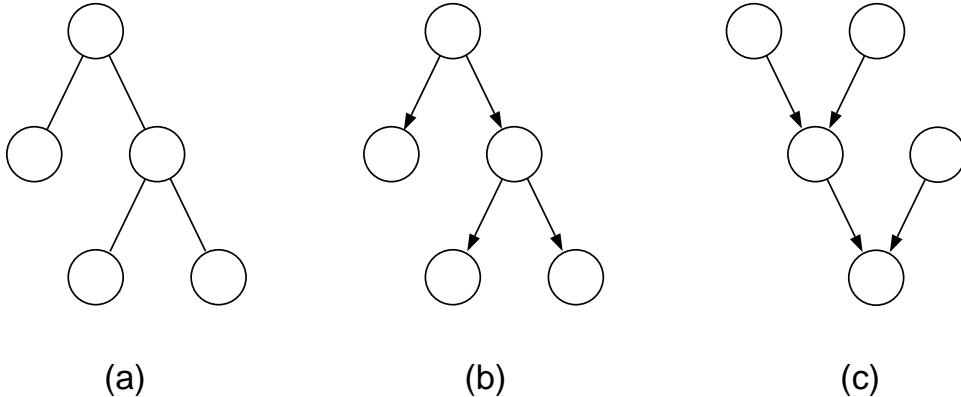


Figure 4.1: (a) An undirected tree. (b) A directed tree. (c) A polytree.

puting marginal and conditional probabilities to the problem of computing *maximum a posteriori probabilities*. We show that this problem can be solved via an algorithm that is closely related to the sum-product algorithm.

The chapter is organized as follows. In Section 4.1 we begin with a discussion of probabilistic inference on trees, treating both the directed case and the undirected case. Section ?? introduces *factor graphs*, discusses the relationships with directed and undirected graphs, and develops the sum-product algorithm for factor graphs. We discuss polytrees in Section 4.2.4, and discuss algorithms for computing maximum a posteriori probabilities in Section 4.3.

4.1 Probabilistic inference on trees

In this section we describe an inference algorithm for trees. Let us first clarify exactly what is meant by a “tree.” In the undirected case, a tree is an undirected graph in which there is one and only one path between any pair of nodes. An example of an undirected tree is shown in Figure 4.1(a).¹ In the directed case, we define a tree to be any graph whose moralized graph is an undirected tree. Figure 4.1(b) shows a directed tree. Note that directed trees have a single node that has no parent—the *root node*—and that all other nodes have exactly one parent. Finally, note that the graph in Figure 4.1(c) is not a directed tree; it has nodes with multiple parents, and the resulting moralized graph has loops.

Any undirected tree can be converted into a directed tree by choosing a root node and orienting all edges to point away from the root.

From the point of view of graphical model representation and inference there is little significant difference between directed trees and undirected trees. A directed tree and the corresponding undirected tree (the tree obtained by dropping the directionality of the edges) make exactly the same set of conditional independence assertions. Moreover, as we show below, the parameterizations

¹Note that throughout the chapter we assume implicitly that our graphs are connected, and thus we have a single tree rather than a forest. This is done without loss of generality—in the case of a forest we have a collection of probabilistically independent trees, and it suffices to run an inference algorithm separately on each tree.

are essentially the same, with the undirected parameterization being slightly more flexible by not requiring potentials to be normalized (but, see Exercise ??, any undirected representation can be readily converted to a directed one).

4.1.1 Parameterization and conditioning

Let us first consider the parameterization of probability distributions on undirected trees. The cliques are single nodes and pairs of nodes, and thus the joint probability can be parameterized via potential functions $\{\psi(x_i)\}$ and $\{\psi(x_i, x_j)\}$. In particular, we have:

$$p(x) = \frac{1}{Z} \left(\prod_{i \in \mathcal{V}} \psi(x_i) \prod_{(i,j) \in \mathcal{E}} \psi(x_i, x_j) \right), \quad (4.1)$$

for a tree $\mathcal{T}(\mathcal{V}, \mathcal{E})$ with nodes \mathcal{V} and edges \mathcal{E} .

For directed trees, the joint probability is formed by taking a product over a marginal probability, $p(x_r)$, at the root node r , and conditional probabilities, $\{p(x_j | x_i)\}$, at all other nodes:

$$p(x) = p(x_r) \prod_{(i,j) \in \mathcal{E}} p(x_j | x_i), \quad (4.2)$$

where (i, j) is a directed edge such that i is the (unique) parent of j (i.e., $\{i\} = \pi_j$). We can treat such a parameterization as a special case of Eq. (4.1), and indeed it will be convenient to do so throughout this chapter. We define:

$$\psi(x_r) = p(x_r) \quad (4.3)$$

$$\psi(x_i, x_j) = p(x_j | x_i), \quad (4.4)$$

for i the parent of j , and define all other singleton potentials, $\psi(x_i)$, for $i \neq r$, to be equal to one. We thereby express the joint probability for a directed tree in the undirected form in Eq. (4.1), with $Z = 1$.

Recall that we use “evidence potentials” to capture conditioning. Thus, if we are interested in the conditional probability $p(x_F | \bar{x}_E)$, for some subset E , we define evidence potentials $\delta(x_i, \bar{x}_i)$, for $i \in E$, and multiply the joint probability by the product of these potentials. This simply reduces to multiplying $\psi(x_i)$ by $\delta(x_i, \bar{x}_i)$, for $i \in E$. In particular, we define:

$$\psi_i^E(x_i) \triangleq \begin{cases} \psi_i(x_i) \delta(x_i, \bar{x}_i) & i \in E \\ \psi_i(x_i) & i \notin E, \end{cases} \quad (4.5)$$

 and substitute in Eq. (4.1) to obtain:

$$p(x | \bar{x}_E) = \frac{1}{Z^E} \left(\prod_{i \in V} \psi_i^E(x_i) \prod_{(i,j) \in \mathcal{E}} \psi(x_i, x_j) \right), \quad (4.6)$$

where $Z^E = \sum_x \left(\prod_{i \in V} \psi^E(x_i) \prod_{(i,j) \in E} \psi(x_i, x_j) \right)$. Note that the original Z vanishes.

In summary, the parameterization of unconditional distributions and conditional distributions on trees is formally identical, involving a product of potential functions associated with each node and each edge in the graph. We can thus proceed without making any special distinction between the unconditional case and the conditional case.

Are there any special features of directed trees that we lose in working exclusively with the undirected formalism? One feature of the parameterization for directed trees is that any summation of the form $\sum_{x_j} p(x_j | x_i)$ is necessarily equal to one, and does not need to be performed explicitly. Indeed, in the unconditional case, we can arrange things such that all sums are of this form, by choosing an elimination ordering that begins at the leaves and proceeds backward to the root. (This shows that the normalization factor Z is necessarily equal to one in the unconditional case). When we condition, however, the resulting product of potentials is unnormalized (the normalization factor Z^E is no longer one), and we are brought closer to the general undirected case. It is still the case that we can “prune” any subtree that contains only variables that are not conditioned on, by again eliminating backwards. We view this as an implementation detail, however, assuming that any implementation of an inference algorithm will be smart enough to prune such subtrees at the outset. We then find ourselves in a situation in which the leaves of the tree are evidence nodes, and all of the sums have to be performed explicitly. In this case, there is no essential difference between the directed case and the undirected case, and in developing the general algorithm for inference on trees, it is convenient to focus exclusively on the latter.

4.1.2 From elimination to message-passing

In this section and the following section, we derive the SUM-PRODUCT algorithm, a general algorithm for probabilistic inference on trees. The algorithm involves a simple mathematical update equation—a sum over a product of potentials—applied once for each outgoing edge at each node. We derive this update equation from the point of view of the ELIMINATE algorithm. We subsequently prove that a more general algorithm based on this update equation finds all (singleton) marginals simultaneously.

Let us begin by returning to ELIMINATE, but specializing to the case of a tree. Recall the basic structure of ELIMINATE: (1) Choose an elimination ordering I in which the query node f is the final node; (2) Place all potentials on an active list; (3) Eliminate a node i by removing all potentials referencing the node from the active list, taking the product, summing over x_i , and placing the resulting intermediate factor back on the active list. What are the special features of this procedure when the graph is a tree?

To take advantage of the recursive structure of a tree, we need to specify an elimination ordering I that respects this structure. In particular, we consider elimination orderings that arise from a *depth-first traversal* of the tree. Treat f as a root and view the tree as a directed tree by directing all edges of the tree to point away from f . We now consider any elimination ordering in which a node is eliminated only after all of its children in the directed version of the tree are eliminated. It can be easily verified that such an elimination ordering proceeds inward from the leaves, and generates elimination cliques of size at most two (showing that the tree-width of a tree is equal to

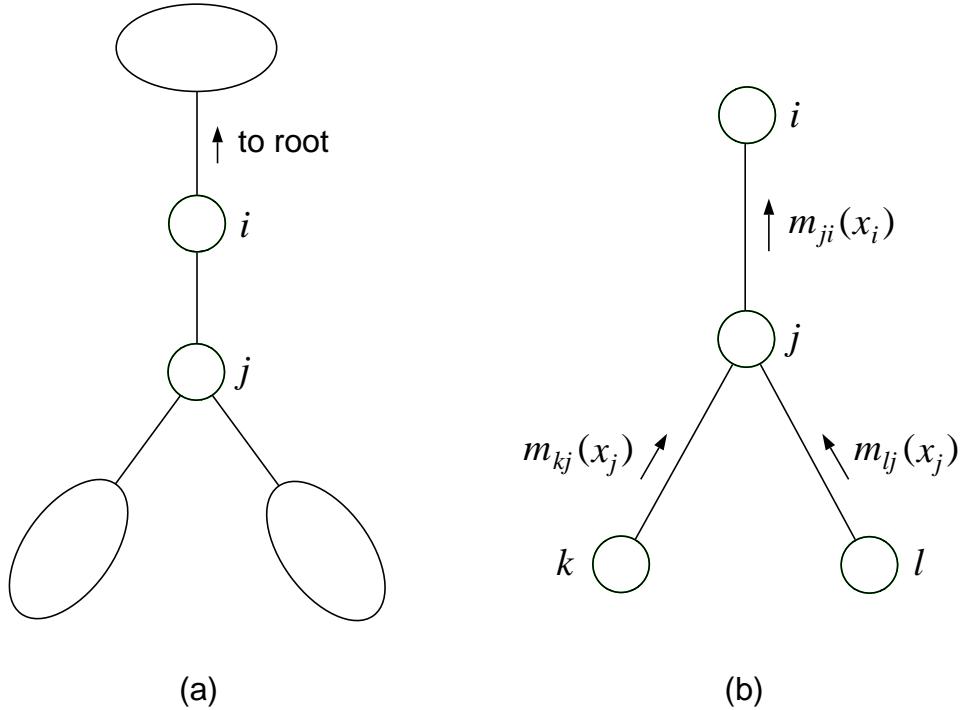


Figure 4.2: (a) A fragment of an undirected graph. Nodes i and j are neighbors, with i nearer to the root than j . (b) The messages that are created when nodes k , l and j are eliminated.

one).

Let us now consider the elimination step. Consider nodes i and j that are neighbors in the tree, where i is closer to the root than j (see Figure 4.2(a)). We are interested in the intermediate factor that is created when j is eliminated. This intermediate factor is a sum over a product of certain potentials. Which potentials are these? Clearly $\psi(x_i, x_j)$ is one of these potentials, given that it references x_j and given that i has yet to be eliminated. Also, $\psi^E(x_j)$ will appear. We can also exclude a number of possibilities. In particular, none of the potentials in the product can reference any variable in the subtree below j , given that all of these variables have already been eliminated. Moreover, none of these potentials can reference any other variable outside the subtree, due to the assumption that the graph is a tree. That is, for a node k in the subtree and a node l outside of the subtree, there can be no potential $\psi(x_k, x_l)$ in the probability model. Thus, when eliminating nodes in the subtree, we can never introduce any variable outside of the subtree into a summand and thus into an intermediate factor.

We have shown that the intermediate factor created by the sum over x_j is a function solely of x_i . Let us introduce the notation " $m_{ji}(x_i)$ " to denote this term, where the first subscript denotes the variable being eliminated and the second subscript denotes the (sole) remaining neighbor of the variable (the "bucket" in the language of Section ??). Note that the latter index is superfluous in the context of ELIMINATE—it is determined by the graph structure and the elimination ordering—but

it will be needed in the context of the more general SUM-PRODUCT algorithm.

We refer to the intermediate factor $m_{ji}(x_i)$ as a “message” that j sends to i . As suggested by Figure 4.2(b), we can think of this message as “flowing” along the edge linking j to i .

Let us now consider the mathematical operation that creates the message $m_{ji}(x_i)$ in more detail. In particular, consider the potentials that are selected from the active list when we eliminate node j —the potentials that reference x_j . As we mentioned earlier, the potentials $\psi(x_i, x_j)$ and $\psi^E(x_j)$ are among the potentials selected. The other potentials that are selected are those created in earlier elimination steps in which the neighbors of node j (other than i) are eliminated. As shown in Figure 4.2(b), these steps can be viewed as creating messages $m_{kj}(x_j)$, messages that flow from each neighbor k —where $k \in \mathcal{N}(j) \setminus i$ —to j .

Thus, following the protocol of the ELIMINATE algorithm, to eliminate x_j we take the product over all potentials that reference x_j and sum over x_j :

$$m_{ji}(x_i) = \sum_{x_j} \left(\psi^E(x_j) \psi(x_i, x_j) \prod_{k \in \mathcal{N}(j) \setminus i} m_{kj}(x_j) \right). \quad (4.7)$$

This is the intermediate factor (“message”) that j sends to i .

Finally, let us consider the final node f in the elimination ordering I . All other nodes have been eliminated when we arrive at f , and thus messages $m_{ef}(x_f)$ have been computed for each of the neighbors $e \in \mathcal{N}(f)$. These messages, and the potential $\psi^E(x_f)$, are the only terms on the active list at this point. Thus, again following the protocol of ELIMINATE, we write the marginal of x_f as the following product:

$$p(x_f | \bar{x}_E) \propto \psi^E(x_f) \prod_{e \in \mathcal{N}(f)} m_{ef}(x_f), \quad (4.8)$$

where the proportionality constant is obtained by summing the right-hand side with respect to x_f .

Eqs. (4.7) and (4.8) provide a concise mathematical summary of the ELIMINATE algorithm, for the special case of a tree. Leaving behind the algorithmic details of ELIMINATE, we see that probabilistic inference essentially involves solving a coupled system of equations in the variables $m_{ji}(x_i)$. To compute $p(x_f)$, we solve these equations in an order that corresponds to a depth-first traversal of a directed tree in which f is the root.

4.1.3 The SUM-PRODUCT algorithm

In this section we show that Eqs. (4.7) and (4.8) suffice for obtaining not only a single marginal, but also for obtaining *all* of the marginals in the tree. The (somewhat magical) fact is that we can obtain all marginals by simply doubling the amount of work required to compute a single marginal. In particular, as we will show, after having passed messages inward from the leaves of the tree to an (arbitrary) root, we simply pass messages from the root back out to the leaves, again using Eq. (4.7) at each step. The net effect is that a single message will flow in both directions along each edge. Once all such messages have been computed, we invoke Eq. (4.8) independently at each node; this yields the desired marginals.

One way to understand why this algorithm works is to consider the naive approach of computing all marginals by using a different elimination ordering for each marginal. Consider in particular the tree fragment shown in Figure 4.3(a). To compute the marginal of X_1 using elimination, we eliminate X_4 and X_3 , which, as we have seen, involves computing messages $m_{42}(x_2)$ and $m_{32}(x_2)$ that are sent to X_2 . We subsequently eliminate X_2 , which creates a message $m_{21}(x_1)$ that is sent from X_2 to X_1 .

Now suppose that we wish to compute the marginal at X_2 using elimination. As shown in Figure 4.3(b), we eliminate X_4 , X_3 , and X_1 , passing messages $m_{42}(x_2)$, $m_{32}(x_2)$ and $m_{12}(x_2)$ to X_2 . The message $m_{12}(x_2)$ is new, but (crucially) $m_{42}(x_2)$ and $m_{32}(x_2)$ are the same messages as computed earlier. Similarly, if we wish to compute the marginal at X_4 , as shown in Figure 4.3(c), we need a new message $m_{24}(x_4)$, but we can reuse the messages $m_{42}(x_2)$ and $m_{12}(x_2)$. In general, **if we compute a message for each direction along each edge in the tree**, as shown in Figure 4.3(d), **we can obtain all singleton marginals**.

The idea that messages can be “reused” is important. In effect we can achieve the effect of computing over all possible elimination orderings (a huge number) by computing all possible messages (a small number). This is the key insight behind the SUM-PRODUCT algorithm.

The SUM-PRODUCT algorithm is based on Eqs. (4.7), (4.8), and a “protocol” that determines when any one of these equations can be invoked. The protocol is given as follows:

Message-Passing Protocol. *A node can send a message to a neighboring node when (and only when) it has received messages from all of its other neighbors.*

There are two principal ways to implement algorithms that respect this protocol. The first (and most direct) way is to interpret the protocol as the specification of a parallel algorithm. In particular, let us view each node as a processor, and assume that the node can repeatedly poll its incoming edges for the presence of messages. For a node of degree d , whenever messages have arrived on any subset of $d - 1$ edges, the node computes a message for the remaining edge and delivers the message along that edge.

An example is shown in Figure 4.4. We **assume a synchronous parallel algorithm**, and at each step show the messages that are delivered along the edges. Note that **messages start to flow in from the leaves**. Note also that when the algorithm terminates, it is the case that a pair of messages have been computed for each edge, one for each direction. Finally, note that all incoming messages are eventually computed for each node, and that Eq. (4.8) can therefore be invoked at each node to compute the node marginal.

For this algorithm to be meaningful in general, we need to insure that all messages will eventually be computed and delivered; that is, that the algorithm will never “block.” We provide a proof that the protocol is non-blocking in Corollary ?? below.

We can also consider **sequential implementations of the SUM-PRODUCT algorithm**, in which messages are computed according to a particular “schedule.” One such schedule (a schedule that is widely used in practice) is a two-phase schedule based on depth-first traversal from an arbitrary root node.² **In the first phase, messages flow inward from the leaves toward the root** (as in Section 4.1.2).

²The original graph may have been a directed tree, with a corresponding root node. The “root” that is designated for the purposes of the SUM-PRODUCT algorithm is unrelated to this root node.

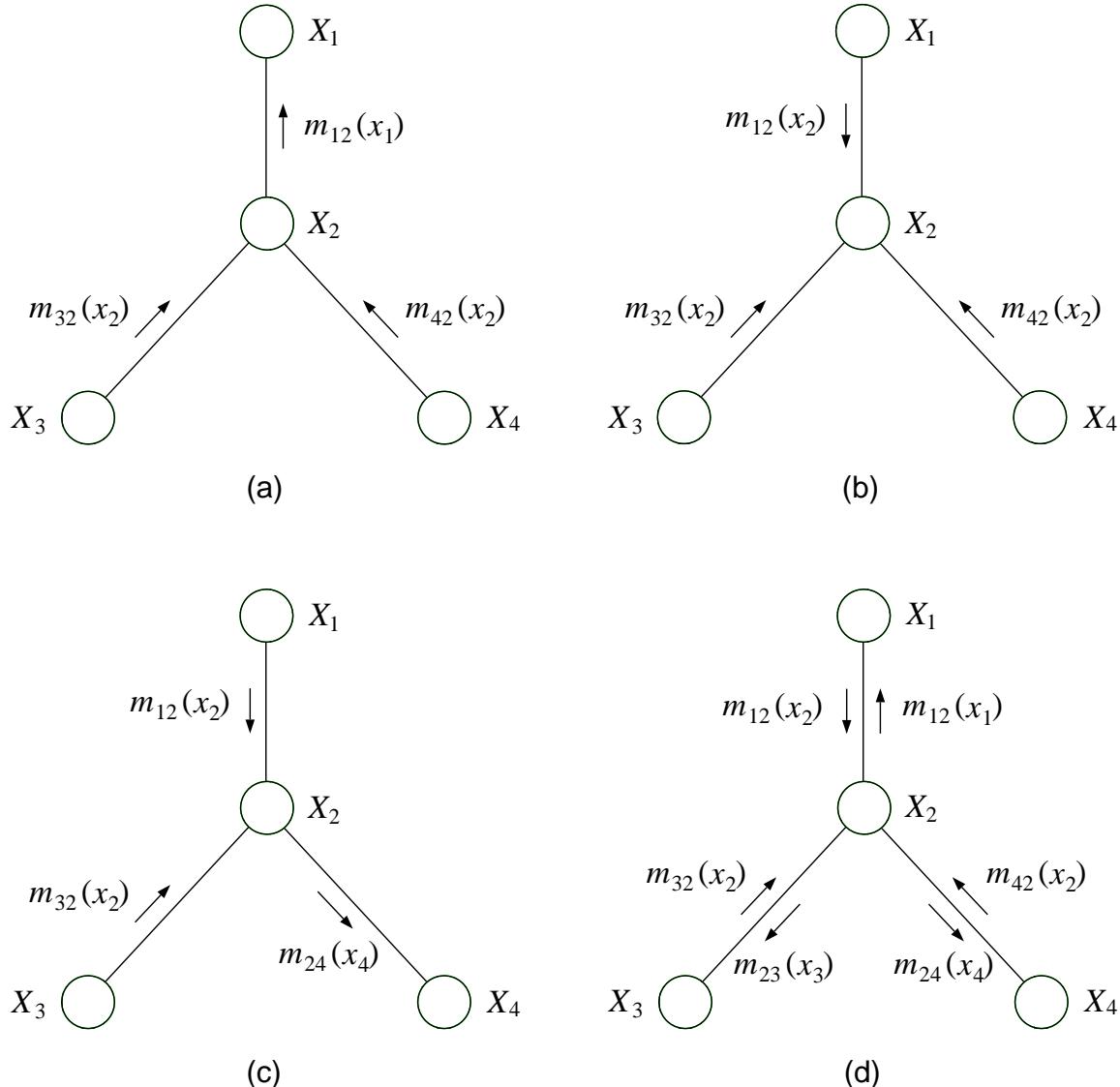


Figure 4.3: (a) The messages formed when computing the marginal of X_1 . (b) The messages formed when computing the marginal of X_2 . (c) The messages formed when computing the marginal of X_4 . (d) All of the messages needed to compute all singleton marginals.

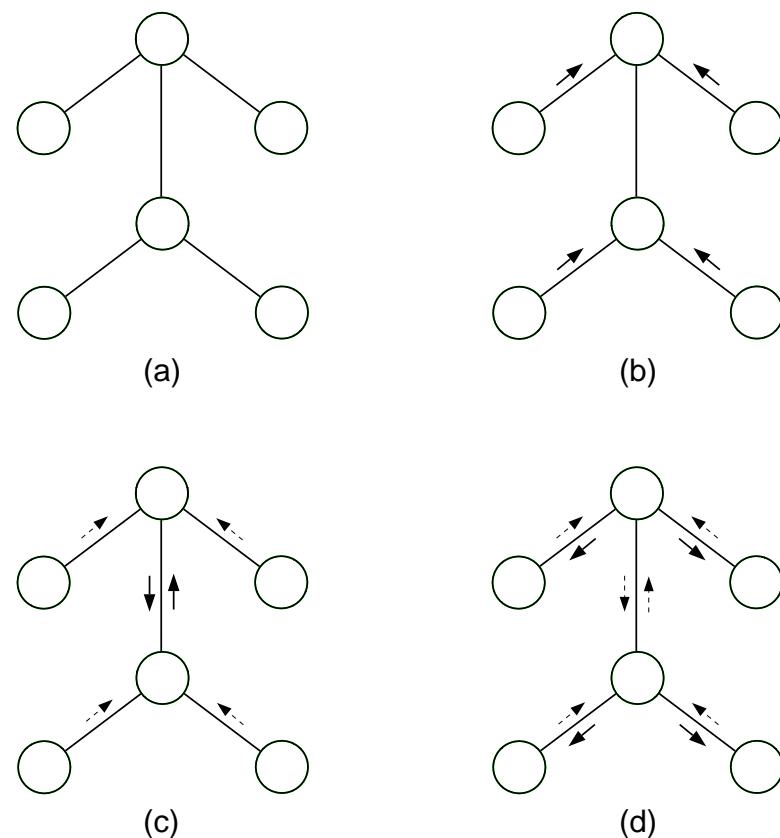


Figure 4.4: Message-passing under a synchronous parallel algorithm. The solid arrows are the messages passed at a given time step, and the dashed arrows are those passed on earlier time steps.

```

SUM-PRODUCT( $\mathcal{T}, E$ )
  EVIDENCE( $E$ )
   $f = \text{CHOSEROOT}(\mathcal{V})$ 
  for  $e \in \mathcal{N}(f)$  
    COLLECT( $f, e$ ) 从 f 递归下去，到叶子节点递归结束返回计算每一个 m_{ji}
  for  $e \in \mathcal{N}(f)$ 
    DISTRIBUTE( $f, e$ )
  for  $i \in \mathcal{V}$ 
    COMPUTEMARGINAL( $i$ )

  EVIDENCE( $E$ )
    for  $i \in E$ 
       $\psi^E(x_i) = \psi(x_i)\delta(x_i, \bar{x}_i)$ 
    for  $i \notin E$ 
       $\psi^E(x_i) = \psi(x_i)$ 

  COLLECT( $i, j$ ) 
    for  $k \in \mathcal{N}(j) \setminus i$ 
      COLLECT( $j, k$ )
    SENDMESSAGE( $j, i$ )

  DISTRIBUTE( $i, j$ )
    SENDMESSAGE( $i, j$ )
    for  $k \in \mathcal{N}(j) \setminus i$ 
      DISTRIBUTE( $j, k$ )

  SENDMESSAGE( $j, i$ )
   $m_{ji}(x_i) = \sum_{x_j} (\psi^E(x_j) \psi(x_i, x_j) \prod_{k \in \mathcal{N}(j) \setminus i} m_{kj}(x_j))$ 

  COMPUTEMARGINAL( $i$ )
   $p(x_i) \propto \psi^E(x_i) \prod_{j \in \mathcal{N}(i)} m_{ji}(x_i)$ 

```

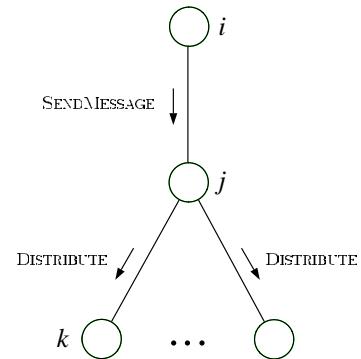
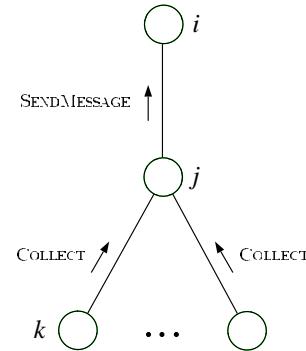


Figure 4.5: A sequential implementation of the SUM-PRODUCT algorithm for a tree $\mathcal{T}(\mathcal{V}, \mathcal{E})$. The algorithm works for any choice of root node, and thus we have left CHOSEROOT unspecified. A call to COLLECT causes messages to flow inward from the leaves to the root. A subsequent call to DISTRIBUTE causes messages to flow outward from the root to the leaves. After these calls have returned, the singleton marginals can be computed locally at each node.

In the second phase—which is initiated once all incoming messages have been received by the root node—messages flow outward from the root toward the leaves. In Figure 4.5, we show how such a schedule can be implemented via a pair of recursive function calls. In Exercise ??, we ask the reader to show that this schedule respects the Message-Passing Protocol, and to show that the overall effect of the schedule is that a single message flows in each direction along each and every edge.

4.1.4 Proof of correctness of the SUM-PRODUCT algorithm³

[Section not yet written].

4.2 Factor graphs and the SUM-PRODUCT algorithm

The graphical model representations that we have discussed thus far—directed and undirected graphical models—aim at characterizing probability distributions in terms of conditional independence statements. Factor graphs, an alternative graphical representation of probability distributions, aim at capturing factorizations. As we have discussed (see Section ??), while closely related, conditional independence and factorization are not exactly the same concepts. Recall in particular our discussion of the parameterization of the complete graph on three nodes. This graph makes no conditional independence assertions, and the corresponding parameterization is simply the arbitrary potential $\psi(x_1, x_2, x_3)$. However, we may be interested in endowing the potentials with algebraic structure, for example:

$$\psi(x_1, x_2, x_3) = f_a(x_1, x_2)f_b(x_2, x_3)f_c(x_1, x_3), \quad (4.9)$$

for given functions f_a , f_b and f_c . Such a factorized potential defines a proper subset of the family of probability distributions associated with the complete graph, a subset which has no interpretation in terms of conditional independence. Factor graphs provide a convenient way to represent subsets of this kind.

In the following section, we introduce the general factor graph representation, and discuss its relationships to directed and undirected graphs. We then focus on the special case of *factor trees* (factor graphs that are trees), and describe the variant of the SUM-PRODUCT algorithm that is geared to factor trees.

4.2.1 Factor graphs

Given a set of variables $\{x_1, x_2, \dots, x_n\}$, we let \mathcal{C} denote a set of subsets of $\{1, 2, \dots, n\}$. Thus, for example, given variables $\{x_1, x_2, x_3, x_4, x_5\}$, we might have $\mathcal{C} = \{\{1, 3\}, \{3, 4\}, \{2, 4, 5\}, \{1, 3\}\}$. Note that \mathcal{C} is a *multiset*—we allow the same subset of indices to appear multiple times. To avoid ambiguity, we therefore index the members of \mathcal{C} using an *index set* \mathcal{F} ; thus, $\mathcal{C} = \{C_s : s \in \mathcal{F}\}$.

³This section can be skipped without loss of continuity.

To each index $s \in \mathcal{F}$, we associate a *factor* $f_s(x_{C_s})$, a function on the subset of variables indexed by C_s . In our example, letting $\mathcal{F} = \{a, b, c, d\}$ denote the indices, the factors are $f_a(x_1, x_3)$, $f_b(x_3, x_4)$, $f_c(x_2, x_4, x_5)$ and $f_d(x_1, x_3)$.

Note also that there is no assumption that the subsets \mathcal{C} correspond to cliques of an underlying graph. Indeed, at this point we do not have any graph structure in mind— \mathcal{C} is just an arbitrary collection of subsets of indices.

Given a collection of subsets and the associated factors, we define a multivariate function on the variables $\{x_1, x_2, \dots, x_n\}$ by taking the product:

$$f(x_1, x_2, \dots, x_n) \triangleq \prod_{s=1}^S f_s(x_{C_s}). \quad (4.10)$$

Our goal will be to define a graphical representation of this function that will permit the efficient evaluation of *marginal functions*—functions of a single variable obtained by summing over all other variables.

Factorized functions in the form of Eq. (4.11) occur in many areas of mathematics, and the methods that we describe in this section has numerous applications outside of probability theory. Our interest, however, will be focused on factorized representations of probability distributions, and indeed the factorized probability distributions associated with directed and undirected graphical models provide examples of the general product-of-factors in Eq. (4.11).⁴

We now introduce a graphical representation of Eq. (4.11). This graphical representation—the *factor graph*—differs from directed and undirected graphical models in that it includes explicit nodes for the factors as well as the variables. We use round nodes to represent the variables and square nodes to represent the factors.

Formally, a factor graph is a bipartite graph $\mathcal{G}(\mathcal{V}, \mathcal{F}, \mathcal{E})$, where the vertices \mathcal{V} index the variables and the vertices \mathcal{F} index the factors. The edges \mathcal{E} are obtained as follows: each factor node $s \in \mathcal{F}$ is linked to all variable nodes in the subset C_s . These are the only edges in the graph.

An example of a factor graph is shown in Figure 4.6. This graph represents the factorized function:

$$f(x_1, x_2, x_3, x_4, x_5) = f_a(x_1, x_3)f_b(x_3, x_4)f_c(x_2, x_4, x_5)f_d(x_1, x_3). \quad (4.11)$$

Note that $f_a(x_1, x_3)$ and $f_d(x_1, x_3)$ refer to the same set of variables. In an undirected graphical model these factors would be collapsed into a single potential function, $\psi(x_1, x_3)$. In a factor graph these functions are allowed to maintain a separate identity.

It will prove useful to define neighborhood functions on the nodes of a factor graph. In particular, let $\mathcal{N}(s) \subset \mathcal{V}$ denote the set of neighbors of a factor node $s \in \mathcal{F}$, and let $\mathcal{N}(i) \subset \mathcal{F}$ denote the set of neighbors of a variable node $i \in \mathcal{V}$. Note that $\mathcal{N}(s)$ refers to the indices of all variables referenced by the factor f_s , and is identical to the subset C_s introduced earlier. On the other hand, the neighborhood set $\mathcal{N}(i)$, for a variable node i , is the set of all factors that reference the variable x_i .

Directed and undirected graphical models can be readily converted to factor graphs. For example, the directed graphical model shown in Figure 4.7(a) can be represented as a factor graph

⁴The normalization factor Z in the parameterization of undirected graphical models can be treated as a factor

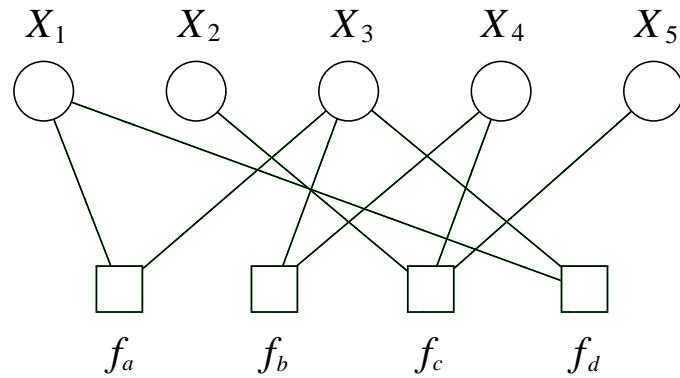


Figure 4.6: An example of a factor graph.

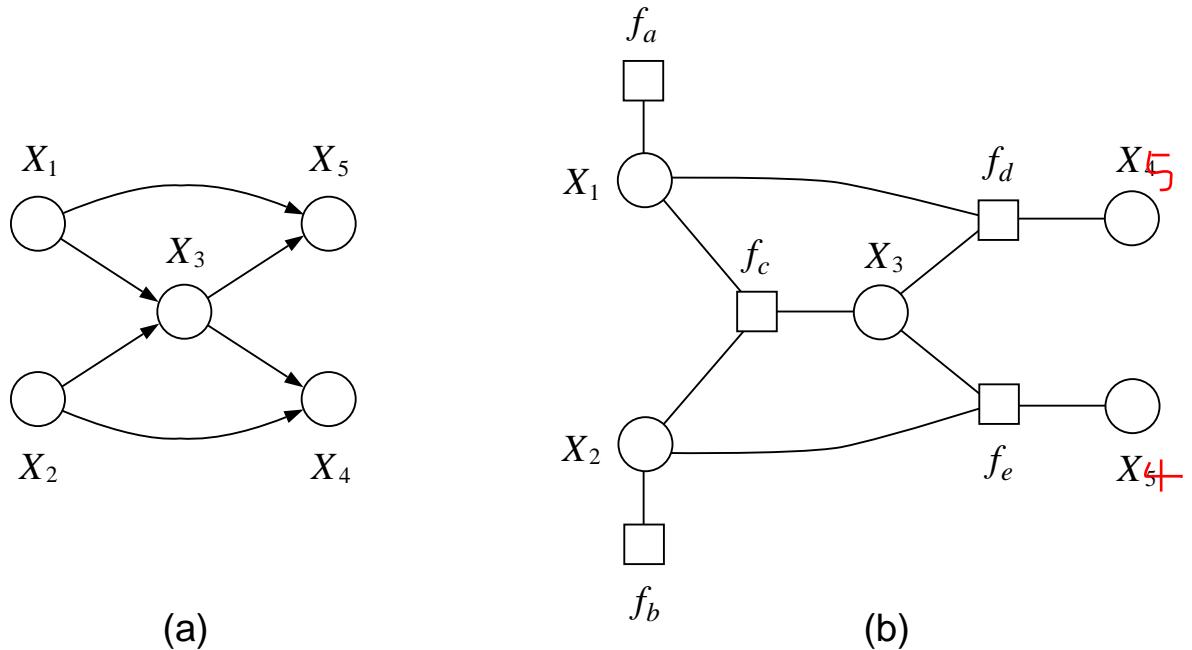


Figure 4.7: (a) A directed graphical model. (b) The corresponding factor graph. Note that there are six factor nodes, one for each local conditional probability in the directed graph.

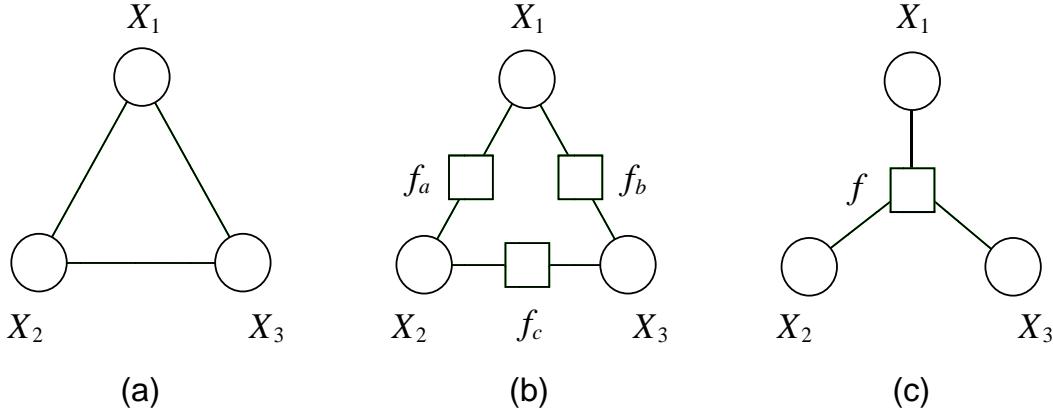


Figure 4.8: (a) An undirected graphical model provides no information about possible factorizations of the potential function associated with a given clique. (b) The factor graph corresponding to the factorized potential $\psi(x_1, x_2, x_3) = f_a(x_1, x_2)f_b(x_2, x_3)f_c(x_1, x_3)$. (c) The factor graph corresponding to the non-factorized potential $\psi(x_1, x_2, x_3) = f(x_1, x_2, x_3)$.

as shown in Figure 4.7(b).⁵

By representing each factor as a node in the graph, factor graphs provide a more fine-grained representation of probability distributions than is provided by directed and undirected graphical models. In particular, returning to the complete graph on three nodes shown in Figure 4.8(a), factor graphs make it possible to display fine-grained assumptions about the parameterization: Figure 4.8(b) shows the factor graph corresponding to the general potential $\psi(x_1, x_2, x_3)$, while Figure 4.8(c) shows the factor graph corresponding to the factorized potential in Eq. (4.9).

It is worth noting that it is always possible to mimic the fine-grained representation of factor graphs within the directed and undirected formalisms, so that formally factor graphs provide no additional representational power. For example, in Figure 4.9(a) we show an undirected graph that can represent the factorization in Eq. (4.9). In this graph, we have introduced three new random variables, Z_1 , Z_2 , and Z_3 . These variables are indicator variables picking out particular combinations of the underlying variables X_1 , X_2 and X_3 . Thus, for example, for binary X_1 and X_2 , Z_1 would take on four possible values, one for each pair of values of X_1 and X_2 , and the potential function $\psi(z_1)$ would be set equal to the corresponding value of $f_a(x_1, x_2)$. (We ask the reader to fill in the details of this construction in Exercise ??).

Similarly, in Figure 4.9(b), we show a directed graph that mimics the factorization in Eq. (4.9). In this graph, the three new variables, W_1 , W_2 , and W_3 , are binary variables that are always set equal to one. We set $p(W_1 = 1 | x_1, x_2)$ to the corresponding value of $f_a(x_1, x_2)$. (We again ask the reader to supply the details in Exercise ??).

associated with the empty set—which is appropriate given that it is a constant.

⁵In general, in the directed case each factor is a local conditional probability, and the subsets C_s correspond to “families” consisting of a node and its parents. Given that we do not assume that the subsets C_s correspond to cliques of an underlying graph, we do not need to “moralize” in the factor graph formalism. This is consistent with the fact that the factor graph does not attempt to represent conditional independencies.

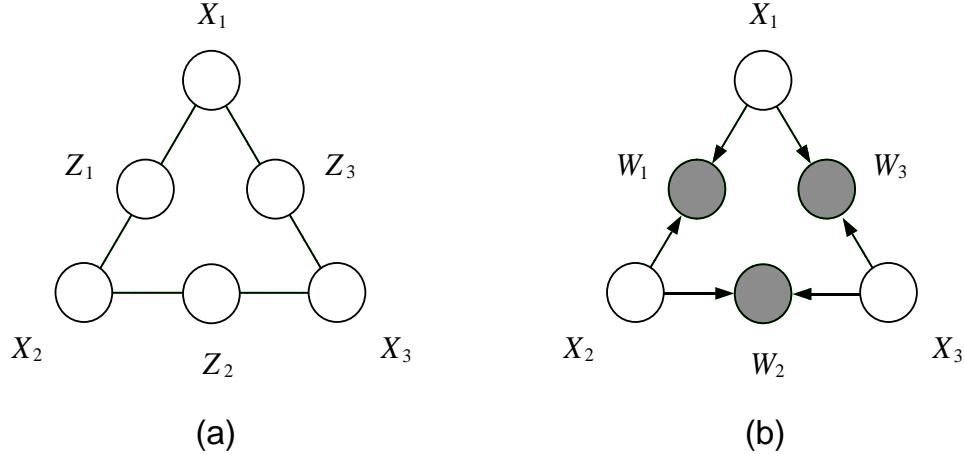


Figure 4.9: (a) An undirected graph that mimics the factorization shown in Figure 4.8(b) for appropriate choices of the indicator variables Z_i . (b) A directed graph that mimics the factorization shown in Figure 4.8(b) for appropriate choices of the indicator variables W_i .

In general, by introducing additional variables in a directed or undirected graph, we can mimic the factorization that is made explicit in the factor graph. However, this procedure is arguably rather artificial, and the factor graph representation provides a natural complement to undirected or directed graphs for situations in which a fine-grained representation of potentials is desired.

4.2.2 The SUM-PRODUCT algorithm for factor trees

We now turn to the inference problem for factor graphs. As before, our goal is to compute all singleton marginal probabilities under the factorized representation of the joint probability. In this section we show how to do this for factor graphs that are trees.

A factor graph is defined to be a *factor tree* if the undirected graph obtained by ignoring the distinction between variable nodes and factor nodes is an undirected tree. Restricting ourselves to trees, we define a variant of the SUM-PRODUCT algorithm that provides all singleton marginal probabilities for factor trees.

As in the earlier SUM-PRODUCT algorithm, we define messages that flow along the edges of the graph. In the case of factor trees, there are two kinds of messages: messages ν that flow from variable nodes to factor nodes, and messages μ that flow from factor nodes to variable nodes.

These messages take the following form. We first consider the messages that flow from variable nodes to factor nodes. As depicted in Figure 4.10(a), the message $\nu_{is}(x_i)$ that flows between the variable node i and the factor node s is computed as follows:

$$\nu_{is}(x_i) = \prod_{t \in \mathcal{N}(i) \setminus s} \mu_{ti}(x_i), \quad (4.12)$$

where the product is taken over all incoming messages to variable node i , other than the message from the factor node s that is the recipient of the message.

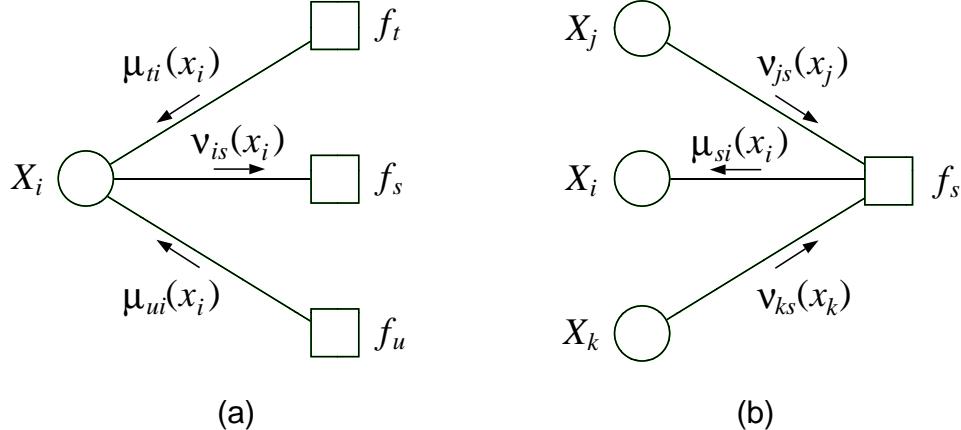


Figure 4.10: (a) The computation of the message $\nu_{is}(x_i)$ that flows from factor node s to variable node i . (b) The computation of the message $\mu_{si}(x_i)$ that flows from variable node i to factor node s .

Similarly, as shown in Figure 4.10(b), a message $\mu_{si}(x_i)$ flows between the factor node s and the variable node i . This message is computed as follows:

$$\mu_{si}(x_i) = \sum_{x_{\mathcal{N}(s) \setminus i}} \left(f_s(x_{\mathcal{N}(s)}) \prod_{j \in \mathcal{N}(s) \setminus i} \nu_{js}(x_j) \right). \quad (4.13)$$

Note that the product is taken over all incoming messages to factor node s , other than the message from the variable node i that is the recipient of the message.

Thus we have a coupled set of equations for a set of messages. As in our earlier SUM-PRODUCT algorithm, a full specification of the algorithm requires a determination of when a given equation can be invoked. The protocol turns out to be exactly the same as the earlier protocol:

Message-Passing Protocol. *A node can send a message to a neighboring node when (and only when) it has received messages from all of its other neighbors.*

In the factor tree case, the protocol applies to both variable nodes and factor nodes.

Finally, once a message has arrived at each node from all of its neighbors, the marginal probability of a node is obtained as follows:

$$p(x_i) \propto \prod_{s \in \mathcal{N}(i)} \mu_{si}(x_i). \quad (4.14)$$

Given the definition of $\nu_{is}(x_i)$ in Eq. (4.12), this can also be written as follows:

$$p(x_i) \propto \nu_{is}(x_i) \mu_{si}(x_i), \quad (4.15)$$

for any $s \in \mathcal{N}(i)$. That is, the marginal probability of node i can be obtained by taking the product of the pair of messages flowing along any edge incident on node i .

A sequential implementation of the SUM-PRODUCT algorithm for factor trees is provided in Figure 4.11.

Consider the example shown in Figure 4.6(a). The factor tree representation of this model is shown in Figure 4.12(b). Let us run through the steps of the SUM-PRODUCT algorithm. In the first step, shown in Figure 4.12(c), the only nodes that are able to send messages are the leaf nodes. These leaf nodes are factor nodes, and the product in Eq. (4.13) is a vacuous product, which by convention we set equal to one. Moreover, the sum in Eq. (4.13) is a vacuous sum. Thus, the message that flows in from a leaf node is simply the factor associated with that node: $\mu_{si}(x_i) = \psi^E(x_i)$, for $i \in \mathcal{V}$.

The second stage in the process is also rather uninteresting. As shown in Figure 4.12(d), the variable nodes X_1 and X_3 are able to send messages in this stage. For each node, the product in Eq. (4.12) is composed of only a single factor, and thus this factor is simply passed along the chain.

Now consider the third stage, shown in Figure 4.12(e). At the factor nodes along the backbone of the chain, a sum is taken over the product of the incoming message and the factor residing at that node. In the case of the message $\mu_{d2}(x_2)$, this yields $\mu_{d2}(x_2) = \sum_{x_1} \psi^E(x_1) \psi(x_1, x_2)$, and, similarly, $\mu_{e2}(x_2) = \sum_{x_3} \psi^E(x_3) \psi(x_2, x_3)$. Note that these messages are the same as the corresponding messages that would pass in a run of the SUM-PRODUCT algorithm for the undirected graph in Figure 4.12(a). That is, we have: $\mu_{d2}(x_2) = m_{12}(x_2)$, and $\mu_{e2}(x_2) = m_{32}(x_2)$.

Finally, in Figure 4.12(f), Figure 4.12(g), and Figure 4.12(h), we show the remaining steps of the algorithm. The reader can again verify a correspondence with the messages that would be computed in Figure 4.12(a): $\mu_{d1}(x_1) = m_{21}(x_1)$ and $\mu_{e3}(x_3) = m_{23}(x_3)$. By the end of the algorithm, a message has passed in both directions along every edge.

In general, if we start with a graph that is an undirected tree and convert to a factor graph, then we find that there is a direct relationship between the “ m messages” of the SUM-PRODUCT algorithm for the undirected graph and the “ μ messages” of the SUM-PRODUCT algorithm for the factor graph. Consider the graph fragment shown in Figure 4.13(a) and the corresponding factor graph representation in Figure 4.13(b). We claim that $m_{ji}(x_i)$ in the undirected graph is equal to $\mu_{si}(x_i)$ in the factor graph. Indeed, we have:

$$\mu_{si}(x_i) = \sum_{x_{\mathcal{N}(s) \setminus i}} \left(f_s(x_{\mathcal{N}(s)}) \prod_{j \in \mathcal{N}(s) \setminus i} \nu_{js}(x_j) \right) \quad (4.16)$$

$$= \sum_{x_j} \psi(x_i, x_j) \nu_{js}(x_j) \quad (4.17)$$

$$= \sum_{x_j} \psi(x_i, x_j) \prod_{t \in \mathcal{N}(j) \setminus s} \mu_{tj}(x_j) \quad (4.18)$$

$$= \sum_{x_j} \left(\psi^E(x_j) \psi(x_i, x_j) \prod_{t \in \mathcal{N}'(j) \setminus s} \mu_{tj}(x_j) \right), \quad (4.19)$$

where $\mathcal{N}'(j)$ denotes the neighborhood of j , omitting the singleton factor node associated with $\psi^E(x_j)$. We see that the expression for $\mu_{si}(x_i)$ is formally identical to the update equation for $m_{ji}(x_i)$ in Eq. (4.7).

```

SUM-PRODUCT( $\mathcal{T}$ ,  $E$ )
  EVIDENCE( $E$ )
   $f = \text{CHOSEROOT}(\mathcal{V})$ 
  for  $s \in \mathcal{N}(f)$ 
     $\mu\text{-COLLECT}(f, s)$ 
    for  $s \in \mathcal{N}(f)$ 
       $\nu\text{-DISTRIBUTE}(f, s)$ 
    for  $i \in \mathcal{V}$ 
      COMPUTEMARGINAL( $i$ )

 $\mu\text{-COLLECT}(i, s)$ 
  for  $j \in \mathcal{N}(s) \setminus i$ 
     $\nu\text{-COLLECT}(s, j)$ 
     $\mu\text{-SENDMESSAGE}(s, i)$ 

 $\nu\text{-COLLECT}(s, i)$ 
  for  $t \in \mathcal{N}(i) \setminus s$ 
     $\mu\text{-COLLECT}(i, t)$ 
     $\nu\text{-SENDMESSAGE}(i, s)$ 

 $\mu\text{-DISTRIBUTE}(s, i)$ 
   $\mu\text{-SENDMESSAGE}(s, i)$ 
  for  $t \in \mathcal{N}(i) \setminus s$ 
     $\nu\text{-DISTRIBUTE}(i, t)$ 

 $\nu\text{-DISTRIBUTE}(i, s)$ 
   $\nu\text{-SENDMESSAGE}(i, s)$ 
  for  $j \in \mathcal{N}(s) \setminus i$ 
     $\mu\text{-DISTRIBUTE}(s, j)$ 

 $\mu\text{-SENDMESSAGE}(s, i)$ 
   $\mu_{si}(x_i) = \sum_{x_{\mathcal{N}(s) \setminus i}} (f_s(x_{\mathcal{N}(s)}) \prod_{j \in \mathcal{N}(s) \setminus i} \nu_{js}(x_j))$ 

 $\nu\text{-SENDMESSAGE}(i, s)$ 
   $\nu_{is}(x_i) = \prod_{t \in \mathcal{N}(i) \setminus s} \mu_{ti}(x_i)$ 

COMPUTEMARGINAL( $i$ )
   $p(x_i) \propto \nu_{is}(x_i) \mu_{si}(x_i)$ 

```

Figure 4.11: A sequential implementation of the SUM-PRODUCT algorithm for a factor tree $\mathcal{T}(\mathcal{V}, \mathcal{F}, \mathcal{E})$. The algorithm works for any choice of root node, and thus we have left CHOSEROOT unspecified. The subroutine EVIDENCE(E) is presented in Figure 4.5.

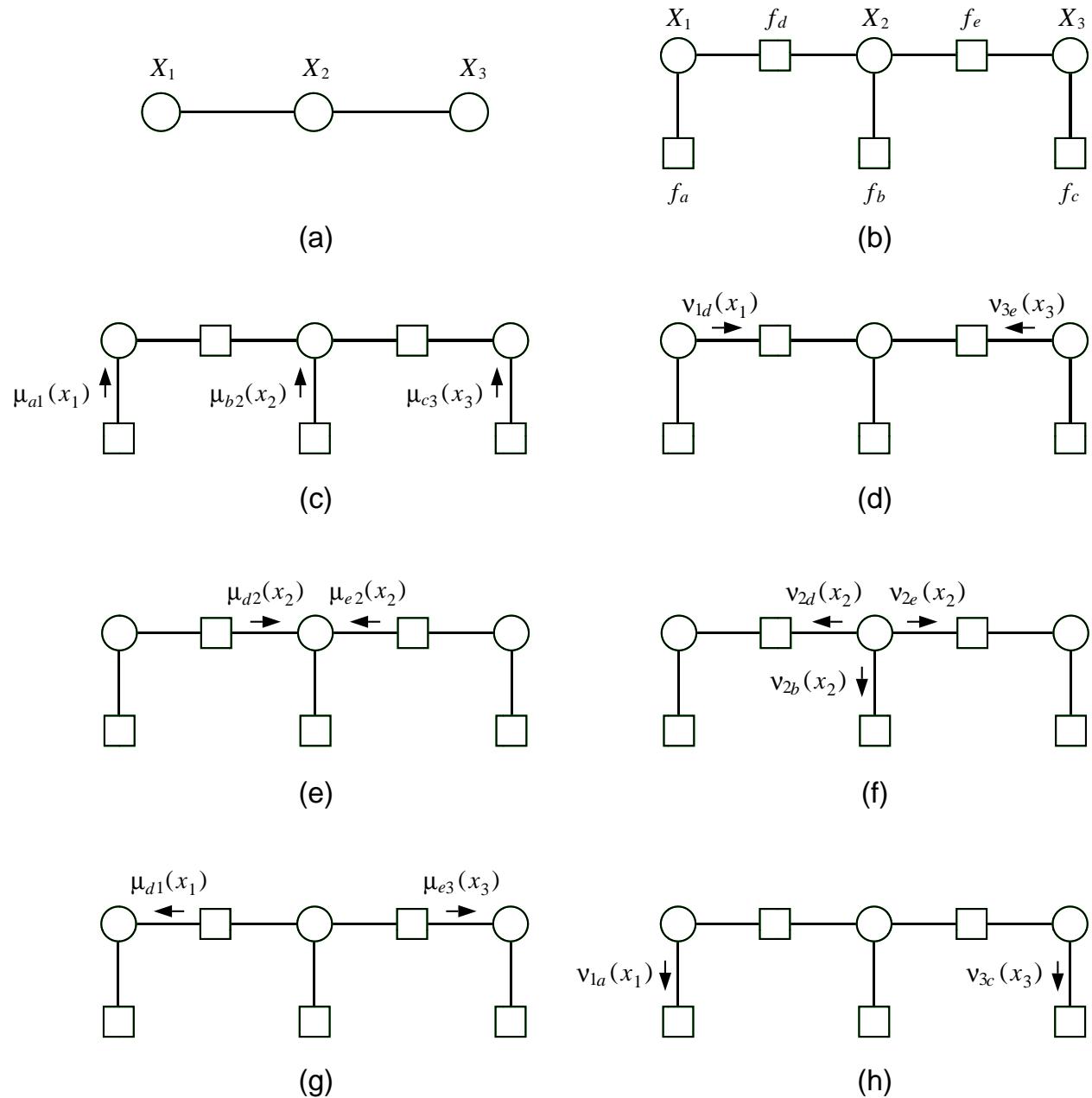


Figure 4.12: (a) A three-node undirected graphical model. (b) The factor tree representation. (c)-(h) A run of the SUM-PRODUCT algorithm on the factor tree.

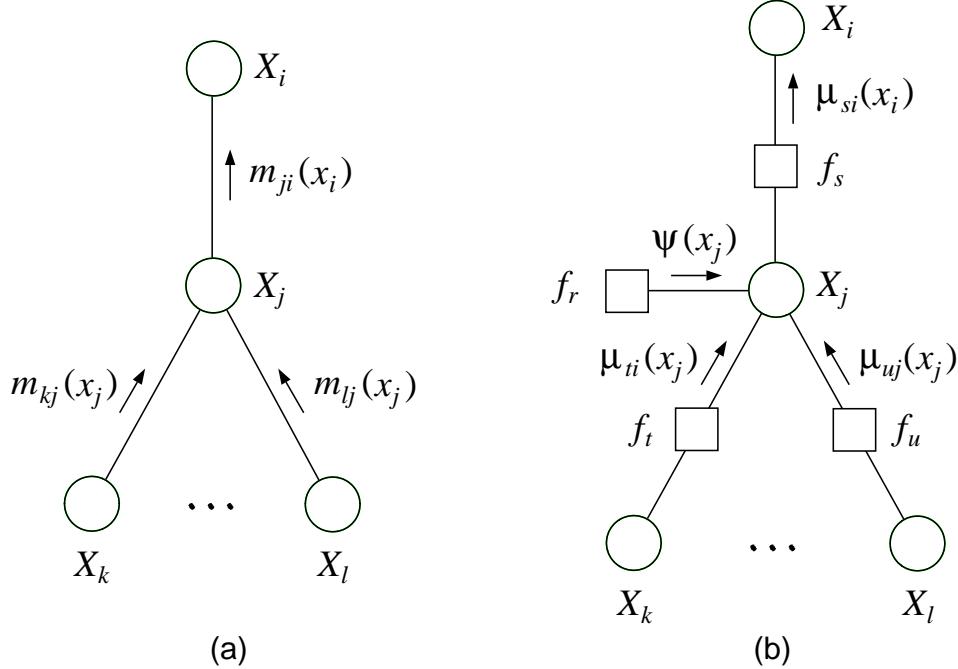


Figure 4.13: (a) A fragment of an undirected tree. (b) The corresponding fragment of a factor tree.

From this observation and an induction argument, it is not difficult to prove that the SUM-PRODUCT algorithm for factor trees is correct for factor trees that are obtained from undirected trees, by simply translating between the two versions of the SUM-PRODUCT algorithm. We leave this as an exercise (Exercise ??). It is also straightforward to develop a standalone proof by induction that the general SUM-PRODUCT algorithm for factor trees is correct, which we again leave as an exercise.

If a graph is originally a tree (undirected or directed), there is little to be gained by translating to the factor graph framework. The payoff for factor graphs arises when we consider various “tree-like” graphs, to which we now turn.

4.2.3 Tree-like graphs

Consider the graph shown in Figure 4.14(a). Assuming that the three-node cluster in the center of the graph is parameterized by a general non-factorized potential function, the probability distribution associated with the graph is given by:

$$p(x) \propto \psi(x_1, x_2)\psi(x_3, x_5)\psi(x_4, x_6)\psi(x_2, x_3, x_4), \quad (4.20)$$

where for simplicity we have neglected the singleton potentials. Although this graph is not a tree, it is “nearly” a tree. In particular, we could replace the three variables X_2 , X_3 , and X_4 with a new “super-variable” Z , whose range is the Cartesian product of the ranges of the three individual

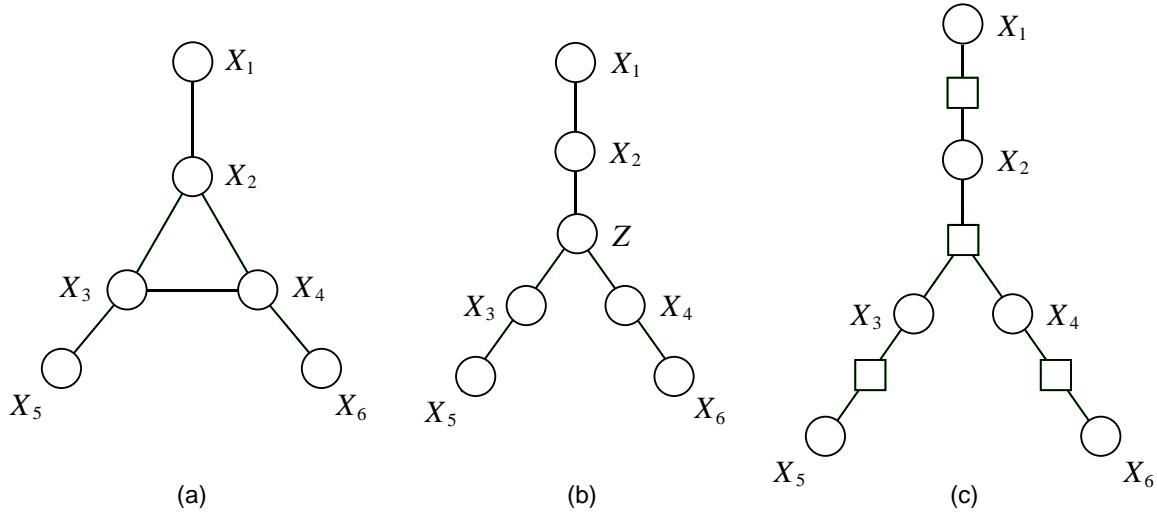


Figure 4.14: (a) An undirected graphical model in which the center cluster of nodes is assumed to be parameterized as a non-factorized potential, $\psi(x_2, x_3, x_4)$. (b) An equivalent undirected model based on the “super-variable” Z . (c) An equivalent factor graph.

variables. By creating new potential functions, $\psi(x_1, z)$, $\psi(x_5, z)$, $\psi(x_6, z)$, and $\psi(z)$, we can mimic the factorization in Eq. (4.20). Moreover, the corresponding undirected graphical model, shown in Figure 4.14(b), is a tree.

We can also capture the probability distribution in Eq. (4.20) using a factor graph. In particular, the graph translates directly to the factor graph shown in Figure 4.14(c). Note that the factor node at the center of the graph has three neighbors—representing the dependency structure of the potential $\psi(x_2, x_3, x_4)$. Note also that the factor graph is a factor tree.

We see that the distribution represented by the tree-like undirected graph in Figure 4.14(a) translates directly to a tree in the factor graph framework. There is no need to invent new variables and new potential functions.

Finally, of most significance is that the SUM-PRODUCT algorithm for factor trees applies directly to the graph in Figure 4.14(c). The fact that the original graph is not a tree is irrelevant—the factor graph *is* a tree, and the algorithm is correct for general factor trees.

In general, if the variables in an undirected graphical model can be clustered into non-overlapping cliques, and the parameterization of each clique is a general, non-factorized potential, then the corresponding factor graph is a tree, and the SUM-PRODUCT applies directly.

4.2.4 Polytrees

A polytree is a tree-like graph that is important enough to merit its own section. In this section we discuss the SUM-PRODUCT algorithm for polytrees, again exploiting the factor graph framework.

As we have discussed, directed trees are essentially equivalent to undirected trees, providing no additional representational capability and no new issues for inference. On the other hand, the

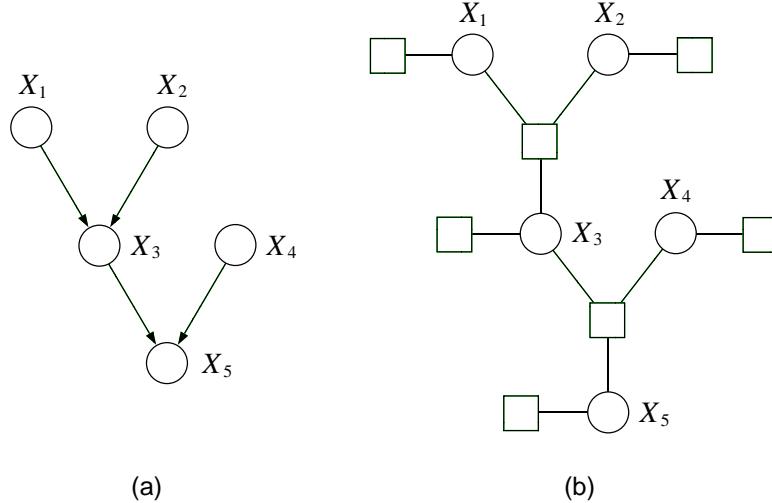


Figure 4.15: (a) A polytree. (b) The factor graph representation of the polytree in (a). Note that the factor graph is a factor tree.

directed graph shown in Figure 4.15(a) is a tree-like graph that does present new capabilities and new issues. As we saw in Chapter 2, the presence of nodes with multiple parents in a directed graph implies a conditional independence semantics that is not available in undirected graphs, including the “explaining-away” semantics that we studied in Chapter 2. Not surprisingly, this semantics has implications for inference, concretely via the conditional probability $p(x_i | x_{\pi_i})$ that links a node with its parents.

A *polytree* is a directed graph that reduces to an undirected tree if we convert each directed edge to an undirected edge. Thus, polytrees have no loops in their underlying undirected graph.

One way to treat polytrees is via the “super-variable” approach. That is, we create a new variable for each combination of a node and its parents (each family) and link the super-variables (with undirected edges). It is easy to see that the resulting graph is a tree. This approach, however, suffers from the inelegance alluded to in the previous section.

Alternatively, we can use factor graphs. In Figure 4.15(b), we show the factor graph corresponding to the polytree in Figure 4.15(a). We see that the factor graph is a tree. Moreover, there is a factor corresponding to each family, representing the conditional probability $p(x_i | x_{\pi_i})$.

The fact that the factor graph corresponding to a polytree is a tree implies that the SUM-PRODUCT algorithm for factor graphs applies directly to polytrees.

Historically, polytrees were an important step along the way in the development of general exact inference algorithms for graphical models. In 1983, Kim and Pearl described a general sum-product-like algorithm for polytrees. As in the case of the SUM-PRODUCT algorithm for factor graphs, this algorithm also involves two kinds of messages—“ λ messages” flowing from children to parents, and “ π messages” flowing from parents to children. The algorithm can be derived readily from the SUM-PRODUCT algorithm for the corresponding factor graph. We present the algorithm in Exercise ??, and ask the reader to provide the derivation.

4.3 Maximum a posteriori probabilities

In this section we discuss a new problem—that of computing *maximum a posteriori probabilities*. Whereas the marginalization problem that we have addressed up until now involves summing over all configurations of sets of random variables, the maximum a posteriori (MAP) problem involves maximizing over such configurations. The problem has two aspects—that of finding the maximal probability and that of finding a configuration that achieves the maximal probability. We begin by focusing on the former problem.⁶

Given a probability distribution $p(x)$, where $x = (x_1, x_2, \dots, x_n)$, given a partition (E, F) of the indices, and given a fixed configuration \bar{x}_E , we wish to compute the maximum a posteriori probability $\max_{x_F} p(x_F | \bar{x}_E)$. Although we use the language of “maximum a posteriori probability” to describe this problem, the conditioning turns out to play little significant role in the problem. Indeed:

$$\max_{x_F} p(x_F | \bar{x}_E) = \max_{x_F} p(x_F, \bar{x}_E) \quad (4.21)$$

$$= \max_x p(x) \delta(x_E, \bar{x}_E) \quad (4.22)$$

$$\triangleq \max_x p^E(x), \quad (4.23)$$

where $p^E(x)$ is the unnormalized representation of conditional probability introduced in Section 3.1.1. We see that without loss of generality we can study the unconditional case. That is, we treat the general problem of maximizing a nonnegative, factorized function of n variables; this includes as a special case the problem of maximizing such a function when some of the variables are held fixed.

It is important to be clear that the MAP problem is quite distinct from the marginalization problem. Naively, one might think that one could solve the MAP problem by first computing the marginal probability for each variable, and then computing the assignment of each variable that maximizes its individual marginal, but this is incorrect. Consider the pair of variables shown in Figure 4.16. The marginal probability of X is maximized by choosing $X = 1$, and the marginal probability of Y is maximized by choosing $Y = 1$. However, the joint probability of the configuration $(X = 1, Y = 1)$ is equal to zero! The maximizing assignment is $(X = 1, Y = 2)$, which has probability 0.36.

Although the MAP problem is distinct from the marginalization problem, its algorithmic solution is quite similar. To see this, let us return to the example shown in Figure 4.17, a directed graphical model with the following factorization:

$$p(x) = p(x_1)p(x_2 | x_1)p(x_3 | x_1)p(x_4 | x_2)p(x_5 | x_3)p(x_6 | x_2, x_5). \quad (4.24)$$

To solve the MAP problem we expand the maximization into component-wise maximizations, and compute:

$$\max_x p(x) = \max_{x_1} \max_{x_2} \max_{x_3} \max_{x_4} \max_{x_5} \max_{x_6} p(x_1)p(x_2 | x_1)p(x_3 | x_1)p(x_4 | x_2)p(x_5 | x_3)p(x_6 | x_2, x_5)$$

⁶There are generalizations of the MAP problem that involve finding a small set of configurations that have high probability, and finding multiple configurations that have maximal probability when the maximum is not unique. In the current section, we restrict ourselves to the simpler problem of finding a single maximum.

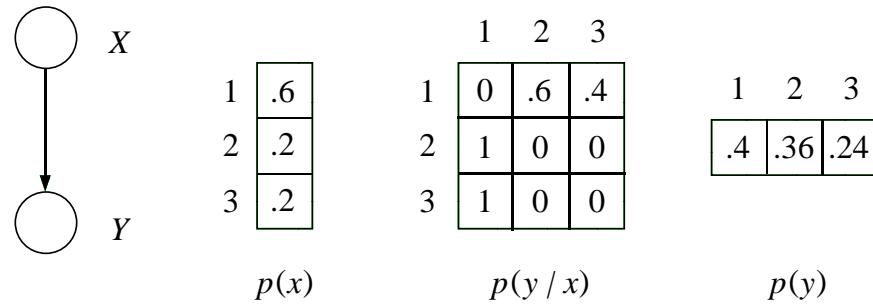


Figure 4.16: The marginal and conditional probabilities for a pair of variables (X, Y) . The maximizing values of the individual marginals are $X = 1$ and $Y = 1$, but the configuration $(X = 1, Y = 1)$ has zero probability.

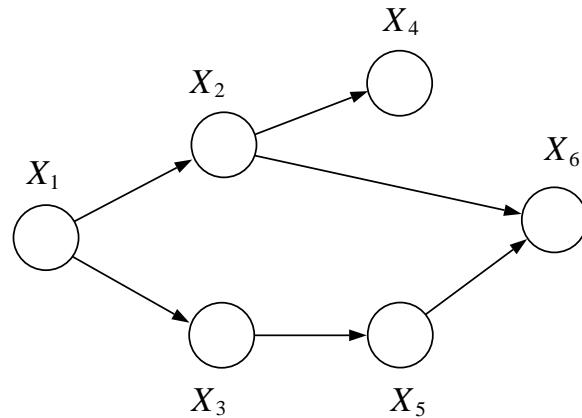


Figure 4.17: A directed graphical model.

```

MAP-ELIMINATE( $\mathcal{G}, E$ )
  INITIALIZE( $\mathcal{G}$ )
  EVIDENCE( $E$ )
  UPDATE( $\mathcal{G}$ )
  MAXIMUM

INITIALIZE( $\mathcal{G}$ )
  choose an ordering  $I$ 
  for each node  $X_i$  in  $\mathcal{V}$ 
    place  $p(x_i | x_{\pi_i})$  on the active list

EVIDENCE( $E$ )
  for each  $i$  in  $E$ 
    place  $\delta(x_i, \bar{x}_i)$  on the active list

UPDATE( $\mathcal{G}$ )
  for each  $i$  in  $I$ 
    find all potentials from the active list that reference  $x_i$  and remove them from the active list
    let  $\phi_i^{\max}(x_{T_i})$  denote the product of these potentials
    let  $m_i^{\max}(x_{S_i}) = \max_{x_i} \phi_i^{\max}(x_{T_i})$ 
    place  $m_i^{\max}(x_{S_i})$  on the active list

MAXIMUM
   $\max_x p^E(x) =$  the scalar value on the active list

```

Figure 4.18: The MAP-ELIMINATE algorithm for solving the maximum a posteriori problem. Note that after the final node has been eliminated in UPDATE, the active list contains a single scalar value, which is the value returned as the maximum by the algorithm.

$$= \max_{x_1} p(x_1) \max_{x_2} p(x_2 | x_1) \max_{x_3} p(x_3 | x_1) \max_{x_4} p(x_4 | x_2) \max_{x_5} p(x_5 | x_3) \max_{x_6} p(x_6 | x_2, x_5).$$

These steps should look familiar from our earlier example of marginalization in this graph. Continuing the computation, we perform the maximization with respect to x_6 , thereby defining an “intermediate factor” that is a function of x_2 and x_5 . Subsequent steps are identical to those of a marginalization computation, with the “sum” operator replaced by the “max” operator.

More generally, all of the derivations that we have presented in this chapter and the previous chapter go through if the “sum” operator is replaced everywhere by the “max” operator. In particular, by making this substitution in ELIMINATE, we obtain a MAP version of ELIMINATE, which we present in Figure 4.18.

The reason that the derivations go through when “sum” is replaced by “max” is that both the “sum-product” pair and the “max-product” pair are examples of an algebraic structure known as a *commutative semiring*. A commutative semiring is a set endowed with two operations—generically

referred to as “addition” and “multiplication”—that obey certain laws. In particular, addition and multiplication are both required to be *associative* and *commutative*. Moreover, multiplication is *distributive* over addition:

$$a \cdot b + a \cdot c = a \cdot (b + c). \quad (4.25)$$

This distributive law played a key role in our derivation of ELIMINATE, in which the “sum” operator repeatedly migrates across the “product” operator. Also, the ability to group and reorder intermediate factors was required in the derivation of the ELIMINATE algorithm. In fact, it can be verified that the associative, commutative and distributive laws are all that are needed to derive the ELIMINATE algorithm and the SUM-PRODUCT algorithm. (Note in particular that we do not require division, an operation that is available in the more restrictive algebraic object known as a *ring*.)

If we let the “max” operator play the role of addition, the fact that “max” distributes over “product”:

$$\max(a \cdot b, a \cdot c) = a \cdot \max(b, c) \quad (4.26)$$

shows that “max-product” is a semiring (given the easy verification that “max” is associative and commutative), and justifies the MAP-ELIMINATE algorithm in Figure 4.18.

A practical problem with the MAP-ELIMINATE algorithm shown in Figure 4.18 is that the products of probabilities tend to underflow. This can be handled by transforming to the log scale, making use of the fact that:

$$\max_x p^E(x) = \max_x \log p^E(x), \quad (4.27)$$

which holds because the logarithm is a monotone function. Given that the logarithm of a product becomes a sum of logarithms, we see that such an implementation essentially involves working with a “max-sum” pair instead of a “max-product” pair. Fortunately, “max-sum” is also a semiring, in which “max” plays the role of addition and “sum” plays the role of multiplication. Indeed, the distributive law is easily verified:

$$\max(a + b, a + c) = a + \max(b, c), \quad (4.28)$$

as are the associative and commutative laws. Thus we can implement MAP-ELIMINATE algorithm by working with logarithms of potentials, and replacing “product” with “sum.”

There are many other commutative semirings, including semirings on polynomials and distributive lattices. We explore some of these commutative semirings in the exercises. The generic ELIMINATE algorithm can be easily adapted to each of these commutative semirings.

In Section ?? we showed that in the case of trees, the ELIMINATE algorithm can be equivalently expressed in terms of a coupled set of equations, or “messages,” a line of argument that led to the SUM-PRODUCT algorithm for inference on trees. The same arguments apply to arbitrary commutative semirings, and in particular we can obtain a “MAX-PRODUCT” version of the algorithm as follows:

$$m_{ji}^{\max}(x_i) = \max_{x_j} \left(\psi^E(x_j) \psi(x_i, x_j) \prod_{k \in \mathcal{N}(j) \setminus i} m_{kj}^{\max}(x_j) \right) \quad (4.29)$$

$$\max_x p^E(x) = \max_{x_i} \left(\psi^E(x_i) \prod_{j \in \mathcal{N}(i)} m_{ji}^{\max}(x_i) \right). \quad (4.30)$$

Implementing a depth-first traversal of the tree, thereby passing messages from the leaves toward an arbitrarily-defined root, we invoke Eq. (4.30) at the root and obtain the MAP solution.

Is there any value in considering a full message-passing algorithm in which we also send messages from the root back toward the leaves? If the problem is simply that of finding the maximal value of the MAP probability, $\max_x p^E(x)$, then the answer is no. Invoking Eq. (4.30) at multiple nodes in the graph, we obtain exactly the same solution—in all cases we have maximized over all nodes in the graph. However, if our goal is also that of obtaining a maximizing configuration—a configuration x^* such that $x^* \in \arg \max_x p^E(x)$ —then we can make use of an appropriately defined outward phase. We explore this issue in the following section.

4.3.1 Maximum a posteriori configurations

Let us now consider the problem of finding a configuration x^* such that $x^* \in \arg \max_x p^E(x)$. This problem can be solved by keeping track of the maximizing values of variables in the inward pass of the MAX-PRODUCT algorithm, and using these values as indices in an outward pass.

Throughout this section we assume that an arbitrary root node f has been chosen, and refer to an “inward pass” in which messages flow from the leaves toward the root, and an “outward pass” in which messages flow from the root toward the leaves.

Note that when the MAX-PRODUCT algorithm arrives at the root node at the end of the inward pass, the final maximization in Eq. (4.30) provides us with a value of the root node that belongs to a maximizing configuration. Thus, letting f denote the root, we compute:

$$x_f^* \in \arg \max_{x_f} \left(\psi^E(x_f) \prod_{e \in \mathcal{N}(f)} m_{ef}^{\max}(x_f) \right), \quad (4.31)$$

and thereby obtain a value x_f^* that necessarily belongs to a maximizing configuration. Moreover, in principle we could perform an outward pass in which we evaluate Eq. (4.29) for each node from the root to the leaves, and subsequently perform the maximization in Eq. (4.30) at each node. This would yield values x_i^* that belong to maximizing configurations. Unfortunately, however, there is no guarantee that these values all belong to the *same* maximizing configuration. To find a single maximizing configuration we have to work a bit harder.

Suppose that during the inward pass we maintain a record of the maximizing values of nodes when we compute the messages $m_{ji}^{\max}(x_i)$. That is, whenever we send a message $m_{ji}^{\max}(x_i)$ from node j to its parent node i , we also record the maximizing values in a table $\delta_{ji}(x_i)$:

$$\delta_{ji}(x_i) \in \arg \max_{x_j} \left(\psi^E(x_j) \psi(x_i, x_j) \prod_{k \in \mathcal{N}(j) \setminus i} m_{kj}^{\max}(x_j) \right). \quad (4.32)$$

Thus, for each x_i , the function $\delta_{ji}(x_i)$ picks out a value of x_j (there may be several) that achieves the maximum.

Having defined the function $\delta_{ji}(x_i)$ during the inward pass, we use $\delta_{ji}(x_i)$ to define a consistent maximizing configuration during an outward pass. Thus, starting at the root f , we choose a maximizing value x_f^* . Given this value, which we pass to the children of f , we set $x_e^* = \delta_{ef}(x_f^*)$ for each $e \in \mathcal{N}(f)$. This procedure continues outward to the leaves.

The resulting algorithm is summarized in Figure 4.19. Note that the computation of the $m_{ji}^{\max}(x_i)$ messages in the inward pass of this algorithm is identical to the MAP-ELIMINATE algorithm (for undirected trees).

4.4 Conclusions

In this chapter we have presented a basic treatment of algorithms for computing probabilities on graphs. Restricting ourselves to trees, we presented the SUM-PRODUCT algorithm, an algorithm for computing all singleton marginal probabilities. We also presented a SUM-PRODUCT algorithm for factor trees, and showed how this algorithm allows us to compute marginal probabilities for various tree-like graphs, including polytrees. Finally, we showed that the algebra underlying the SUM-PRODUCT algorithm can be abstracted, yielding a general family of propagation algorithms based on commutative semirings. In particular, we presented the MAX-PRODUCT algorithm, an algorithm for computing maximum a posteriori probabilities.

Henceforth we will refer to all such propagation algorithms as *probability propagation algorithms*. While we have restricted ourselves to trees in the current chapter, we will be considering probability propagation algorithms on more general graphs in later chapters.

Thus far we have focused on the problems of representation and inference in graphical models. We return to these problems in Chapters 16 and 17, providing a more general and more formal treatment of topics such as conditional independence and probability propagation. In the intervening chapters, however, we shift to a different line of inquiry. In particular, we now begin to address the problem of interfacing graphical models to data, and we begin to develop methods for evaluating and improving models on the basis of such data. We thus take up the statistical side of the story.

4.5 Historical remarks and bibliography

```

MAX-PRODUCT( $\mathcal{T}$ ,  $E$ )
  EVIDENCE( $E$ )
   $f = \text{CHOSEROOT}(\mathcal{V})$ 
  for  $e \in \mathcal{N}(f)$ 
    COLLECT( $f, e$ )
     $MAP = \max_{x_f} (\psi^E(x_f) \prod_{e \in \mathcal{N}(f)} m_{ef}^{\max}(x_f))$ 
     $x_f^* = \arg \max_{x_f} (\psi^E(x_f) \prod_{e \in \mathcal{N}(f)} m_{ef}^{\max}(x_f))$ 
    for  $e \in \mathcal{N}(f)$ 
      DISTRIBUTE( $f, e$ )

COLLECT( $i, j$ )
  for  $k \in \mathcal{N}(j) \setminus i$ 
    COLLECT( $j, k$ )
    SENDMESSAGE( $j, i$ )

DISTRIBUTE( $i, j$ )
  SETVALUE( $i, j$ )
  for  $k \in \mathcal{N}(j) \setminus i$ 
    DISTRIBUTE( $j, k$ )

SENDMESSAGE( $j, i$ )
   $m_{ji}^{\max}(x_i) = \max_{x_j} (\psi^E(x_j) \psi(x_i, x_j) \prod_{k \in \mathcal{N}(j) \setminus i} m_{kj}^{\max}(x_j))$ 
   $\delta_{ji}(x_i) \in \arg \max_{x_j} (\psi^E(x_j) \psi(x_i, x_j) \prod_{k \in \mathcal{N}(j) \setminus i} m_{kj}^{\max}(x_j))$ 

SETVALUE( $i, j$ )
   $x_j^* = \delta_{ji}(x_i^*)$ 

```

Figure 4.19: A sequential implementation of the MAX-PRODUCT algorithm for a tree $\mathcal{T}(\mathcal{V}, \mathcal{E})$. The algorithm works for any choice of root node, and thus we have left CHOSEROOT unspecified. The subroutine EVIDENCE(E) is presented in Figure 4.5.

Chapter 5

Statistical Concepts

It is useful to attempt to distinguish the activities of the probability theorist and the statistician. Our perspective in the previous chapters has been mainly that of the former—we have built graphical models involving sets of random variables and shown how to compute the probabilities of certain events associated with these random variables. Given a particular choice of graphical model, consisting of a graph and a set of local conditional probabilities or potentials, we have seen how to infer the probabilities of various events of interest, such as the marginal or conditional probability that a particular random variable takes on a particular value.

Statistics is in a certain sense the inverse of probability theory. In a statistical setting the random variables in our domain have been *observed* and are therefore no longer unknown, rather it is the model that is unknown. We wish to infer the model from the data rather than the data from the model.

The problem of “inferring the model from the data” is a deep one, raising fundamental questions regarding the nature of knowledge, reasoning, learning, and scientific inquiry. In statistics, the study of these fundamental questions has often come down to a distinction between two major schools of thought—the *Bayesian* and the *frequentist*. In the following section we briefly outline the key distinctions between these two schools. It is worth noting that our discussion here will be incomplete and that we will be returning to these distinctions at various junctures in the book as our development of graphical models begins to bring the distinctions into clearer relief. But an equally important point to make is that many of the problems—particularly the computational problems—faced in these frameworks are closely related, even identical. A great deal of important work can be done within the graphical models formalism that is equally useful to Bayesian and frequentist statistics.

Beyond our discussion of foundational issues, we will also introduce several classes of statistical problems in this chapter, in particular the core problems of *density estimation*, *regression* and *classification*. As in earlier chapters our goal is to present enough in the way of concrete details to make the discussion understandable, but to emphasize broad themes that will serve as landmarks for our more detailed presentation in later chapters.

5.1 Bayesian and frequentist statistics

Bayesian statistics is in essence an attempt to deny any fundamental distinction between probability theory and statistics. Probability theory itself provides the capability for inverting relationships between uncertain quantities—this is the essence of *Bayes rule*—and Bayesian statistics represents an attempt to treat all statistical inference as probabilistic inference.

Let us consider a problem in which we have already decided upon the model *structure* for a given problem domain—for example, we have chosen a particular graphical model including a particular pattern of connectivity—but we have not yet chosen the values of the model *parameters*—the numerical values of the local conditional probabilities or potentials. We wish to choose these parameter values on the basis of observed data. (In general we might also want to choose the model structure on the basis of observed data, but let us postpone that problem—see Section 5.3).

For every choice of parameter values we obtain a different numerical specification for the joint distribution of the random variables X . We will henceforth write this probability distribution as $p(x|\theta)$ to reflect this dependence. Putting on our hats as probability theorists, we view the model $p(x|\theta)$ as a conditional probability distribution; intuitively it is an assignment of probability mass to unknown values of X , given a fixed value of θ . Thus, θ is known and X is unknown. As statisticians, however, we view X as known—we have observed its realization x —and θ as unknown. We thus in some sense need to invert the relationship between x and θ . The Bayesian point of view implements this notion of “inversion” using Bayes rule:

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)}. \quad (5.1)$$

The assumptions allowing us to write this equation are noteworthy. First, in order to interpret the left-hand side of the equation we must view θ as a random variable. This is characteristic of the Bayesian approach—all unknown quantities are treated as random variables. Second, we view the data x as a quantity to be conditioned on—our inference is conditional on the event $\{X = x\}$. Third, in order to calculate $p(\theta|x)$ we see (from the right-hand side of Eq. (5.1)) that we must have in hand the probability distribution $p(\theta)$ —the *prior probability* of the parameters. Given that we are viewing θ as a random variable, it is formally reasonable to assign a (marginal) probability to it, but one needs to think about what such a prior probability means in terms of the problem we are studying. Finally, note that Bayes rule yields a distribution over θ —the *posterior probability* of θ given x , not a single estimate of θ . If we wish to obtain a single value, we must (and will) invoke additional principles, but it is worth noting at the outset that the Bayesian approach tends to resist collapsing distributions to points.

The frequentist approach wishes to avoid the use of prior probabilities in statistics, and thus avoids the use of Bayes rule for the purpose of assigning probabilities to parameters. The goal of frequentist methodology is to develop an “objective” statistical theory, in which two statisticians employing the methodology must necessarily draw the same conclusions from a particular set of data.

Consider in particular a coin-tossing experiment, where $X \in \{0, 1\}$ is a binary variable representing the outcome of the coin toss, and $\theta \in (0, 1)$ is a real-valued parameter denoting the probability of heads. Thus the model is the Bernoulli distribution, $p(x|\theta) = \theta^x(1-\theta)^{1-x}$. Approaching the

problem from a Bayesian perspective requires us to assign a prior probability to θ before observing the outcome of the coin toss. Two different Bayesian statisticians may assign different priors to θ and thus obtain different conclusions from the experiment. The frequentist statistician wishes to avoid such “subjectivity.” From another point of view, a frequentist may claim that θ is a fixed property of the coin, and that it makes no sense to assign probability to it. A Bayesian may agree with the former statement, but would argue that $p(\theta)$ need not represent anything about the physics of the situation, but rather represents the *statistician’s uncertainty* about the value of θ . Tossing the coin reduces the statistician’s uncertainty, and changes the prior probability into the posterior probability $p(\theta | x)$. Bayesian statistics views the posterior probability and the prior probability alike as (possibly) subjective.

There are situations in which frequentist statistics and Bayesian statistics agree that parameters can be endowed with probability distributions. Suppose that we consider a factory that makes coins in batches, where each batch is characterized by a smelting process that affects the fairness of the resulting coins. A coin from a given batch has a different probability of heads than a coin from a different batch, and ranging over batches we obtain a distribution on the probability of heads θ . A frequentist is in general happy to assign prior probabilities to parameters, as long as those probabilities refer to objective frequencies of observing values of the parameters in repeated experiments.

From the point of view of frequentist statistics, there is no single preferred methodology for inverting the relationship between parameters and data. Rather, the basic idea is to consider various *estimators* of θ , where an estimator is some function of the observed data x (we will discuss a particular example below). One establishes various general criteria for evaluating the quality of various estimators, and chooses the estimator that is “best” according to these criteria. (Examples of such criteria include the *bias* and *variance* of estimators; these criteria will be discussed in Chapter 26). An important feature of this evaluation process is that it generally requires that the data x be viewed as the result of a random experiment that can be repeated and in which other possible values of x could have been obtained. This is of course consistent with the general frequentist philosophy, in which probabilities correspond to objective frequencies.

There is one particular estimator that is widely used in frequentist statistics, namely the *maximum likelihood* estimator. This estimator is popular for a number of reasons, in particular because it often yields “natural estimators” (e.g., sample proportions and sample means) in simple settings and also because of its favorable asymptotic properties.

To understand the maximum likelihood estimator, we must understand the notion of “likelihood” from which it derives. Recall that the probability model $p(x | \theta)$ has the intuitive interpretation of assigning probability to X for each fixed value of θ . In the Bayesian approach this intuition is formalized by treating $p(x | \theta)$ as a conditional probability distribution. In the frequentist approach, however, such a formal interpretation is suspect, because it suggests that θ is a random variable that can be conditioned on. The frequentist instead treats the model $p(x | \theta)$ as a family of probability distributions indexed by θ , with no implication that we are conditioning on θ .¹ Moreover, to implement a notion of “inversion” between x and θ , we simply change our point

¹To acknowledge this interpretation, frequentist treatments often adopt the notation $p_\theta(x)$ in place of $p(x | \theta)$. We will stick with $p(x | \theta)$, hoping that the frequentist-minded reader will forgive us this abuse of notation. It will

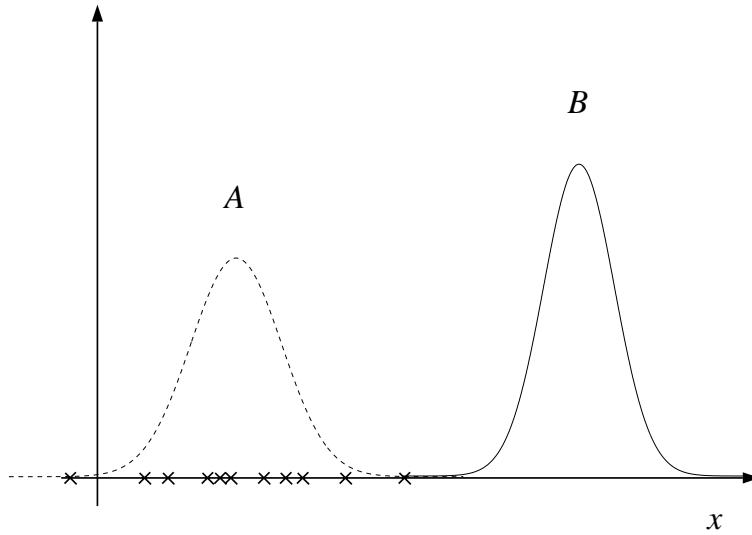


Figure 5.1: A univariate density estimation problem. (See Section 5.2.1 for a discussion of density estimation). The data $\{x_1, x_2, \dots, x_N\}$ are given as X's along the abscissa. The parameter vector θ is the mean μ and variance σ^2 of a Gaussian density. Two candidate densities, involving different values of θ , are shown in the figure. Density A assigns higher probability to the observed data than density B, and thus would be preferred according to the principle of maximum likelihood.

of view—we treat $p(x | \theta)$ as a function of θ for fixed x . When interpreted in this way, $p(x | \theta)$ is referred to as the *likelihood function* and it provides the basis for maximum likelihood estimation.

As suggested in Figure 5.1, the likelihood function can be used to evaluate particular choices of θ . In particular, if for a given value of θ we find that the observed value of x is assigned low probability, then this is perhaps a poor choice of θ . A value of θ that assigns higher probability to x is preferred. Ranging over all possible choices of θ , we pick that value of θ that assigns maximal probability to x , and treat this value as an estimate of the true θ :

$$\hat{\theta}_{ML} = \operatorname{argmax}_\theta p(x | \theta). \quad (5.2)$$

Thus the maximum likelihood estimate is that value of θ that maximizes the likelihood function.

Regardless of whether one agrees that this justification of the maximum likelihood estimate is a natural one, it is certainly true that we have an estimator—a function of x —and we can evaluate the properties of this estimator under various frequentist criteria. It turns out that maximum likelihood is a good estimator under a variety of measures of quality, particularly in settings of large sample sizes when asymptotic analyses are meaningful (indeed, maximum likelihood estimates can be shown to be “optimal” in such settings). In other settings, particularly in cases of small sample sizes, maximum likelihood plays an important role as the starting point for the development of more complex estimators.

simplify our presentation throughout the rest of the book, liberating us from having to make distinctions between Bayesian and frequentist interpretations where none are needed or implied.

Another appealing feature of likelihood-based estimation is that it provides a link between Bayesian methods and frequentist methods. In particular, note that the distribution $p(x|\theta)$ appears in our basic Bayesian equation Eq. (5.1). Note moreover that Bayesian statisticians refer to this probability as a “likelihood” as do frequentist statisticians, even though the interpretation is different. Symbolically, we can interpret Eq. (5.1) as follows:

$$\text{posterior} \propto \text{likelihood} \times \text{prior}, \quad (5.3)$$

where we see that in the Bayesian approach the likelihood can be viewed as a data-dependent operator that transforms between the prior probability and the posterior probability. At a bare minimum, Bayesian approaches and likelihood-based frequentist approaches have in common the need to calculate the likelihood for various values of θ . This is not a trivial fact—indeed a major focus of this book is the set of complex statistical models in which the computation of the likelihood is itself a daunting computational task. In working out effective computational procedures to deal with such models we are contributing to both Bayesian and frequentist statistics.

Let us explore this connection between Bayesian and frequentist approaches a bit further. Suppose in particular that we force the Bayesian to choose a particular value of θ ; that is, to collapse the posterior distribution $p(\theta|x)$ to a point estimate. Various possibilities present themselves; in particular one could choose the mean of the posterior distribution or perhaps the mode. The mean of the posterior is often referred to as a *Bayes estimate*:

$$\hat{\theta}_{\text{Bayes}} = \int \theta p(\theta|x)d\theta, \quad (5.4)$$

and it is possible and worthwhile to study the frequentist properties of Bayes estimates. The mode of the posterior is often referred to as the *maximum a posteriori (MAP)* estimate:

$$\hat{\theta}_{\text{MAP}} = \operatorname{argmax}_{\theta} p(\theta|x) \quad (5.5)$$

$$= \operatorname{argmax}_{\theta} p(x|\theta)p(\theta), \quad (5.6)$$

where in the second equation we have utilized the fact that the factor $p(x)$ in the denominator of Bayes rule is independent of θ . In a setting in which the prior probability is taken to be uniform on θ , the MAP estimate reduces to the maximum likelihood estimate. When the prior is not taken to be uniform, one can still view Eq. (5.6) as the maximization of a *penalized likelihood*. To see this, note that one generally works with logarithms when maximizing over probability distributions (the fact that the logarithm is a monotonic function implies that it does not alter the optimizing value). Thus one has:

$$\hat{\theta}_{\text{MAP}} = \operatorname{argmax}_{\theta} \{ \log p(x|\theta) + \log p(\theta) \}, \quad (5.7)$$

as an alternative expression for the MAP estimate. Here the “penalty” is the additive term $\log p(\theta)$. Penalized log likelihoods are widely used in frequentist statistics to improve on maximum likelihood estimates in small sample settings (as we will see in Chapter 26).

It is important to emphasize, however, that MAP estimation involves a rather un-Bayesian use of the Bayesian formalism, and it would be wrong to understand the distinction between Bayesian and frequentist statistics as merely a matter of how to interpret a penalized log likelihood. To clarify,

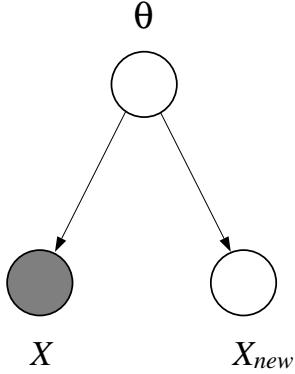


Figure 5.2: A graphical representation of the problem of prediction from a Bayesian point of view.

let us consider a somewhat broader problem in which the difference between MAP estimation and a fuller Bayesian approach is more salient. Let us consider the problem of *prediction*, where we are not interested in the value of θ per se, but are interested in using a model based on θ to predict future values of the random variable X . Let us suppose in particular that we have two random variables, X and X_{new} , which are characterized by the same distribution, and that we wish to use an observation of X to make a prediction regarding likely values of X_{new} . For simplicity, let us assume that X and X_{new} are independent; more precisely, we assume that they are conditionally independent given θ . We write:

$$p(x_{new} | x) = \int p(x_{new}, \theta | x) d\theta \quad (5.8)$$

$$= \int p(x_{new} | \theta, x) p(\theta | x) d\theta \quad (5.9)$$

$$= \int p(x_{new} | \theta) p(\theta | x) d\theta. \quad (5.10)$$

From the latter equation we see that the Bayesian prediction is based on combining the predictions across all values of θ , with the posterior distribution serving as a “weighting function.” That is, interpreting the conditional probability $p(x_{new} | \theta)$ as the prediction of X_{new} given θ , we weight this prediction by the posterior probability $p(\theta | x)$, and integrate over all such weighted predictions. Note in particular that this calculation requires the entire posterior probability, not merely its value at a single point.

Within a frequentist approach, we are not allowed to treat θ as a random variable, and thus we do not attribute meaning to the integral in Eq. (5.10). Rather, we would consider various “estimates” of x_{new} ; a natural choice might be the “plug-in estimate” $p(x_{new} | \hat{\theta}_{ML})$. Here we see that the difference between the frequentist approach and the Bayesian approach has become more significant; in the latter case we have to perform an integral in order to obtain a prediction. We can relate the two approaches if we approximate the posterior distribution by collapsing it to a delta function at $\hat{\theta}_{MAP}$, in which case the integral in Eq. (5.10) reduces to the plug-in estimate

$p(x_{new} | \hat{\theta}_{MAP})$. But in general this collapse would not satisfy the Bayesian (who views the integral as providing a better predictor than any predictor based on a point estimate) nor the frequentist (who wants to be free to consider a wider class of estimates than the plug-in estimate).

As a final note, consider the graphical model shown in Figure 5.2. This model captures the Bayesian point of view on the prediction problem that we have just discussed. The parameter θ is depicted as a node in the model; this is of course consistent with the Bayesian approach of treating parameters as random variables. Moreover, the conditional independence of X and X_{new} given θ is reflected as a Markov property in the graph. Finally, as we invite the reader to verify in Exercise ??, applying the elimination algorithm to the graph yields exactly the calculation in Eq. (5.10). This is a reflection of a general fact—graphical models provide a nice way to visualize and organize Bayesian calculations. We will return to this point in later chapters. But let us emphasize here that this linkage, appealing as it is, does not reflect any special affinity between graphical models and Bayesian methods, but rather is a reflection of the more general link between Bayesian methods and probabilistic inference.

5.2 Statistical problems

Let us now descend from the somewhat ethereal considerations of statistical foundations to a rather more concrete consideration of problems in statistical estimation. In this section we will discuss three major classes of statistical problems—*density estimation*, *regression*, and *classification*. Not all statistical problems fall into one of these three classes, nor is it always possible to unambiguously characterize a given problem in terms of these classes, but there are certain core aspects of these three problem categories that are worth isolating and studying in a purified form.

We have two main goals in this section. The first is to introduce the graphical approach to representing statistical modeling problems, in particular emphasizing how the graphical representation helps makes modeling assumptions explicit. Second, we wish to begin to work with specific probability distributions, in particular the Gaussian and multinomial distributions. We will use this introductory section to illustrate some of the calculations that arise when using these distributions.

5.2.1 Density estimation

Suppose that we have in hand a set of observations on a random variable X —in general a vector-valued random variable—and we wish to use these observations to induce a probability density (probability mass function for discrete variables) for X . This problem—which we refer to generically as the problem of density estimation—is a very general statistical problem. Obtaining a model of the density of X allows us to assess whether a particular observation of X is “typical,” an assessment that is required in many practical problems including *fault detection*, *outlier detection* and *clustering*. Density estimation also underlies many *dimensionality reduction* algorithms, where a joint density is projected onto a subspace or manifold, hopefully reducing the dimensionality of a data set while retaining its salient features. A related application is *compression*, where Shannon’s fundamental relationship between code length and the negative logarithm of the density can be used to design a source code. Finally, noting that a joint density on X can be used to infer conditional

densities among components of X , we can also use density estimates to solve problems in prediction.

To delimit the scope of the problem somewhat, note that in regression and classification the focus is on the relationship between a pair of variables, X and Y . That is, regression and classification problems differ from density estimation in that their focus is on a conditional density, $p(y | x)$, with the marginal $p(x)$ and the corresponding joint density of less interest, and perhaps not modeled at all. We develop methods that are specific to conditional densities in Sections 5.2.2 and 5.2.3.

Density estimation arises in many ways in the setting of graphical models. In particular we may be interested in inferring the density of a parentless node in a directed graphical model, or the density of a set of nodes in a larger model (in which case the density of interest is a marginal density), or the joint density of all of the nodes of our model.

Let us begin with an example. Our example will be one of the most classical of all statistical problems—that of estimating the mean and variance of a univariate Gaussian distribution.

Univariate Gaussian density estimation

Let us assume that X is a univariate random variable with a Gaussian distribution, that is:

$$p(x | \theta) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x - \mu)^2\right\}, \quad (5.11)$$

where μ and σ^2 are the mean and variance, respectively, and $\theta \triangleq (\mu, \sigma^2)$.² We wish to estimate θ based on observations of X . Here we are assuming that we know the parametric form of the density of X , and what is unknown are the numerical values of the parameters (cf. Figure 5.1). Plugging estimates of the parameters back into Eq. (5.11) provides an estimate of the density function.

Clearly a single observation of X provides no information about the variance and relatively poor information about the mean. Thus we need to consider multiple observations. What do we mean by “multiple observations”? Let us interpret this to mean that we have a *set of random variables*, $\{X_1, X_2, \dots, X_N\}$, and that these random variables are *identically distributed*. Thus each of the variables X_n is characterized by a Gaussian distribution $p(x_n | \theta)$, with the same θ for each X_n .

In graphical model terms, we have a model with N nodes, one for each random variable. Which graphical model should we use? What connectivity pattern should we use? Let us suppose that the variables are not only identically distributed but that they are also *independent*. Thus we have the graphical model shown in Figure 5.3. It should be emphasized that these assumptions are by no means necessary; they are simply one possible set of assumptions, corresponding to a particular choice of graphical model. (We will be seeing significantly more complex graphical models on N Gaussian nodes; see, e.g., the Kalman filter in Chapter 15).

The nodes in Figure 5.3 are shaded, reflecting the fact that they are *observed data*. In general, “data” are designated by the shading of nodes in our models. In the context of the Bayesian approach to estimation, this use of shading is the same convention as we used in Chapter 2—in the Bayesian approach we *condition on the data* in order to compute probabilities for the parameters. In the context of frequentist approaches, where we no longer view ourselves as conditioning on the

²We will often denote this density as $\mathcal{N}(\mu, \sigma^2)$.

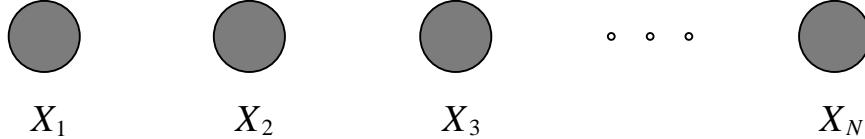


Figure 5.3: A graphical model representing the density estimation problem under an IID sampling model. The assumption that the data are sampled independently is reflected by the absence of links between the nodes. Each node is characterized by the same density.

data, we simply treat shading as a diagrammatic convention to indicate which nodes correspond to the observed data.

Letting X refer to the set of random variables (X_1, X_2, \dots, X_N) , and letting x refer to the observations (x_1, x_2, \dots, x_N) , we write the joint probability $p(x | \theta)$ as the product of local probabilities, one for each node in Figure 5.3:

$$p(x | \theta) = \prod_{n=1}^N \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x_n - \mu)^2\right\} \quad (5.12)$$

$$= \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left\{-\frac{1}{2\sigma^2} \sum_{n=1}^N (x_n - \mu)^2\right\}, \quad (5.13)$$

or alternatively, given that this particular graph can be interpreted as either a directed graph or an undirected graph, we can view this joint probability as a product of potential functions on the cliques of the graph (which are singleton nodes in this case).

Let us proceed to calculating parameter estimates. In particular let us calculate the maximum likelihood estimates of μ and σ^2 . To do so we must maximize the likelihood $p(x | \theta)$ with respect to θ . We find it more convenient to maximize the logarithm of the likelihood, which, given that the logarithm is a monotonic function, will not change the results. Thus, let us define the *log likelihood*, denoted $l(\theta; x)$, as:

$$l(\theta; x) = \log p(x | \theta), \quad (5.14)$$

where we have reordered the variables on the left-hand side to emphasize that θ is to be viewed as the variable and x is to be viewed as a fixed constant. We now take the derivative of the log likelihood with respect to μ :

$$\frac{\partial l(\theta; x)}{\partial \mu} = \frac{\partial}{\partial \mu} \left(-\frac{N}{2} \log(2\pi) - \frac{N}{2} \log \sigma^2 - \frac{1}{2\sigma^2} \sum_{n=1}^N (x_n - \mu)^2 \right) \quad (5.15)$$

$$= \frac{1}{\sigma^2} \sum_{n=1}^N (x_n - \mu). \quad (5.16)$$

Setting equal to zero and solving, we obtain:

$$\hat{\mu}_{ML} = \frac{1}{N} \sum_{n=1}^N x_n. \quad (5.17)$$

Thus we see that the maximum likelihood estimate of the mean of a Gaussian distribution is the sample mean.

Similarly let us take the derivative of the log likelihood with respect to σ^2 :

$$\frac{\partial l(\theta; x)}{\partial \sigma^2} = \frac{\partial}{\partial \sigma^2} \left(-\frac{N}{2} \log(2\pi) - \frac{N}{2} \log \sigma^2 - \frac{1}{2\sigma^2} \sum_{n=1}^N (x_n - \mu)^2 \right) \quad (5.18)$$

$$= -\frac{N}{2\sigma^2} + \frac{1}{2\sigma^4} \sum_{n=1}^N (x_n - \mu)^2. \quad (5.19)$$

Setting equal to zero and solving, we obtain:

$$\hat{\sigma}_{ML}^2 = \frac{1}{N} \sum_{n=1}^N (x_n - \hat{\mu}_{ML})^2, \quad (5.20)$$

and we see that the maximum likelihood estimate of the variance is the sample variance. (Note that we are finding the joint estimates of μ and σ^2 by setting both partial derivatives equal to zero and solving simultaneously; this explains the presence of $\hat{\mu}_{ML}$ in the equation for $\hat{\sigma}_{ML}^2$).

Bayesian univariate Gaussian density estimation

In the Bayesian approach to density estimation the goal is to form a posterior density $p(\theta | x)$. Let us consider a simple version of this problem in which we take the variance σ^2 to be a known constant and restrict our attention to the mean μ . Thus we wish to obtain the posterior density $p(\mu | x)$, based on the prior density $p(\mu)$ and the Gaussian likelihood $p(x | \mu)$.

What prior distribution should we take for μ ? This is a modeling decision, as was the decision to utilize a Gaussian for the probability of the data x in the first place. As we will see, it is mathematically convenient to take $p(\mu)$ to also be a Gaussian distribution. We will make this assumption in this section, but let us emphasize at the outset that mathematical convenience should not, and need not, dictate all of our modeling decisions. Indeed, a major thrust of this book is the development of methods for treating complex models, pushing back the frontier of what is “mathematically convenient” and, in the Bayesian setting, permitting a wide and expressive range of prior distributions.

If we take $p(\mu)$ to be a Gaussian distribution, then we face another problem: what should we take as the mean and variance of this distribution? To be consistent with the general Bayesian philosophy, we should treat these parameters as random variables and endow them with a prior distribution. This is indeed the approach of *hierarchical Bayesian modeling*, where we endow parameters with distributions characterized by “hyperparameters,” which themselves can in turn

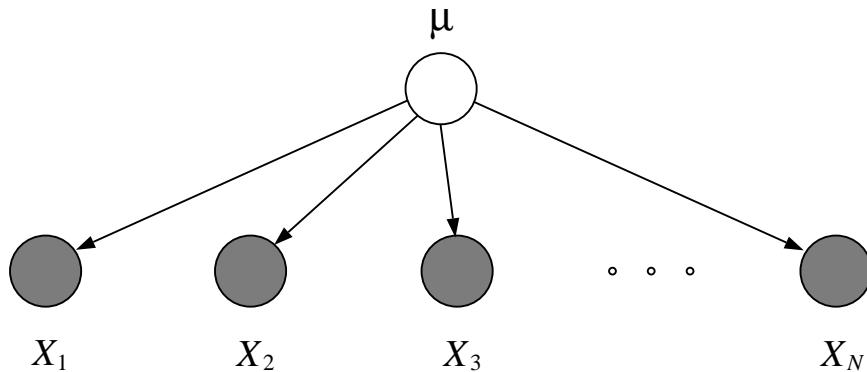


Figure 5.4: The graphical model for the Bayesian density estimation problem.

be endowed with distributions. While an infinite regress looms, in practice it is rare to take the hierarchical Bayesian approach to more than two or three levels, largely because of diminishing returns—additional levels make little difference to the marginal probability of the data and thus to the expressiveness of our model.

Let us take the mean of $p(\mu)$ to be a fixed constant μ_0 and take the variance to be a fixed constant τ^2 , while recognizing that in general we might endow these parameters with distributions.

The graphical model characterizing our problem is shown in Figure 5.4. The graph has been augmented with a node for the unknown mean μ . Note that there is a single such node and that its children are the data $\{X_n\}$. Thus this graph provides more information than the graph of Figure 5.3; in particular the independence assumption is elaborated—the data are assumed to be *conditionally independent given the parameters*.

The likelihood is identical in form to the frequentist likelihood in Eq. (5.13). To obtain the posterior we therefore need only multiply by the prior:

$$p(\mu) = \frac{1}{(2\pi\tau^2)^{1/2}} \exp \left\{ -\frac{1}{2\tau^2} (\mu - \mu_0)^2 \right\} \quad (5.21)$$

to obtain the joint probability:

$$p(x, \mu) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp \left\{ -\frac{1}{2\sigma^2} \sum_{n=1}^N (x_n - \mu)^2 \right\} \frac{1}{(2\pi\tau^2)^{1/2}} \exp \left\{ -\frac{1}{2\tau^2} (\mu - \mu_0)^2 \right\}, \quad (5.22)$$

which when normalized yields the posterior $p(\mu | x)$. Multiplying the two exponentials together yields an exponent which is quadratic in the variable μ ; thus, normalization involves “completing the square.” Appendix A presents the algebra (and in Chapter 13 we present a general matrix-based approach to completing the square—an operation that crops up often when working with Gaussian random variables). The result takes the following form:

$$p(\mu \mid x) = \frac{1}{(2\pi\tilde{\sigma}^2)^{1/2}} \exp \left\{ -\frac{1}{2\tilde{\sigma}^2} (\mu - \tilde{\mu})^2 \right\}, \quad (5.23)$$

where

$$\tilde{\mu} = \frac{N/\sigma^2}{N/\sigma^2 + 1/\tau^2} \bar{x} + \frac{1/\tau^2}{N/\sigma^2 + 1/\tau^2} \mu_0, \quad (5.24)$$

where \bar{x} is the sample mean, and where

$$\tilde{\sigma}^2 = \left(\frac{N}{\sigma^2} + \frac{1}{\tau^2} \right)^{-1}. \quad (5.25)$$

We see that the posterior probability is a Gaussian, with mean $\tilde{\mu}$ and variance $\tilde{\sigma}^2$.

Both the posterior variance and the posterior mean have an intuitive interpretation. Note first that σ^2/N is the variance of a sum of N independent random variables with variance σ^2 , thus σ^2/N is the variance associated with the data. Eq. (5.25) says that we add the inverse of this variance to the inverse of the prior variance to obtain the inverse of the posterior variance. Thus, inverse variances add. From Eq. (5.24) we see that the posterior mean is obtained as a linear combination of the sample mean and the prior mean. The weights in this combination can be interpreted as the fraction of the posterior variance accounted for by the variance from the data term and the prior variance respectively. These weights sum to one; thus, the combination in Eq. (5.24) is a *convex combination*.

As the number of data points N becomes large, the weight associated with \bar{x} goes to one and the weight associated with μ_0 approaches zero. Thus in the limit of large data sets, the Bayes estimate of μ approaches the maximum likelihood estimate of μ .

Plates

Let us take a quick detour to discuss a notational device that we will find useful. Graphical models representing independent, identically distributed (IID) sampling have a repetitive structure that can be captured with a formal device known as a *plate*. Plates allow repeated motifs to be represented in a simple way. In particular, the simple IID model shown in Figure 5.5(a) can be represented more succinctly using the plate shown in Figure 5.5(b).

For the Bayesian model in Figure 5.6(a) we obtain the representation in Figure 5.6(b). Note that the parameter μ appears *outside* the plate; this captures the fact that there is a single parameter value that is shared among the distributions for each of the X_n .

Formally, a plate is simply a graphical model “macro.” That is, to interpret Figure 5.5(b) or Figure 5.6(b) we copy the graphical object in the plate N times, where the number N is recorded in the lower right-hand corner of the box, and apply the usual graphical model semantics to the result.

Density estimation for discrete data

Let us now consider the case in which the variables X_n are discrete variables, each taking on one of a finite number of possible values. We wish to study the density estimation problem in this setting, recalling that “probability density” means “probability mass function” in the discrete case.

As before, we will make the assumption that the data are IID, thus the modeling problem is represented by the plate shown in Figure 5.5(b). Each of the variables X_n can take on one of M

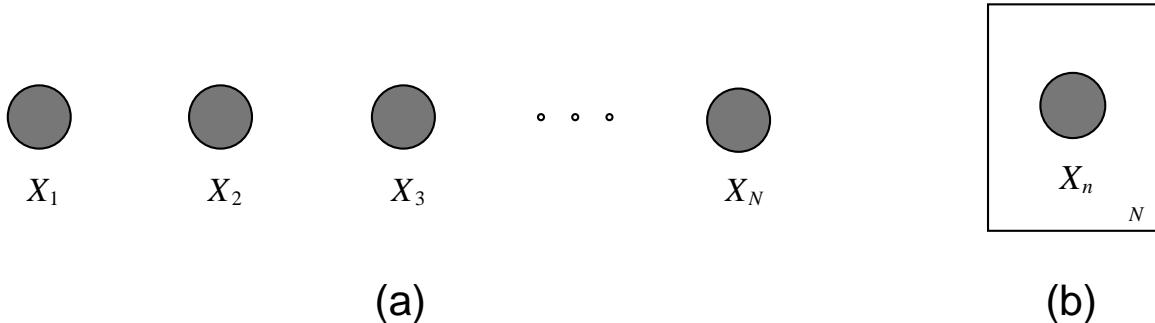


Figure 5.5: Repeated graphical motifs can be represented using plates. The IID sampling model for density estimation shown in (a) is represented using a plate in (b). The plate is interpreted by copying the graphical object within the box N times; thus the graph in (b) is a shorthand for the graph in (a).

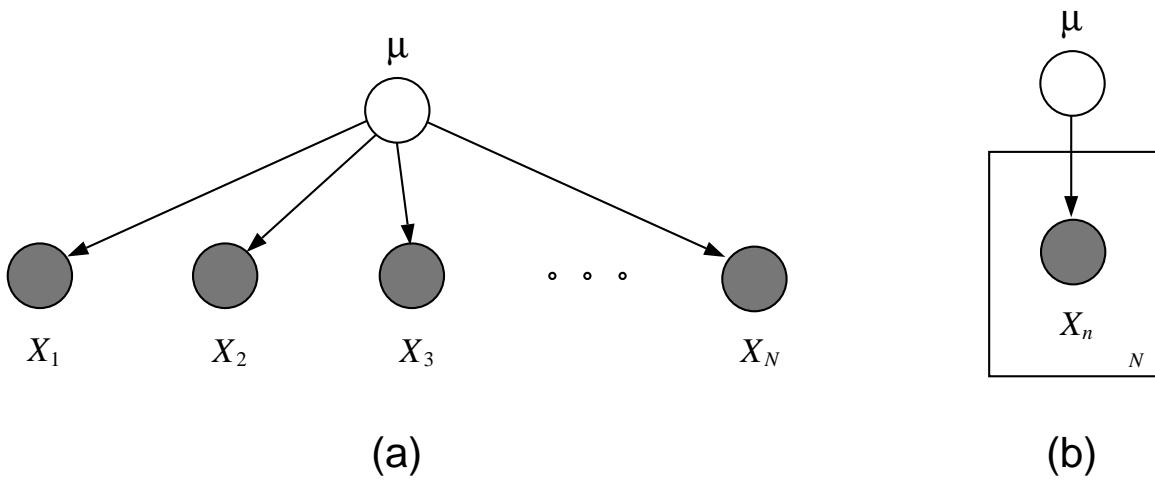


Figure 5.6: The Bayesian density estimation model shown in (a) is represented using a plate in (b). Again, the graph in (b) is to be interpreted as a shorthand for the graph in (a).

values. To represent this set of M values we will find it convenient to use a vector representation. In particular, let the range of X_n be the set of binary M -component vectors with one component equal to one and the other components equal to zero. Thus for a variable X_n taking on three values, we have:

$$X_n \in \left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right\}. \quad (5.26)$$

We use superscripts to refer to the components of these vectors, thus X_n^k refers to the k th component of the variable X_n . We have $X_n^k = 1$ if and only if the variable X_n takes on its k th value. Note that $\sum_k X_n^k = 1$ by definition.

Using this representation, we can write the probability distribution for X_n in a convenient general form. In particular, letting θ_k represent the probability that X_n takes on its k th value, i.e., $\theta_k \triangleq p(X_n^k = 1)$, we have:

$$p(x_n | \theta) = \theta_1^{x_n^1} \theta_2^{x_n^2} \cdots \theta_M^{x_n^M}. \quad (5.27)$$

This is the *multinomial* probability distribution, $\text{Mult}(1, \theta)$, with parameter vector $\theta = (\theta_1, \theta_2, \dots, \theta_M)$. To calculate the probability of the observation x , we take the product over the individual multinomial probabilities:

$$p(x | \theta) = \prod_{n=1}^N \theta_1^{x_n^1} \theta_2^{x_n^2} \cdots \theta_M^{x_n^M} \quad (5.28)$$

$$= \theta_1^{\sum_{n=1}^N x_n^1} \theta_2^{\sum_{n=1}^N x_n^2} \cdots \theta_M^{\sum_{n=1}^N x_n^M}, \quad (5.29)$$

where the exponent $\sum_{n=1}^N x_n^k$ is the count of the number of times the k th value of the multinomial variable is observed across the N observations.

To calculate the maximum likelihood estimates of the multinomial parameters we take the logarithm of Eq. (5.29) to obtain the log likelihood:

$$l(\theta; x) = \sum_{n=1}^N \sum_{k=1}^M x_n^k \log \theta_k, \quad (5.30)$$

and it is this expression that we must maximize with respect to θ .

This is a constrained optimization problem for which we use Lagrange multipliers. Thus we form the Lagrangian:

$$\tilde{l}(\theta; x) = \sum_{n=1}^N \sum_{k=1}^M x_n^k \log \theta_k + \lambda \left(1 - \sum_{k=1}^M \theta_k \right), \quad (5.31)$$

take derivatives with respect to θ_k :

$$\frac{\partial \tilde{l}(\theta; x)}{\partial \theta_k} = \frac{\sum_{n=1}^N x_n^k}{\theta_k} - \lambda \quad (5.32)$$

and set equal to zero:

$$\frac{\sum_{n=1}^N x_n^k}{\hat{\theta}_{k,ML}} = \lambda. \quad (5.33)$$

Multiplying through by $\hat{\theta}_{k,ML}$ and summing over k yields:

$$\lambda = \sum_{k=1}^M \sum_{n=1}^N x_n^k \quad (5.34)$$

$$= \sum_{n=1}^N \sum_{k=1}^M x_n^k \quad (5.35)$$

$$= N. \quad (5.36)$$

Finally, substituting Eq. (5.36) back into Eq. (5.33) we obtain:

$$\hat{\theta}_{k,ML} = \frac{1}{N} \sum_{n=1}^N x_n^k. \quad (5.37)$$

Noting again that $\sum_{n=1}^N x_n^k$ is the count of the number of times that the k th value is observed, we see that the maximum likelihood estimate of θ_k is a sample proportion.

Bayesian density estimation for discrete data

In this section we discuss a Bayesian approach to density estimation for discrete data. As in the Gaussian setting, we specify a prior using a parameterized distribution and show how to compute the corresponding posterior.

An appealing feature of the solution to the Gaussian problem was that the prior and the posterior have the same distribution—both are Gaussian distributions. Among other virtues, this implies that Eq. (5.24) and Eq. (5.25) can be used *recursively*—the posterior based on earlier observations can serve as the prior for additional observations. At each step the posterior distribution remains in the Gaussian family.

To achieve a similar closure property in the discrete problem we must find a prior distribution which when multiplied by the multinomial distribution yields a posterior distribution in the same family. Clearly, this can be achieved by a prior distribution of the form:

$$p(\theta) = C(\alpha)\theta_1^{\alpha_1-1}\theta_2^{\alpha_2-1}\cdots\theta_M^{\alpha_M-1}, \quad (5.38)$$

for $\sum_i \theta_i = 1$, where $\alpha = (\alpha_1, \dots, \alpha_M)$ are hyperparameters and $C(\alpha)$ is a normalizing constant.³ This distribution, known as the *Dirichlet distribution*, has the same functional form as the multinomial, but the θ_i are random variables in the Dirichlet distribution and parameters in the multinomial distribution. The constant $C(\alpha)$ is obtained via a bit of calculus (see Appendix B):

$$C(\alpha) = \frac{\Gamma(\sum_{i=1}^M \alpha_i)}{\prod_{i=1}^M \Gamma(\alpha_i)}, \quad (5.39)$$

³The negative one in the exponent is a convention; we could redefine the α_i to remove it.

where $\Gamma(\cdot)$ is the gamma function. In the rest of this section we will not bother with calculating the normalization; once we have a distribution in the Dirichlet form we can substitute into Eq. (5.39) to find the normalization factor.

We now calculate the posterior probability:

$$p(\theta | x) \propto \theta_1^{\sum_{n=1}^N x_n^1} \theta_2^{\sum_{n=1}^N x_n^2} \dots \theta_M^{\sum_{n=1}^N x_n^M} \theta_1^{\alpha_1-1} \theta_2^{\alpha_2-1} \dots \theta_M^{\alpha_M-1} \quad (5.40)$$

$$= \theta_1^{\sum_{n=1}^N x_n^1 + \alpha_1 - 1} \theta_2^{\sum_{n=1}^N x_n^2 + \alpha_2 - 1} \dots \theta_M^{\sum_{n=1}^N x_n^M + \alpha_M - 1}. \quad (5.41)$$

This is a Dirichlet density, with parameters $\sum_{n=1}^N x_n^k + \alpha_k$. We see that to update the prior into a posterior we simply add the count $\sum_{n=1}^N x_n^k$ to the prior parameter α_k .

It is worthwhile to consider the special case of the multinomial distribution when $M = 2$. In this setting, X_n is best treated as a binary variable rather than a vector; thus: $x_n \in \{0, 1\}$. The multinomial distribution reduces to:

$$p(x_n | \theta) = \theta^{x_n} (1 - \theta)^{1-x_n}; \quad (5.42)$$

the *Bernoulli distribution*. The parameter θ encodes the probability that X_n takes the value one.

In the case $M = 2$, the Dirichlet distribution specializes to the *beta distribution*:

$$p(\theta) = C(\alpha) \theta^{\alpha_1-1} (1 - \theta)^{\alpha_2-1}, \quad (5.43)$$

where $\alpha = (\alpha_1, \alpha_2)$ is the hyperparameter. The beta distribution has its support on the interval $[0, 1]$. Plots of the beta distribution are shown in Figure 5.7 for various values of α_1 and α_2 . Note that the *uniform distribution* is the special case of the beta distribution when $\alpha_1 = 1$ and $\alpha_2 = 1$.

As the number of data points N becomes large, the sums $\sum_{n=1}^N x_n^k$ dominate the prior terms α_k in the posterior probability. In this limit, the posterior approaches the log likelihood in Eq. (5.30) and the Bayes estimate of θ approaches the maximum likelihood estimate of θ .

Mixture models

It is important to recognize that the Gaussian and multinomial densities are by no means the universally best choices of density model. Suppose, for example, if the data are continuous data restricted to the half-infinite interval $[0, \infty)$. The Gaussian, which assigns density to the entire real line, is unnatural here, and densities such as the gamma or lognormal, whose support is $[0, \infty)$, may be preferred. Similarly, the multinomial distribution treats discrete data as an unordered, finite set of values. In problems involving ordered sets, and/or infinite ranges, probability distributions such as the Poisson or geometric may be more appropriate. Maximum likelihood and Bayesian estimates are available for these distributions, and indeed there is a general family known as the *exponential family*—which includes all of the distributions listed above and many more—in which explicit formulas can be obtained. (We will discuss the exponential family in Chapter 8).

This larger family of distributions is still, however, restrictive. Consider the probability density shown in Figure 5.8. This density is bimodal and we are unable to represent it within the family of Gaussian, gamma or lognormal densities. Given a data set $\{x_n\}$ sampled from this density, we

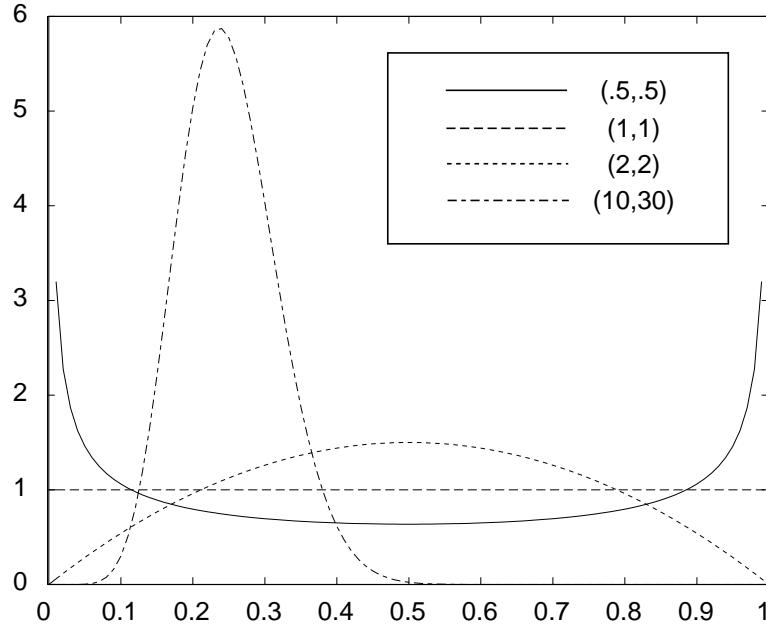


Figure 5.7: The beta(α_1, α_2) distribution for various values of the parameters α_1 and α_2 .

can naively fit a Gaussian density, but the likelihood that we achieve will in general be significantly smaller than the likelihood of the data under the true density, and the resulting density estimate will bear little relationship to the truth.

Multimodal densities often reflect the presence of *subpopulations* or *clusters* in the population from which we are sampling. Thus, for example, we would expect the density of heights of trees in a forest to be multimodal, reflecting the different distributions of heights of different species. It may be that for a particular species the heights are unimodal and reasonably well modeled by a simple density, such as a density in the exponential family. If so, this suggests a “divide-and-conquer” strategy in which the overall density estimation is broken down into a set of smaller density estimation problems that we know how to handle. Let us proceed to develop such a strategy.

Let $f_k(x | \theta_k)$ be the density for the k th subpopulation, where θ_k is a parameter vector. We define a *mixture density* for a random variable X by taking the convex sum over the component densities $f_k(x | \theta_k)$:

$$p(x | \theta) = \sum_{k=1}^K \alpha_k f_k(x | \theta_k), \quad (5.44)$$

where the α_k are nonnegative constants that sum to one:

$$\sum_{k=1}^K \alpha_k = 1. \quad (5.45)$$

The densities $f_k(x | \theta_k)$ are referred to in this setting as *mixture components* and the parameters α_k

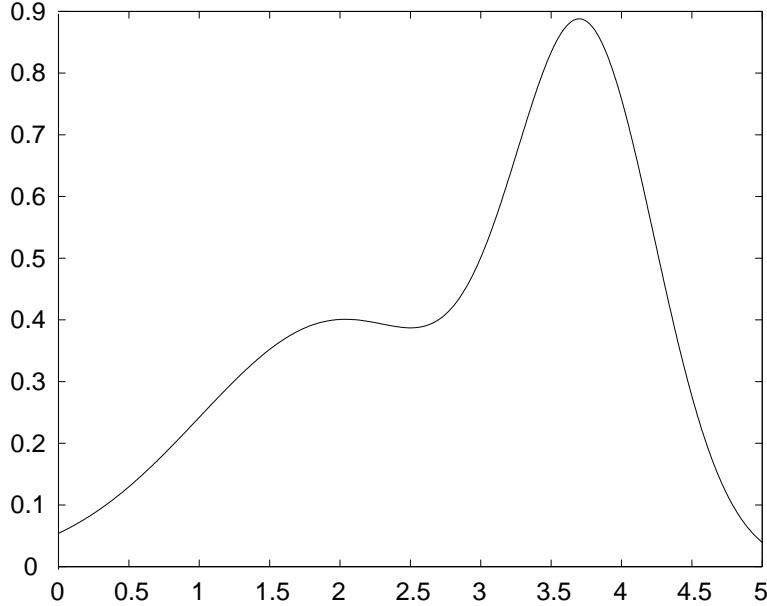


Figure 5.8: A bimodal probability density.

are referred to as *mixing proportions*. The parameter vector θ is the collection of all of the parameters, including the mixing proportions: $\theta \triangleq (\alpha_1, \dots, \alpha_K, \theta_1, \dots, \theta_K)$. That the function $p(x|\theta)$ that we have defined is in fact a density follows from the constraint that the mixing proportions sum to one.

The example shown in Figure 5.8 is a mixture density with $K = 2$:

$$p(x|\theta) = \alpha_1 \mathcal{N}(x|\mu_1, \sigma_1^2) + \alpha_2 \mathcal{N}(x|\mu_2, \sigma_2^2), \quad (5.46)$$

where the mixture components are Gaussian distributions with means μ_k and variances σ_k^2 . Gaussian mixtures are a popular form of mixture model, particular in multivariate settings (see Chapter 10).

It is illuminating to express the mixture density in Eq. (5.44) in a way that makes explicit its interpretation in terms of subpopulations. Let us do this using the machinery of graphical models. As shown in Figure 5.9, we introduce a multinomial random variable Z into our model. We also introduce an edge from Z to X . Following the recipe from Chapter 2 we endow this graph with a joint probability distribution by assigning a marginal probability to Z and a conditional probability to X . Let α_k be the probability that Z takes on its k th value; thus, $\alpha_k \triangleq p(z^k = 1)$. Moreover, conditional on Z taking on its k th value, let the conditional probability of X , $p(x|z^k = 1)$, be given by $f_k(x|\theta_k)$. The joint probability is therefore given by:

$$p(x, z^k = 1 | \theta) = p(x | z^k = 1, \theta)p(z^k = 1 | \theta) \quad (5.47)$$

$$= \alpha_k f_k(x | \theta_k), \quad (5.48)$$



Figure 5.9: A mixture model represented as a graphical model. The latent variable Z is a multinomial node taking on one of K values.

where $\theta \triangleq (\alpha_1, \dots, \alpha_K, \theta_1, \dots, \theta_K)$. To obtain the marginal probability of X we sum over k :

$$p(x | \theta) = \sum_{k=1}^K p(x, z^k = 1 | \theta) \quad (5.49)$$

$$= \sum_{k=1}^K \alpha_k f_k(x | \theta_k), \quad (5.50)$$

which is the mixture model in Eq. (5.44).

This model gives us our first opportunity to invoke our discussion of probabilistic inference from Chapter 3. In particular, given an observation x , we can use Bayes rule to invert the arrow in Figure 5.9 and calculate the conditional probability of Z :

$$p(z^k = 1 | x, \theta) = \frac{p(x | z^k = 1, \theta_k)p(z^k = 1)}{\sum_j p(x | z^j = 1, \theta_j)p(z^j = 1)} \quad (5.51)$$

$$= \frac{\alpha_k f_k(x | \theta_k)}{\sum_j \alpha_j f_j(x | \theta_j)}. \quad (5.52)$$

This calculation allows us to use the mixture model to *classify* or *categorize* the observation x into one of the subpopulations or clusters that we assume to underly the model. In particular we might classify x into the class k that maximizes $p(z^k = 1 | x, \theta)$.

Let us turn to the problem of estimating the parameters of the mixture model from data. We again assume for simplicity a sampling model in which we have N IID observations $\{x_n; n = 1, \dots, N\}$, while again noting that we will move beyond the IID setting in later chapters. The IID assumption corresponds to replicating our basic graphical model N times, yielding the plate shown in Figure 5.10. Note again that the variables Z_n are unshaded—they are *unobserved* or *latent* variables. We have introduced them into our model in order to make explicit the structural assumptions that lie behind the mixture density that we are using, but we need not assume that these variables are observed.

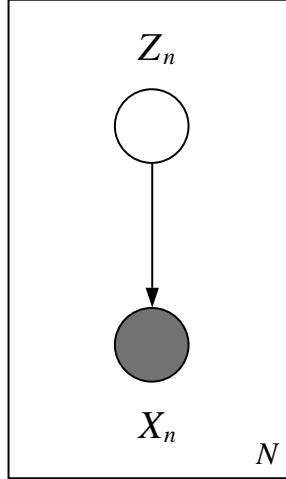


Figure 5.10: The mixture model under an IID sampling assumption.

The log likelihood is given by taking the logarithm of the joint probability associated with the model, which in the IID case becomes a sum of log probabilities. Again letting $x = (x_1, \dots, x_N)$, we have:

$$l(\theta; x) = \sum_{n=1}^N \log \sum_{k=1}^K \alpha_k f_k(x_n | \theta_k). \quad (5.53)$$

To obtain maximum likelihood estimates we take derivatives with respect to θ and set to zero. The resulting equations are, however, nonlinear and do not admit a closed-form solution; solving these equations requires iterative methods. While any of a variety of numerical methods can be used, there is a particular iterative method—the *Expectation-Maximization (EM) algorithm*—that is natural not only for mixture models but also for more general graphical models. The EM algorithm involves an alternating pair of steps, the *E step* and the *M step*. The E step involves running an inference algorithm—for example the elimination algorithm that we discussed in Chapter 3—to essentially “fill in” the values of the unobserved nodes given the observed nodes. In the case of mixture models, this reduces to the invocation of Bayes rule in Eq. (5.52). The M step treats the “filled-in” graph as if all of the filled-in values had been observed, and updates the parameters to obtain improved values. In the mixture model setting this essentially reduces to finding separate density estimates for the separate subpopulations. We will present the EM algorithm formally in Chapter 11, and present its application to mixture models in Chapter 10.

Nonparametric density estimation

In many cases data may come from a complex mechanism about which we have little or no prior knowledge. The density underlying the data may not fall into one of the “standard” forms. The density may be multimodal, but we may have no reason to suspect underlying subpopulations and may have no reason to attribute any particular meaning to the modes. When we find ourselves in

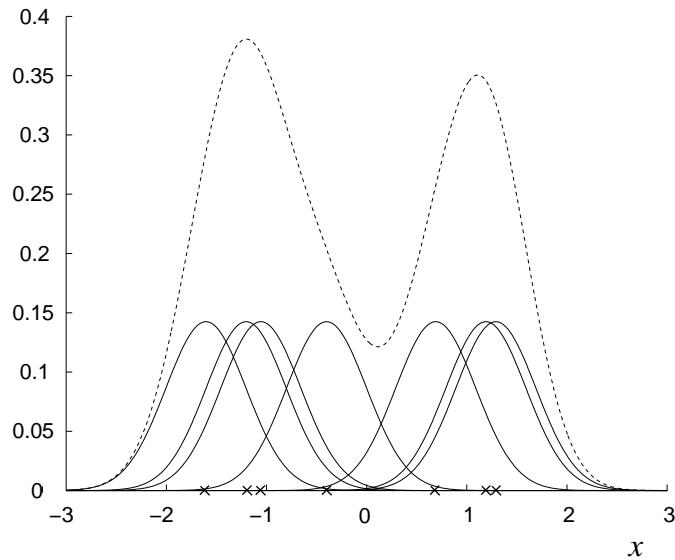


Figure 5.11: An example of kernel density estimation. The kernel functions are Gaussians centered at the data points x_n (shown as crosses on the abscissa). Each Gaussian has a standard deviation $\lambda = 0.35$. The Gaussians have been scaled by dividing by the number of data points ($N = 8$). The density estimate (shown as a dotted curve) is the sum of these scaled kernels.

such a situation—by no means uncommon—what do we do?

Nonparametric density estimation provides a general class of methods for dealing with such knowledge-poor cases. In this section we introduce this approach via a simple, intuitive nonparametric method known as a *kernel density estimator*. We return to a fuller discussion of nonparametric methods in Chapter 25.

The basic intuition behind kernel density estimation is that each data point x_n provides evidence for non-zero probability density at that point. A simple way to harness this intuition is to place an “atom” of mass at that point (see Figure 5.11). Moreover, making the assumption that the underlying probability density is smooth, we let the atoms have a non-zero “width.” Superimposing N such atoms, one per data point, we obtain a density estimate.

More formally, let $k(x, x_n, \lambda)$ be a *kernel function*—a nonnegative function integrating to one (with respect to x). The argument x_n determines the location of the kernel function; kernels are generally symmetric about x_n . The parameter λ is a general “smoothing” parameter that determines the width of the kernel functions and thus the smoothness of the resulting density estimate. Superimposing N such kernel functions, and dividing by N , we obtain a probability density:

$$\hat{p}(x) = \frac{1}{N} \sum_{n=1}^N k(x, x_n, \lambda). \quad (5.54)$$

This density is the kernel density estimate of the underlying density $p(x)$.

A variety of different kernel functions are used in practice. Simple (e.g., piecewise polynomial)

functions are often preferred, partly for computational reasons (calculating the density at a given point x requires N function evaluations). Gaussian functions are sometimes used, in which case x_n plays the role of the mean and λ plays the role of the standard deviation.

While the kernel function is often chosen a priori, the value of λ is generally chosen based on the data. This is a nontrivial estimation problem for which classical estimation methods are often of little help. In particular, it is important to understand that maximum likelihood is *not* appropriate for solving this problem. Suppose that we interpret the density in Eq. (5.54) as a likelihood function, with λ as the parameter. For most reasonable kernels, this “likelihood” increases monotonically as λ goes to zero, because the kernel assigns more probability density to the points x_n for smaller values of λ . Indeed, in the limit of $\lambda = 0$, the kernel generally approaches a delta function, giving infinite likelihood to the data. A sum of delta functions is obviously a poor density estimate.

We will discuss methods for choosing smoothing parameters in Chapter 25. As we will see, most practical methods involve some form of *cross-validation*, in which a fraction of the data are held out and used to evaluate various choices of λ . Both overly small and overly large values of λ will tend to assign small probability density to the held-out data, and this provides a rational approach to choosing λ .

The problem here is a general one, motivating a distinction between *parametric models* and *nonparametric models* and suggesting the need for distinct methods for their estimation. Understanding the distinction requires us to consider how a given model would change if the number of data points N were to increase. For parametric models the basic structure of the model remains fixed as N increases. In particular, for the Gaussian estimation problem treated in Section 5.2.1, the class of densities that are possible fits to the data remains the same whatever the value of N ; for each N we obtain a Gaussian density with estimated parameters $\hat{\mu}$ and $\hat{\sigma}^2$. Increasing the number of data points increases the precision of these estimates, but it does not increase the class of densities that we are considering. In the nonparametric case, on the other hand, the class of densities increases as N increases. In particular, with $N + 1$ data points it is possible to obtain densities with $N + 1$ modes; this is not possible with N data points.

An alternative perspective is to view the locations of the kernels as “parameters”; the number of such “parameters” increases with the number of data points. In effect, we can view nonparametric models as parametric, but with an unbounded, data-dependent, number of parameters. Indeed, in an alternative language that is often used, parametric models are referred to as “finite-dimensional models,” and nonparametric models are referred to as “infinite-dimensional models.”

It is worthwhile to compare the kernel density estimator in Eq. (5.54) to the mixture model in Eq. (5.44). Consider in particular the case in which Gaussian mixture components are used in Eq. (5.44) and Gaussian kernel functions are used in Eq. (5.54). In this case the kernel estimator can be viewed as a mixture model in which the means are fixed to the data point locations, the variances are set to λ^2 , and the mixing proportions are set to $1/N$. In what sense are the two different approaches to density estimation really different?

Again, the key difference between the two approaches is revealed when we let the number of data points N grow. The mixture model is generally viewed as a parametric model, in which case the number of mixture components, K , does not increase as the number of data points grows. This is consistent with our interpretation of a mixture model in terms of a set of K underly-

ing subpopulations—if we believe that these subpopulations exist, then we do not vary K as N increases. In the kernel estimation approach, on the other hand, we have no commitment to underlying subpopulations, and we accord no special treatment to the number of kernels. As the number of data points grows, we allow the number of kernels to grow. Moreover we generally expect that λ will shrink as N grows to allow an increasingly close fit to the details of the true density.

There are several caveats to this discussion. First, in the mixture model setting, we may not know the number K of mixture components in practice and we may wish to estimate K from the data. This is a model selection problem (see Section 5.3). Solutions to model selection problems generally involve allowing K to increase as the number of data points increases, based on the fact that more data points are generally needed to provide more compelling evidence for multiple modes. Second, mixture models can also be used nonparametrically. In particular, a *mixture sieve* is a mixture model in which the number of components is allowed to grow with the number of data points. This differs from kernel density estimation in that the location of the mixture components are treated as free parameters rather than being fixed at the data points; moreover, each mixture component generally has its own (free) scale parameter. Also, the growth rate of the number of “parameters” in mixture sieves is slower than that of kernel density estimation (e.g., $\log N$ vs. N). As this discussion begins to suggest, however, it becomes difficult to enforce a clear boundary between parametric and nonparametric methods. A given approach can be treated in one way or the other, depending on a modeler’s goals and assumptions.

There is a general tradeoff between flexibility and statistical efficiency that is relevant to this discussion. If the underlying “true” density is a Gaussian, then we probably want to estimate this density using a parametric approach, we can also use a kernel density estimate. The latter estimate will eventually converge to the true density, but it may require very many data points. A parametric estimator will converge more rapidly. Of course, if the true density is not a Gaussian, then the parametric estimate would still converge, but to the wrong density, whereas the nonparametric estimate would eventually converge to the true density. In sum, if we are willing to make more assumptions then we get faster convergence, but with the possibility of poor performance if reality does not match our assumptions. Nonparametric estimators allow us to get away with fewer assumptions, while requiring more data points for comparable levels of performance.

There is also a general point to be made with respect to the representation of densities in graphical models. As suggested in Figure 5.12, there are two ways to represent a multi-modal density as a graphical model. As shown in Figure 5.12(a), we can allow the class of densities $p(x)$ at node X to include multi-modal densities, such as mixtures or kernel density estimates. Alternatively, we can use the “structured” model depicted in Figure 5.12(b), where we obtain a mixture distribution for X_n by marginalizing over the latent variable Z_n . Although it may seem natural to reserve the latter representation for parametric modeling, in particular for the setting in which we attribute a “meaning” to the latent variable, such a step is in general unwarranted. The mixture sieve exemplifies a situation in which we may wish to use graphical machinery to represent the structure of a nonparametric model explicitly. In general, the choices of how to use and how to interpret graphical structure are modeling decisions. While we may wish to use graphical representations to express domain-specific structural knowledge, we may also be guided by other factors, including mathematical convenience and the availability of computational tools.

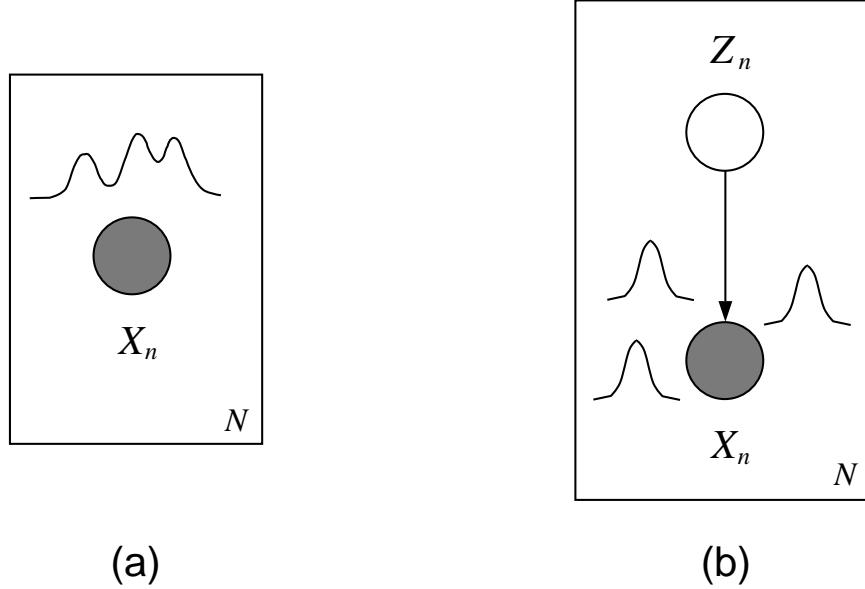


Figure 5.12: Two ways to represent a multi-modal density within the graphical model formalism. (a) The local probability model at each node is a mixture or a kernel density estimate. (b) A latent variable is used to represent mixture components explicitly; marginalizing over the latent variable yields a mixture model for the observable X_n .

There is nothing inappropriate about letting such factors be a guide, but in doing so we must be cautious about any interpretation or meaning that we attach to the model.

Summary of density estimation

Our goal in this section has not been to provide a full treatment of density estimation; indeed we have only scratched the surface of what is an extensive literature in statistics. We do hope, however, to have introduced a few key ideas—the calculation of maximum likelihood and Bayesian parameter estimates for Gaussian and multinomial densities, the use of mixture models to obtain a richer class of density models, and the distinction between parametric and nonparametric density estimation. Each of these ideas will be picked up and pursued in numerous contexts throughout the book.

5.2.2 Regression

In a *regression model* the goal is to model the dependence of a *response* or *output* variable Y on a *covariate* or *input* variable X . We capture this dependence via a conditional probability distribution $p(y|x)$. In graphical model terms, we have a two-node model in which X is the parent and Y is the child (see Figure 5.13).

One way to treat regression problems is to estimate the joint density of X and Y and to calculate



Figure 5.13: A regression model.

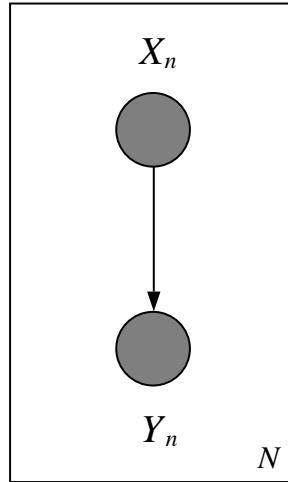


Figure 5.14: The IID regression model represented graphically.

the conditional $p(y|x)$ from the estimated joint. This approach forces us to model X , however, which may not be desired. Indeed, in many applications of regression, X is high-dimensional and hard to model. Moreover, the observations of X are often fixed by experimental design or another form of non-random process, and it is problematic to treat them via a simple sampling model, such as the IID model. In summary, it is necessary to develop methods appropriate to conditional densities.

Our discussion here will be brief, with a focus on basic representational issues.

We assume that we have a set of pairs of observed data, $\{(x_n, y_n); n = 1, \dots, N\}$, where x_n is an observation of the input variable and y_n is a corresponding observation of the output variable. We again assume an independent, identical distributed (IID) sampling model for simplicity. The graphical representation of the IID regression model is shown as a plate in Figure 5.14.

Let us now consider some of the possible choices for the conditional probability model $p(y_n | x_n)$.

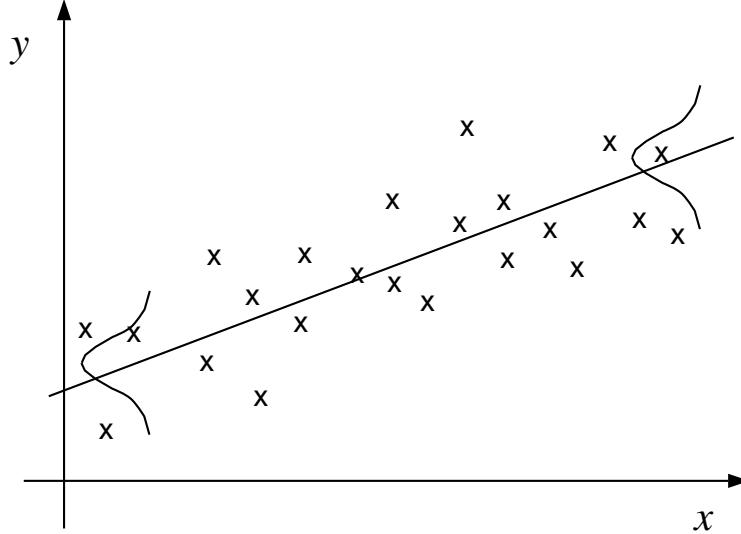


Figure 5.15: The linear regression model expresses the response variable Y in terms of the conditional mean function—the line in the figure—and input-independent random variation around the conditional mean.

As in the case of density estimation, we have a wide spectrum of possibilities, including parametric models, mixture models, and nonparametric models. We will discuss these models in detail in Chapters 6, 10, and 25, respectively, but let us sketch some of the possibilities here.

A *linear regression* model expresses Y_n as the sum of (1) a purely deterministic component that depends parametrically on x_n , and (2) a purely random component that is functionally independent of x_n :

$$Y_n = \beta^T x_n + \epsilon_n, \quad (5.55)$$

where β is a parameter vector and ϵ_n is a random variable having zero mean. Taking the conditional expectation of both sides of this equation yields $E[Y_n | x_n] = \beta^T x_n$. Thus the linear regression model expresses Y_n in terms of input-independent random variation ϵ_n around the conditional mean $\beta^T x_n$ (see Figure 5.15). The choice of the distribution of ϵ_n , which completes the specification of the model, is analogous to the choice of a density model in density estimation, and depends on the nature of Y_n . “Linear regression” generally refers to the case in which Y_n is real-valued and the distribution is taken to be $\mathcal{N}(0, \sigma^2)$. (In Chapter 8 we will be discussing “generalized linear models,” which are regression models that are appropriate for other types of response variables). In the linear regression case, we have:

$$P(y_n | x_n, \theta) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp \left\{ -\frac{1}{2\sigma^2} (y_n - \beta^T x_n)^2 \right\}, \quad (5.56)$$

where for simplicity we have taken y_n to be univariate. The parameter vector θ includes β , which determines the conditional mean, and σ^2 , which is the variance of ϵ_n and determines the scale of

the variation around the conditional mean.

Linear regression is in fact broader than it may appear at first sight, in that the function $\beta^T x_n$ need only be linear in β and in particular may be nonlinear in x_n . Thus the model:

$$Y_n = \beta^T \phi(x_n) + \epsilon_n, \quad (5.57)$$

where $\phi(\cdot)$ is a vector-valued function of x_n , is a linear regression model. This model is a parametric model in that $\phi(\cdot)$ is fixed and our freedom in modeling the data comes only from the finite set of parameters β .

The problem of estimating the parameters of regression models is in principle no different from the corresponding estimation problem for density estimation. In the maximum likelihood approach, we form the log likelihood:

$$l(\theta; x) = \sum_{n=1}^N \log p(y_n | x_n, \theta), \quad (5.58)$$

take derivatives with respect to θ , set to zero and (attempt to) solve. We will discuss the issues that arise in carrying out this calculation in later chapters.

Conditional mixture models

Mixture models provide a way to move beyond the strictures of linear regression modeling. We can consider both a broader class of conditional mean functions as well as a broader class of density models for ϵ_n . Consider in particular the graphical model shown in Figure 5.16(a). We have introduced a multinomial latent variable Z_n that depends on the input X_n ; moreover, the response Y_n depends on both X_n and Z_n . This graph corresponds to the following probabilistic model:

$$p(y_n | x_n, \theta) = \sum_{k=1}^K p(z_n^k = 1 | x_n, \theta) p(y_n | z_n^k = 1, x_n, \theta), \quad (5.59)$$

a *conditional mixture model*. Each mixture component $p(y_n | z_n^k = 1, x_n)$ corresponds to a different regression model, one for each value of k . The mixing proportions $p(z_n^k = 1 | x_n)$ “switch” among the regression models as a function of x_n . Thus, as suggested in Figure 5.16(a), the mixing proportions can be used to pick out regions of the input space where different regression functions are used. We can parameterize both the mixing proportions and the regression models and estimate both sets of parameters from data. This is a “divide-and-conquer” methodology in the regression domain. (We provide a fuller description of this model in Chapter 10).

The example in Figure 5.16(a) utilizes mixing proportions that are sharp, nearly binary functions of X_n , but it is also of interest to consider models in which these functions are smoother, allowing overlap in the component regression functions. Indeed, in the limiting case we obtain the model shown in Figure 5.16(b) in which the latent variable Z_n is independent of X_n . Here the presence of the latent variable serves only to induce multimodality in the conditional distribution $p(y_n | x_n)$. Much as in the case of density estimation, such a regression model may arise from a set of subpopulations, each characterized by a different “conditional mean.”

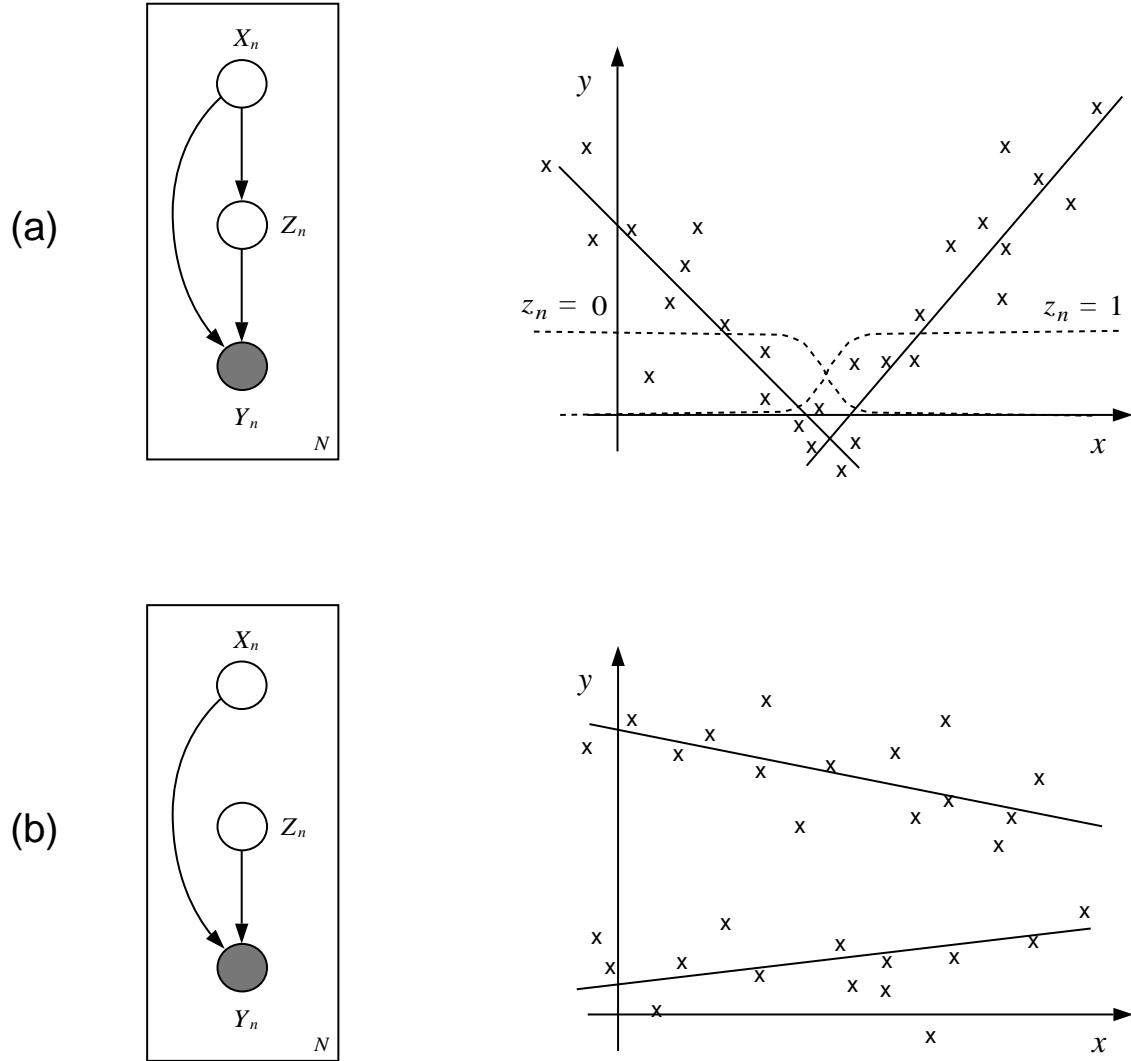


Figure 5.16: Two variants of conditional regression model. In (a), the latent variable Z_n is dependent on X_n . This corresponds to breaking up the input space into (partially overlapping) regions labeled by the values of Z_n . An example with binary Z_n is shown in the figure on the right, where the dashed line labeled by $z_n = 1$ is the probability $p(z_n = 1 | x_n)$, and the dashed line labeled by $z_n = 0$ is the probability $p(z_n = 0 | x_n)$. The two lines are the conditional means of the regressions, $p(y_n | z_n, x_n)$, for the two values of z_n , with the leftmost line corresponding to $z_n = 0$ and the rightmost line corresponding to $z_n = 1$. In (b), the latent variable Z_n is independent of X_n . This corresponds to total overlap of the regions corresponding to the values of Z_n and yields an input-independent mixture density for each value of x_n .

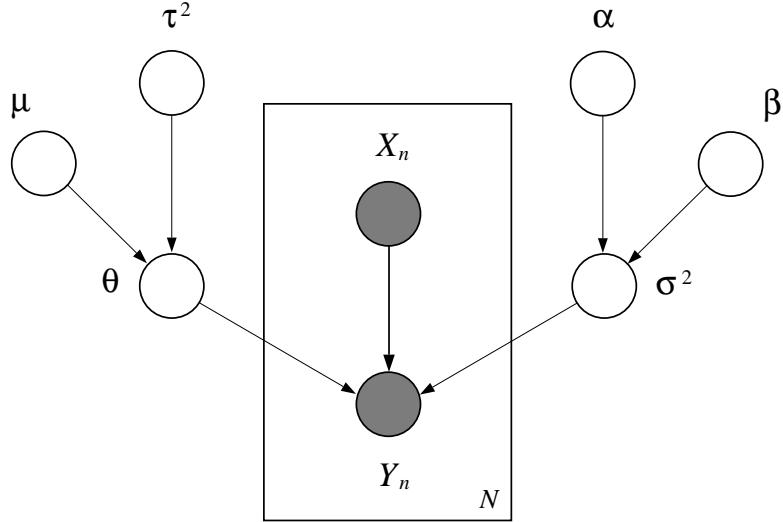


Figure 5.17: A Bayesian linear regression model. The parameter vector θ is endowed with a Gaussian prior, $\mathcal{N}(\mu, \tau^2)$. The variance σ^2 is endowed with an inverse gamma prior, $IG(\alpha, \beta)$.

The parameters of conditional mixture models can be estimated using the EM algorithm, as discussed in Chapter 10. Indeed, EM can be used quite generically for latent variable models such as those in Figure 5.16.

Nonparametric regression

Let us briefly consider the nonparametric approach to regression. While it is possible to use nonparametric methods to expand the repertoire of probability models for ϵ_n , a more common usage of nonparametric ideas involves allowing a wider class of conditional mean functions. The basic idea is to break up the input space into (possibly overlapping) regions, with one such region for each data point. Let us give an example from the class of methods known as *kernel regression*. As in kernel density estimation, let $k(x, x_n, \lambda)$ be a *kernel function* centered around the data point x_n . Denoting the conditional mean function as $f(x)$, we form an estimate as follows:

$$\hat{f}(x) = \frac{\sum_{n=1}^N k(x, x_n, \lambda) y_n}{\sum_{m=1}^N k(x, x_m, \lambda)} \quad (5.60)$$

That is, we estimate the conditional mean at x as the convex sum of the observed values y_n , where the weights in the sum are given by the normalized values of the kernel functions, one for each x_n , evaluated at x . Given that kernel functions are generally chosen to be “local,” having most of their support near x_n , we see that the kernel regression estimate at x is a local average of the values y_n in the neighborhood of x .

We can once again forge a link between the mixture model approach and the nonparametric kernel regression approach. As we ask the reader to verify in Exercise ??, taking the conditional

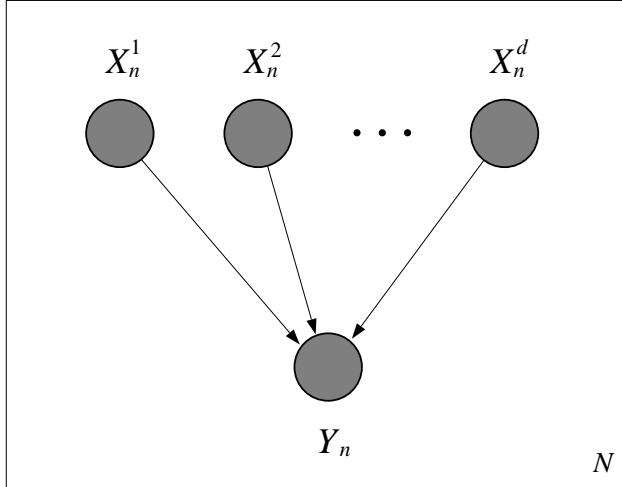


Figure 5.18: A graphical representation of the regression model in which the components of the input vector are treated as explicit nodes.

mean of Eq. (5.59) yields a weighted sum of conditional mean functions, one for each component k , where the weights are the mixing proportions $p(z_n^k = 1 | x_n)$. The kernel regression estimate in Eq. (5.60) can be viewed as an instance of this model, if we treat the normalized kernels $k(x, x_n, \lambda) / \sum_{m=1}^N k(x, x_m, \lambda)$ as mixing proportions, and the values y_n as (constant) conditional means. The same comments apply to this reduction as to the analogous reduction in the case of density estimation. In particular, as N increases, the number of components K in a parametric conditional mixture model generally remain fixed, whereas the number of kernels in the kernel regression model grow. We can, however, consider *conditional mixture sieves*, and obtain a nonparametric variant of a mixture model.

Bayesian approaches to regression

All of the models that we have considered in this section can be treated via Bayesian methods, where we endow the parameters (or entire conditional mean functions) with prior distributions. We then invoke Bayes rule to calculate posterior distributions. Figure 5.17 illustrates one such Bayesian regression model.

Remarks

Let us make one final remark regarding the graphical representation of regression models. Note that in this section we have treated the input variables X_n as single nodes, not availing ourselves of the opportunity to represent the components of these vector-valued variables as separate nodes (see Figure 5.18). This is consistent with our treatment of X_n as fixed variables to be conditioned on; representing the components as separate nodes would imply marginal independence between the components, an assumption that we may or may not wish to make. It is important to note,

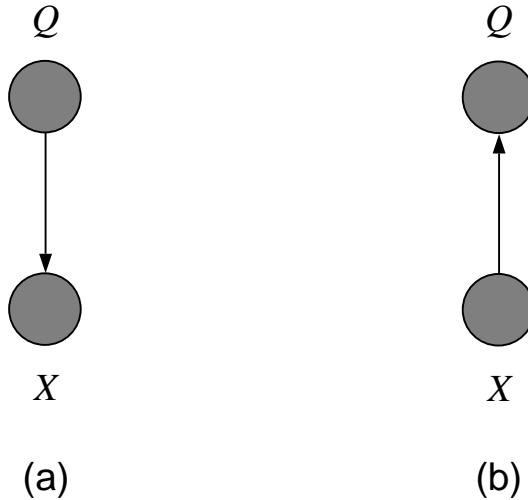


Figure 5.19: (a) The generative approach to classification represented as a graphical model. Fitting the model requires estimating the marginal probability $p(q)$ and the conditional probability $p(x|q)$. (b) The discriminative approach to classification represented as a graphical model. Fitting the model requires estimating the conditional probability $p(q|x)$.

however, that regression methods are agnostic regarding modeling assumptions about the conditioning variables. Regression methods form an estimate of $p(y|x)$ and this conditional density can be composed with an estimate of $p(x)$ to obtain an estimate of the joint. This allows us to use regression models as components of larger models. In particular, in the context of a graphical model in which a node A has multiple parents B_1, B_2, \dots, B_k , we are free to use regression methods to represent $p(A|B_1, B_2, \dots, B_k)$, regardless of the modeling assumptions made regarding the nodes B_i . Indeed each of the B_i may themselves be modeled in terms of regressions on variables further “upstream.”

5.2.3 Classification

Classification problems are related to regression problems in that they involve pairs of variables. The distinguishing feature of classification problems is that the response variable ranges over a finite set, a seemingly minor issue that has important implications.

In classification we often refer to the covariate X as a *feature vector*, and the corresponding discrete response, which we denote by Q , as a *class label*. We typically view the feature vectors as descriptions of objects, and the goal is to label the objects, i.e., to classify the objects into one of a finite set of categories.

There are two basic approaches to classification problems, which can be interpreted graphically in terms of the direction of the edge between X and Q . The first approach, which we will refer to as *generative*, is based on the graphical model shown in Figure 5.19(a), in which there is an arrow from Q to X . This approach is closely related to density estimation—for each value of the discrete

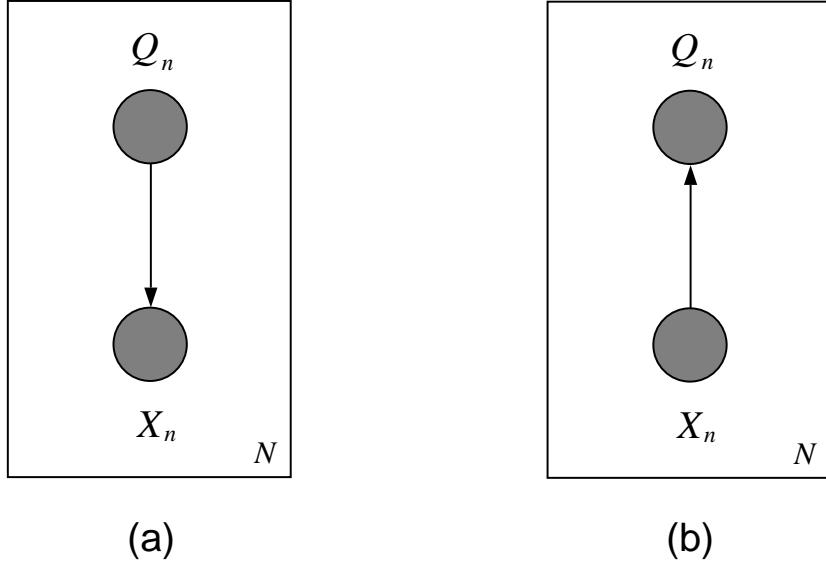


Figure 5.20: The IID classification models for the (a) generative approach and (b) discriminative approach.

variable Q we have a density, $p(x|q)$, which we refer to as a *class-conditional density*. We also require the marginal probability $p(q)$, which we refer to as the *prior probability* of the class Q (it is the probability of the class before a feature vector X is observed). This marginal probability is required if we are going to be able to “invert the arrow” and compute $p(q|x)$ —the *posterior probability* of class Q .

The second approach to classification, which we refer to as *discriminative*, is closely related to regression. Here we represent the relationship between the feature vectors and the labels in terms of an arrow from X to Q (see Figure 5.19(b)). That is, we represent the relationship in terms of the conditional probability $p(q|x)$. When classifying an object we simply plug the corresponding feature vector x into the conditional probability and calculate $p(q|x)$. Performing this calculation, which tells us which class label has the highest probability, makes no reference to the marginal probability $p(x)$ and, as in regression, we may wish to abstain from incorporating such a marginal into the model.

As in regression, we have a set of data pairs $\{(x_n, q_n) : n = 1, \dots, N\}$, assumed IID for simplicity. The representations of the classification problem as plates are shown in Figure 5.20.

Once again we postpone a general presentation of particular representations for the conditional probabilities in classification problems until later chapters. But let us briefly discuss a canonical example that will illustrate some typical representational choices, as well as illustrate some of the relationships between the generative and the discriminative approaches to classification. This example and several others will be developed in considerably greater detail in later chapters.

We specialize to two classes. Let us choose Gaussian class-conditional densities with equal covariance matrices for the two classes. An example of these densities (where we have assumed equal

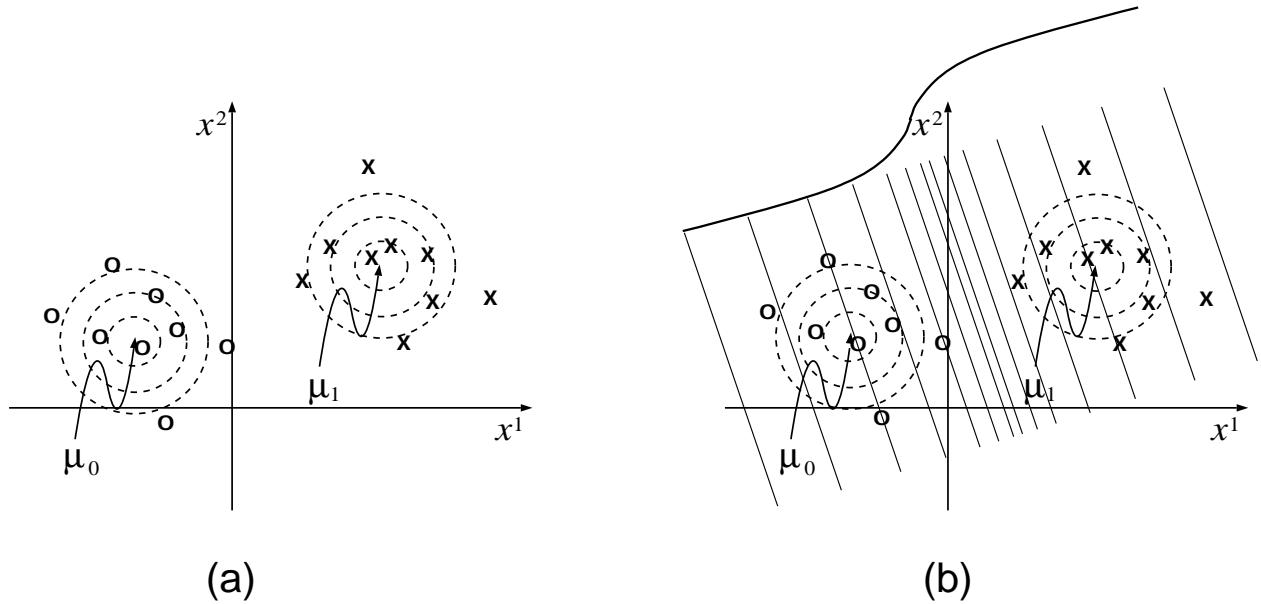


Figure 5.21: (a) Contour plots and samples from two Gaussian class-conditional densities for two-dimensional feature vectors $x_n = (x_n^1, x_n^2)$. The Gaussians have means μ_0 and μ_1 for class $q_n = 0$ and $q_n = 1$, respectively, and equal covariance matrices. (b) The solid lines are the contours of the posterior probability, $p(q_n = 1 | x_n)$. In the direction orthogonal to the linear contours, the posterior probability is a monotonically increasing function given by (Eq. (5.61)). This function is sketched at the top of the figure.

class priors) is shown in Figure 5.21(a). We use Bayes rule to compute the posterior probability that a given feature vector x_n belongs to class $q_n = 1$. Intuitively, we expect to obtain a ramp-like function which is zero in the vicinity of the class $q_n = 0$, increases to one-half in the region between the two classes, and approaches one in the vicinity of the class $q_n = 1$. This posterior probability function is shown in Figure 5.21(b), where indeed we see the ramp-like shape.

Analytically, as we show in Chapter 7, for Gaussian class-conditional densities the ramp-like posterior probability turns out to be the *logistic function*:

$$p(q_n = 1 | x_n) = \frac{1}{1 + e^{-\theta^T x_n}}, \quad (5.61)$$

where θ is a parameter vector that depends on the particular choices of means and covariances for the class-conditional densities, as well as the class priors. The inner product between θ and x_n is a projection operation that is responsible for the linear contours that we see in Figure 5.21(b).

Given these parametric forms for the class-conditional densities (the Gaussian densities) and the posterior probability (the logistic function), we must specify how to estimate the parameters based on the data. It is here that the generative and discriminative approaches begin to diverge. From the generative point of view, the problem is that of estimating the means and covariances of the Gaussian class-conditional densities, as well as the class priors. These are density estimation

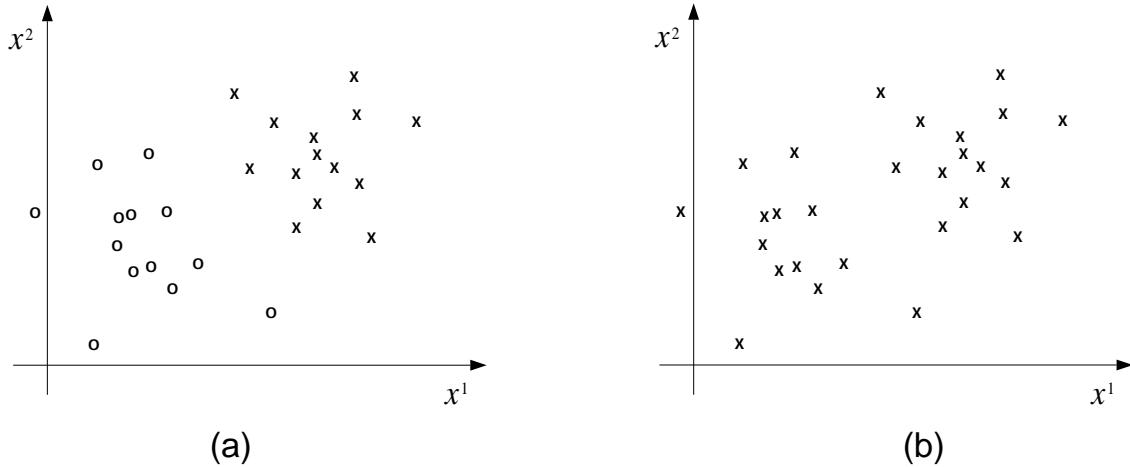


Figure 5.22: (a) A classification problem with the class $q_n = 0$ labeled with a “0” and the class $q_n = 1$ labeled with a “x”. (b) The same feature vectors x_n as in (a), but with the labels erased.

problems, and the machinery of Section 5.2.1 is invoked to solve them. With these density estimates in hand, we derive an estimate of θ and thereby calculate an estimate of the posterior probability. Essentially, the goal is to model the classes, without any direct attempt to discriminate between the classes.

In the discriminative approach, on the other hand, the logistic function is the central object of analysis. Indeed, in Chapter 7, we describe a regression-like method for estimating θ directly from data, without making reference to the means and covariances of an underlying generative model. Intuitively, this method can be viewed as an attempt to orient and position the ramp-like posterior probability in Figure 5.21(b) so as to assign a posterior probability that is near zero to the points x_n having label $q_n = 0$, and a posterior probability near one to the points x_n having label $q_n = 1$. Essentially, the goal is to discriminate between the classes, without any direct attempt to model the classes.

More generally, in a discriminative approach to classification we are not restricted to the logistic function, or to any other function that is derived from a generative model. Rather we can choose functions whose contours appear to provide a natural characterization of boundaries between classes. On the other hand, it may not always be apparent how to choose such functions, and in such cases we may prefer to take advantage of the generative approach, in which the boundaries arise implicitly via Bayes rule. In general, both the discriminative and the generative approaches are important tools to have in a modeling toolbox.

Mixture models revisited

Suppose we consider a classification problem in which none of the class labels are present. Is this a sensible problem to pose? What can one possibly learn from unlabeled data, particularly data that are completely unlabeled?

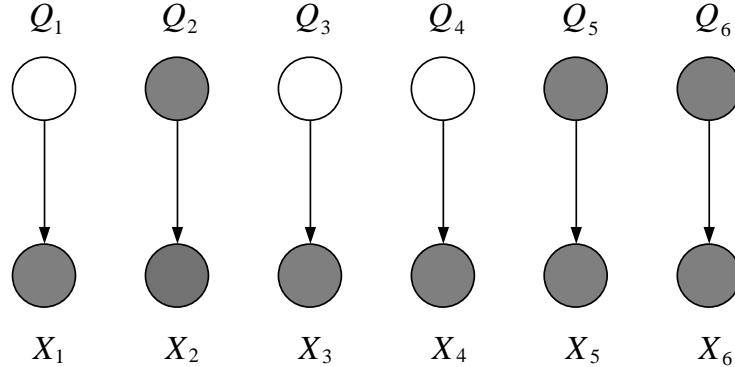


Figure 5.23: A model for partially labeled data in which the feature vectors x_2, x_5 and x_6 are labeled and the other feature vectors are unlabeled.

Consider Figure 5.22(a), where we have depicted a typical classification problem with two classes. Now consider Figure 5.22(b), where we have retained the feature vectors x_n , but erased the labels q_n . As this latter plot makes clear, although the labels are missing, there is still substantial statistical structure in the problem. Rather than solving a classification problem, we can solve a *clustering* problem, making explicit the fact that the data appear to fall into two clusters and assigning feature vectors to clusters.

In fact we have already solved this problem. The mixture model approach to density estimation discussed in Section 5.2.1 treats the density in terms of a set of underlying “subpopulations” labeled by a latent variable Z . The inferential calculation $p(z_n | x_n)$ given in Eq. (5.52) explicitly calculates the probability that the feature vector x_n belongs to each of the subpopulations.

The relationship between classification and mixture models is also clarified by comparing the “generative” graphical model in Figure 5.19(b) and the mixture model in Figure 5.9(a). These are the same graphical model—the only difference is the shading, corresponding to the assumption that the labels Q_n are observed in classification whereas the latent variables Z_n are unobserved in mixture modeling. In the setting of unlabeled data the generative classification model becomes identical to a mixture model.

In a more general setting we may have a “partially labeled” case in which the labels Q_n are observed for some data points and unobserved for other data points. This situation is represented graphically in Figure 5.23. We will be able to treat the problem of estimation in this case using the EM algorithm; indeed this “partially labeled” case requires no additional machinery beyond that already required for the mixture model.

It is common to refer to classification and regression models as “supervised learning” models and to refer to density estimation models as “unsupervised learning” models. In the comparison between mixture models and classification models just discussed, the distinction refers to the observation of the labels Q_n ; one says that the labels in classification are provided by a “supervisor.” While this terminology can be useful in making broad distinctions between models, it is our view that the terminology does not reflect a fundamental underlying distinction and we will tend to avoid its use in this book. It is our feeling that many models are neither entirely “supervised” nor entirely

“unsupervised,” and invoking the distinction often forces us to group together methods that have little in common as well as to separate methods that are closely related. We feel that a better way to understand relationships between models is to make them explicit as graphs. Models can then be compared in terms of graphical features such as which variables are considered latent and which observed, the directionalities of arcs that are used to represent conditional relationships, and the presence or absence of particular structural motifs.

Remarks

We have already indicated a relationship between mixture models and classification, but there are other roles for mixture models in the classification setting. In particular, we can use mixtures as class-conditional densities in the generative approach to classification, just as we used mixture models in the density estimation setting to extend the range of models that we considered. Also, in the context of the discriminative approach to classification, we can use conditional mixtures to represent the posterior probability $p(q|x)$, breaking this function into overlapping pieces, much as we did with the conditional mean in the case of regression.

Similarly, nonparametric methods have many roles to play in classification models. We can either extend the generative approach to allow nonparametric estimates of the class-conditional densities, or extend the discriminative approach to allow nonparametric estimates of the posterior probability.

Finally, there are once again Bayesian approaches in all of these cases. From a graphical point of view, these Bayesian approaches essentially involve making the parameters explicit as nodes, and using hyperparameters to express prior probability distributions on these nodes.

5.3 Model selection and model averaging

Thus far we have assumed that a specific model has been chosen in advance and we have focused on representing the model graphically and estimating its parameters. In some cases this assumption is reasonable—the model is determined by the problem and there is no need to consider data-driven approaches to choosing the model. More commonly, however, we wish to use the data to make informed choices regarding the model. We present a brief discussion of this problem—known as the *model selection* problem—in this section, anticipating our more detailed presentation in Chapter 26.

We consider a class \mathcal{M} of possible models, letting $m \in \mathcal{M}$ denote a specific model in this family. We also augment our earlier notation to include explicit reference to the model; thus, $p(x|\theta, m)$ refers to the probability model for the random variable X , given a specific model and a specific choice of parameters for that model.⁴ Also, in the Bayesian approach, $p(\theta|m)$ refers to the prior probability that we attach to the parameters θ , and $p(\theta|x, m)$ refers to the corresponding posterior. We wish to develop methods for choosing m based on the data x .

Let us begin with the Bayesian approach. Recall that unknowns are treated as random variables in the Bayesian approach; thus we introduce a random variable M to denote the model. The range

⁴For simplicity we use the same notation θ to represent the parameters in each of the models; in general we could allow the parameterization to vary with m .

of M is \mathcal{M} , and m denotes a realization of M . The goal of Bayesian analysis is to calculate the posterior probability of M , conditioning on the data x :

$$p(m | x) = \frac{p(x | m)p(m)}{p(x)}. \quad (5.62)$$

Note two important features of this equation. First, as in the case of parameter estimation, we require a prior probability; in particular, we need to specify the prior probability $p(m)$ of the model m . Second, note the absence of explicit mention of the parameter θ . The probabilities needed for Bayesian model selection are *marginal* probabilities.

Let us consider this latter issue in more detail. The calculation of the posterior probability in Eq. (5.62) requires the probability $p(x | m)$, a conditional probability that is referred to as the *marginal likelihood*. We compute the marginal likelihood from the likelihood by integrating over the parameters:

$$p(x | m) = \int p(x, \theta | m) d\theta \quad (5.63)$$

$$= \int p(x | \theta, m)p(\theta | m) d\theta, \quad (5.64)$$

where the prior probability $p(\theta | m)$ plays the role of a weighting function. Multiplying the marginal likelihood by the prior probability $p(m)$ yields the desired posterior $p(m | x)$, up to the normalization factor $p(x)$.

If we wish to use the posterior to *select* a model, then we must collapse the posterior to a point. As in the case of parameter estimation, various possibilities present themselves; in particular, a popular approach is to pick the model that maximizes the posterior probability. An advantage of this approach is that it obviates the need to calculate the normalization constant $p(x)$.

More generally, however, the Bayesian approach aims to use the entire posterior. To illustrate the use of the model posterior, let us again consider the problem of prediction. Taking X_{new} to be conditionally independent of X , given θ and m , we have:

$$p(x_{new} | x) = \int \int p(x_{new}, \theta, m | x) d\theta dm \quad (5.65)$$

$$= \int \int p(x_{new} | \theta, m)p(\theta, m | x) d\theta dm \quad (5.66)$$

$$= \int \int p(x_{new} | \theta, m)p(\theta | x, m)p(m | x) d\theta dm. \quad (5.67)$$

From this latter equation, we see that a full Bayesian approach to prediction requires two posterior probabilities: the model posterior $p(m | x)$ from Eq. (5.62) and the parameter posterior $p(\theta | x, m)$ from Eq. (5.1). These posteriors can be viewed as “weights” for the prediction $p(x_{new} | \theta, m)$; the total prediction can be viewed as a “weighted prediction.” This approach to prediction is referred to as *model averaging*.

It should be acknowledged that it is a rare circumstance in which the integrals in Eq. (5.64) and Eq. (5.67) can be done exactly, and Bayesian model averaging and model selection generally involve making approximations. We will discuss some of these approximations in Chapter 26.

Frequentist approaches to model selection avoid the use of prior probabilities and Bayes rule. Rather, one considers various model selection *procedures*, and evaluates these procedures in terms of various frequentist criteria. For example, one could consider a scenario in which the true probability density is assumed to lie within the class \mathcal{M} , and ask that a model selection procedure pick the true model with high frequency. Alternatively, one could ask that the procedure select the “best” model in \mathcal{M} , where “best” is defined in terms of a measure such as the Kullback-Leibler divergence between a model and the true probability density.

It is important to understand that maximum likelihood itself cannot be used as a model selection procedure. Augmenting a model with additional parameters cannot decrease the likelihood, and thus maximum likelihood will prefer more complex models. More complex models may of course be better than simpler models, if they provide access to probability densities that are significantly closer to the true density, but at some point there are diminishing returns and more complex models principally provide access to additional poor models. The fact that we have to estimate parameters implies that with some probability we will select one of the poor models. Thus the “variance” introduced by the parameter estimation process can lead to poorer performance with a more complex model. Maximum likelihood is unable to address this “overfitting” phenomenon.

One approach to frequentist model selection is to “correct” maximum likelihood to account for the variance due to parameter estimation. The AIC method to be discussed in Chapter 26 exemplifies this approach. An alternative approach, also discussed in Chapter 26, is the *cross-validation* idea, in which the data are partitioned in subsets, with one subset used to fit parameters for various models, and another subset used to evaluate the resulting models.

5.4 Appendix A

In this section we calculate the posterior density of μ in the univariate Gaussian density estimation problem. Recall that the joint probability of x and μ is given by:

$$p(x, \mu) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp \left\{ -\frac{1}{2\sigma^2} \sum_{n=1}^N (x_n - \mu)^2 \right\} \frac{1}{(2\pi\tau^2)^{1/2}} \exp \left\{ -\frac{1}{2\tau^2} (\mu - \mu_0)^2 \right\}, \quad (5.68)$$

and the problem is to normalize this joint probability.

Let us focus on the terms involving μ , treating all other terms as “constants” and dropping them throughout. We have:

$$p(x, \mu) \propto \exp \left\{ -\frac{1}{2\sigma^2} \sum_{n=1}^N (x_n^2 - 2x_n\mu + \mu^2) - \frac{1}{2\tau^2} (\mu^2 - 2\mu_0\mu + \mu_0^2) \right\} \quad (5.69)$$

$$= \exp \left\{ -\frac{1}{2} \sum_{n=1}^N \left[\frac{1}{\sigma^2} (x_n^2 - 2x_n\mu + \mu^2) + \frac{1}{\tau^2} \left(\frac{\mu^2}{N} - 2\frac{\mu_0\mu}{N} + \frac{\mu_0^2}{N} \right) \right] \right\} \quad (5.70)$$

$$= \exp \left\{ -\frac{1}{2} \sum_{n=1}^N \left[\left(\frac{1}{\sigma^2} + \frac{1}{N\tau^2} \right) \mu^2 - 2 \left(\frac{x_n}{\sigma^2} + \frac{\mu_0}{N\tau^2} \right) \mu + C \right] \right\} \quad (5.71)$$

$$\propto \exp \left\{ -\frac{1}{2} \left[\left(\frac{N}{\sigma^2} + \frac{1}{\tau^2} \right) \mu^2 - 2 \left(\frac{N\bar{x}}{\sigma^2} + \frac{\mu_0}{\tau^2} \right) \mu \right] \right\} \quad (5.72)$$

$$\propto \exp \left\{ -\frac{1}{2} \left(\frac{N}{\sigma^2} + \frac{1}{\tau^2} \right) \left[\mu^2 - 2 \left(\frac{N}{\sigma^2} + \frac{1}{\tau^2} \right)^{-1} \left(\frac{N\bar{x}}{\sigma^2} + \frac{\mu_0}{\tau^2} \right) \mu \right] \right\} \quad (5.73)$$

$$= \exp \left\{ -\frac{1}{2\tilde{\sigma}^2} [\mu^2 - 2\tilde{\mu}\mu] \right\}, \quad (5.74)$$

where

$$\tilde{\sigma}^2 = \left(\frac{N}{\sigma^2} + \frac{1}{\tau^2} \right)^{-1} \quad (5.75)$$

and

$$\tilde{\mu} = \frac{N/\sigma^2}{N/\sigma^2 + 1/\tau^2} \bar{x} + \frac{1/\tau^2}{N/\sigma^2 + 1/\tau^2} \mu_0, \quad (5.76)$$

This identifies the posterior as a Gaussian distribution with mean $\tilde{\mu}$ and variance $\tilde{\sigma}^2$.

5.5 Historical remarks and bibliography

Chapter 6

Linear Regression and the LMS algorithm

In the following chapters we discuss elementary building blocks for graphical models. We begin with the simple case of a single continuous-valued node whose mean is a linear function of the values of its parents. The parents can be discrete or continuous.

In specifying the linear regression model in Chapter 5, we made several assumptions in addition to the linearity assumption, in particular the assumption of IID sampling and the assumption of a Gaussian distribution for the variation around the conditional mean. These latter assumptions yielded a fully specified probabilistic model, enabling us to define a likelihood and thereby invoke frequentist or Bayesian statistical methods to estimate parameters. It might be useful, however, to step momentarily outside of the probabilistic framework and ask why we consider a parameter estimation problem to be well posed once we have defined a “fully specified probabilistic model.” In the current chapter, we address this foundational issue in a rather concrete way, taking advantage of the simplicity of the linear model to bring to the fore a different set of intuitions about parameter estimation. We begin by making the linearity assumption, but then let geometric rather than probabilistic intuitions be our guide. In particular, we view each data point as imposing a linear constraint on the parameters and treat parameter estimation as a (deterministic) constraint satisfaction problem. We focus on obtaining algorithms that solve this constraint satisfaction problem, exploiting the geometric framework to analyze the convergence of these algorithms.

The emphasis on constraint satisfaction algorithms in the current chapter has the advantage of focusing attention on some computational issues that are important in practice and were glossed over in our purely statistical discussion in Chapter 5. In particular, we will introduce the distinction between “batch” and “on-line” algorithms, a distinction which is of importance in real-time applications of statistical modeling and in situations involving large data sets.

At the end of the chapter, we return to the probabilistic perspective, showing that there is a natural correspondence between the (Euclidean) geometry underlying the constraint satisfaction formulation and the statistical assumptions alluded to above. Thus, we can view the excursion into geometry as providing support for the statistical perspective in Chapter 5; thus encouraged, we will be less bashful about bringing probabilistic machinery to bear at the outset in future chapters. At

the same time, we will continue to seek external support for probabilistic assumptions, particularly when they shed light on computational concerns.

6.1 Batch and on-line algorithms

Let us consider in some more detail how data points may be presented to the learner. We wish to distinguish two basic situations—the setting of “batch” presentation, in which data are available as a block, and the “on-line” setting in which data arrive sequentially. Both settings arise naturally in practice: In many problems it is necessary to respond in real time, and on-line methods are dictated; in other situations our only interest is in a final answer—the best answer that we can obtain given a certain data-gathering budget—and in such cases batch methods are natural.

On the other hand, we are often free to take either the batch or the on-line point of view on a learning problem—a sequential data stream can be stored for subsequent analysis as a block, and a block of data can be accessed sequentially. Moreover, a theoretical understanding of algorithms for parameter estimation is enhanced by approaching the problem from both points of view. We will see that the on-line point of view yields simple, intuitive algorithms, but a full analytical understand of on-line algorithms can be difficult, and we therefore turn to a related batch analysis to enhance understanding. On the other hand, batch methods are often usefully understood by taking an on-line point of view—in particular, large-scale batch problems generally require iterative algorithms that sweep repeatedly through the data. These sweeps can often be usefully analyzed as on-line algorithms.

A great deal of insight can be obtained by considering the elemental problem of updating the parameters of a linear model based on the presentation of a single data point. Let us begin with a discussion of the geometry underlying this problem, and show how simple geometric intuition leads us to an on-line algorithm known as the *LMS algorithm*. The acronym “LMS” refers to “least mean squares,” which, as we shall see, reflects the fact that the algorithm can be viewed as an optimization or constraint satisfaction procedure.

6.2 The LMS algorithm

Let us begin with a minimum of probabilistic pretension and consider the core of the linear model—the linear dependence of one variable on another. We consider the following question: Suppose that we have a pair of observed variables x_n and y_n that we assume are related linearly. What should a learning algorithm do when presented with a data point consisting of the pair (x_n, y_n) ? We shall be very naive and see if we can get any clues as to how to design a learning algorithm by considering the vector space geometry that characterizes the model.

We wish to express y_n as a linear function of x_n :

$$y_n = \theta^T x_n + \epsilon_n, \quad (6.1)$$

where θ is a parameter vector. Let us view ϵ_n as a deterministic “error term” whose presence in Eq. (6.1) is an admission that we don’t necessarily expect to be able to express y_n perfectly as a

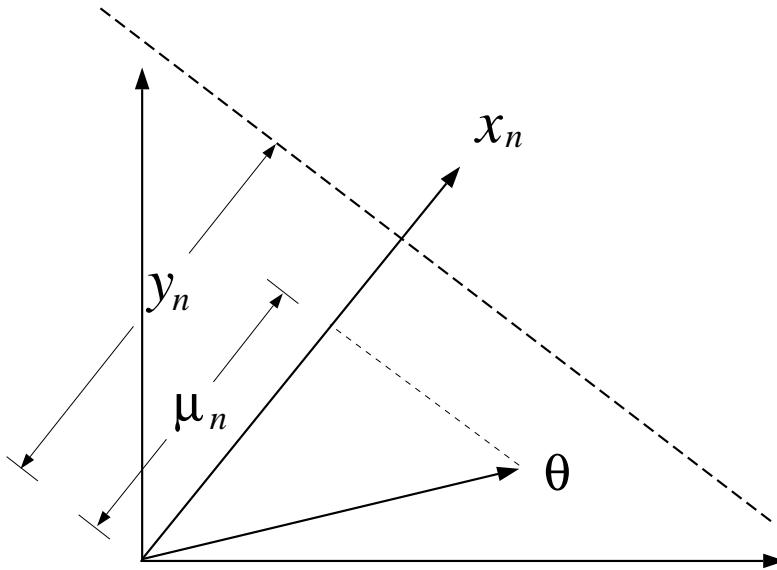


Figure 6.1: The geometry associated with the LMS algorithm. The figure shows the projection μ_n of the parameter vector θ on the input vector x_n . Also shown is the output value y_n as a distance along the x_n vector. The dashed line is the set of all vectors θ that have a projection of y_n and thus are solutions. The error associated with θ is the distance $(y_n - \mu_n)$. Changing θ by the vector $(y_n - \mu_n)x_n$ thus yields a solution vector.

linear function of x_n . In particular, let us forgo endowing ϵ_n or any of the other terms in Eq. (6.1) with probability distributions.

Figure 6.1 displays the vectors x_n and θ as well as the projection of θ on x_n , whose value we denote by μ_n . The projection μ_n is the inner product $\theta^T x_n$ divided by the norm of x_n .¹ Let us suppose for simplicity (temporarily) that x_n has norm one, so that the inner product $\theta^T x_n$ is the same as the projection μ_n . Now consider the problem of finding a vector θ that maps x_n to y_n exactly; that is, a vector such that ϵ_n is zero. Clearly we require a vector θ whose projection onto x_n is equal to y_n , and as the figure shows, there is a line of possible solutions that is orthogonal to x_n . Any vector along this line projects to the desired value y_n . We can view the data point (x_n, y_n) as imposing a constraint upon the vector θ that it lie along this line.

How might we design a learning algorithm to update the current value of the parameter vector

¹Recall the fundamental relationship:

$$\cos \alpha = \frac{\theta^T x_n}{\|\theta\| \|x_n\|}, \quad (6.2)$$

where α is the angle between θ and x_n . From this relationship we obtain

$$\mu_n = \frac{\theta^T x_n}{\|x_n\|}. \quad (6.3)$$

for the projection $\mu_n = \|\theta\| \cos \alpha$.

θ such that the new value of θ meets the constraint and lies on the solution line? Although there are an infinity of possible directions that we could move, note that there are two *natural* directions available to us: the direction associated with x_n and that associated with θ . Let us be very naive and decide that we should choose one of these two directions as the direction in which to update θ .

Although it is possible to figure out how far to move along the θ direction so as to intersect the solution line, it is rather easier to figure out how far to move along the x_n direction, given the orthogonality of x_n and the solution line. Let us opt for simplicity and choose to follow x_n .

Given that x_n is a unit vector, and given that the error we incur using the current parameter vector is the difference $(y_n - \theta^T x_n)$, it is clear that we should move the parameter vector a distance $(y_n - \theta^T x_n)$ in the x_n direction. Thus:

$$\theta^{(t+1)} = \theta^{(t)} + (y_n - \theta^{(t)T} x_n) x_n, \quad (6.4)$$

where $\theta^{(t)}$ is the estimated value of θ at the t th step of the algorithm. This algorithm jumps to the solution line.

If the vector x_n is not a unit vector, then we need to scale all of our distances by the norm of x_n . The projection is now $\theta^T x_n / \|x_n\|$ and thus we need to choose a parameter vector whose projection onto x_n is $y_n / \|x_n\|$. This implies that the error we incur using θ is given by $(y_n - \theta^T x_n) / \|x_n\|$, which is the amount we need to move in the direction of x_n . The unit vector in this direction is given by $x_n / \|x_n\|$; thus we obtain the following learning algorithm:

$$\theta^{(t+1)} = \theta^{(t)} + \frac{1}{\|x_n\|^2} (y_n - \theta^{(t)T} x_n) x_n, \quad (6.5)$$

which again hops to the solution line in a single step.

More generally, we express our learning algorithm in the following form:

$$\theta^{(t+1)} = \theta^{(t)} + \rho (y_n - \theta^{(t)T} x_n) x_n, \quad (6.6)$$

where ρ is a free parameter known as the “step size.” Our analysis has shown that the choice $\rho = 1 / \|x_n\|^2$ yields an algorithm that hops to the solution line in a single step. It is also easy to see that if $0 < \rho < 2 / \|x_n\|^2$, then on repeated presentations of x_n the algorithm will converge to the solution line asymptotically.

The algorithm in Equation 6.6 is the LMS algorithm.

6.2.1 Multiple data points

Let us now consider the case in which multiple data points are available. In particular we suppose that we have a “training set” $\mathcal{X} = \{(x_n, y_n)\}_{n=1}^N$, where N , the number of data points, is at least as large as k , the dimensionality of the parameter vector.

Let us begin by considering the simplest case, in which $N = k$; let us also assume for simplicity that the vectors x_n are linearly independent. Under these conditions, the model given by Equation 6.1 imposes a set of k linearly independent equations on k unknowns. This implies the existence of a unique parameter vector θ that achieves $\epsilon_n = 0$ for each n . Will the LMS algorithm find this solution?

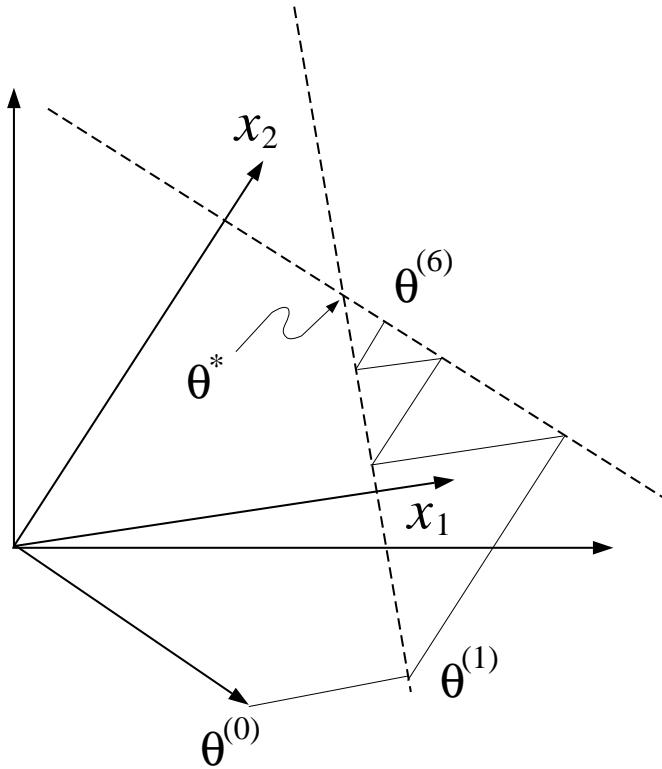


Figure 6.2: The geometry associated with the LMS algorithm in the case of two input vectors x_1 and x_2 . Associated with each vector is a line of solutions and the intersection of these lines is the vector θ^* that solves the problem for both vectors. We show the path taken by the LMS algorithm upon repeated presentations of x_1 and x_2 .

Figure 6.2 presents an example for the case of $N = 2$. As shown in the figure, each of the directions determined by the vectors x_1 and x_2 is associated with a solution line of vectors θ that map the given x_i to the corresponding y_i . The value θ^* that maps both vectors x_n to their desired values lies at the intersection of these two lines. Assuming that the training regime alternates between the two data points, we see that the LMS algorithm takes a zigzag path, following first the x_1 direction and then the x_2 direction. It seems clear, and it is in fact true (as we will show), that there exists a maximum value of the step size for which the algorithm converges to the solution.

If we turn to the case of $N > k$, we expect to see a qualitatively similar behavior in which LMS takes a zigzag path through the parameter space. In this case, however, we have an overdetermined set of equations and the lines that achieve ϵ_n for the various data points do not meet at a single point (see Figure 6.3). Given that LMS always moves from the current θ towards the line of solutions corresponding to the current data point, we see that we cannot expect the algorithm to move to a single point and stay put. We do expect, however, that the LMS algorithm should “converge” towards a small region of the parameter space, given an appropriate choice of the step size. Making further progress on these issues requires us to characterize more formally the constraint satisfaction

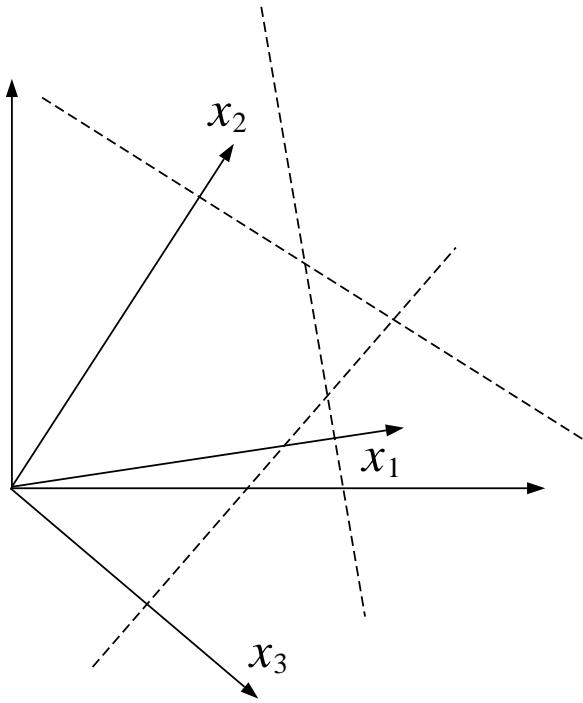


Figure 6.3: The geometry associated with the LMS algorithm in the case of three input vectors x_1 , x_2 and x_3 . Associated with each vector is a line of solutions. In general these lines do not intersect.

problem underlying the algorithm.

6.3 The sum of squares cost function and the normal equations

The approach that we pursue—which dates back to Gauss if not before—is to search for parameter vectors θ that yield “small” values of ϵ_n . We need to characterize what we mean by “small,” and decide how to combine the errors for different values of n . To make these decisions we again reason geometrically. We now work in a different geometry, however, namely an N -dimensional vector space, where N is the number of data points.

Let y denote a column vector with components y_n and let \hat{y} denote a column vector with components $\hat{y}_n = \theta^T x_n$. These are vectors in an N -dimensional vector space. We want to express the relationship between these vectors in a way that reveals more of the geometry behind the linear model. To do so, let X represent the matrix whose n th row is the row vector x_n^T . We write:

$$\hat{y} = X\theta, \quad (6.7)$$

showing that \hat{y} lies in the column space of the matrix X . Each column of X corresponds to a particular component of the vector x_n , and the set of columns of X can be viewed as spanning a vector subspace (see Figure 6.4). The vector \hat{y} lies in this vector subspace. The vector y , on the

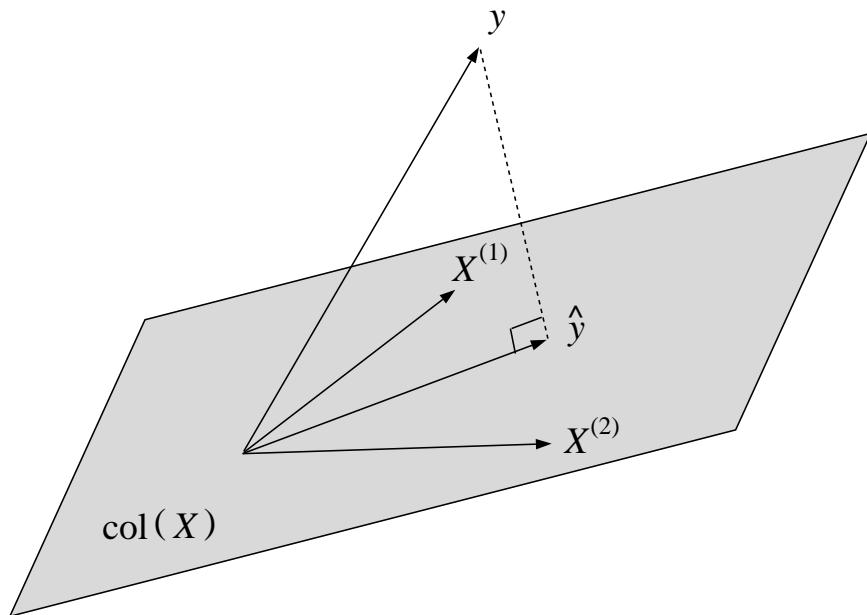


Figure 6.4: A geometric perspective on the linear regression problem for the case of the matrix X having two columns. Let $X^{(1)}$ represent the first column of X and let $X^{(2)}$ represent the second column. Denote the column space of X as $\text{col}(X)$. The approximating vector \hat{y} lies in this vector subspace, while the data vector y generally lies outside of this subspace. We wish to find a vector \hat{y} that is the orthogonal projection of y on $\text{col}(X)$.

other hand, generally lies outside of this vector subspace, reflecting the fact that in general the errors ϵ_n cannot simultaneously be zero.

Our problem reduces to choosing a vector \hat{y} in a vector subspace that best represents a vector y outside of the subspace. A natural solution, from a geometric point of view, is to choose the *orthogonal projection* of y onto the subspace. We will solve the problem of finding a vector θ^* that yields this projection in three different ways.

Our first solution appeals directly to the geometry in Figure 6.4. In particular, for \hat{y} to be the orthogonal projection of y on the column space of X , the difference vector $\epsilon = y - \hat{y}$ must be orthogonal to this vector subspace. Thus $y - \hat{y} = y - X\theta^*$ must be orthogonal to the columns of X , or, equivalently, orthogonal to the rows of X^T . This yields:

$$X^T(y - X\theta^*) = 0 \quad (6.8)$$

which implies

$$X^T X \theta^* = X^T y. \quad (6.9)$$

These equations, which characterize an optimizing vector θ^* , are referred to as the *normal equations*.

There is an equivalent characterization of the orthogonal projection in terms of minimal Euclidean length; this characterization leads us to a calculus-based derivation of the normal equations. In particular, let us choose \hat{y} such that the error vector $\epsilon = y - \hat{y}$ has minimal Euclidean length. Thus, working (equivalently) with the squared length, we wish to minimize the *least squares cost function* $J(\theta)$:

$$J(\theta) \triangleq \frac{1}{2} \sum_{n=1}^N \epsilon_n^2 = \frac{1}{2} \sum_{n=1}^N (y_n - \theta^T x_n)^2, \quad (6.10)$$

with respect to θ .²

Differentiating J with respect to the i th component, θ^i , of the vector θ , we obtain:

$$\frac{\partial J}{\partial \theta^i} = - \sum_{n=1}^N (y_n - \theta^T x_n) x_n^i, \quad (6.11)$$

where x_n^i is the i th component of the vector x_n . Collecting these partial derivatives into a vector we obtain the following gradient:

$$\nabla_{\theta} J = - \sum_{n=1}^N (y_n - \theta^T x_n) x_n, \quad (6.12)$$

which we must set to zero to obtain conditions on the optimizing solution θ^* .

To obtain an explicit solution it is useful to make use of the matrix X and write the gradient as a single matrix equation. Recalling that X has the vectors x_n on its rows, we can view Eq. (6.12) as a sum of the rows of X , weighted by the values $(y_n - \theta^T x_n)$. Equivalently this sum is the sum

²The factor of $1/2$ is included for convenience; it cancels the factor arising from the exponent of 2 when we take derivatives.

of the columns of X^T . Recalling that the values $\theta^T x_n$ are the components of the vector $\hat{y} = X\theta$, we have:

$$\nabla_{\theta} J = -X^T(y - X\theta). \quad (6.13)$$

Finally, setting to zero we obtain:

$$X^T(y - X\theta^*) = 0, \quad (6.14)$$

or equivalently:

$$X^T X \theta^* = X^T y, \quad (6.15)$$

which are the normal equations.

In Appendix XXX, we provide a short review of matrix and vector derivatives, which allows the reader to go directly from the cost function expressed in vector notation as:

$$J(\theta) = \frac{1}{2}(y - X\theta)^T(y - X\theta) \quad (6.16)$$

$$= \frac{1}{2}(y^T y - 2y^T X\theta + \theta^T X^T X\theta), \quad (6.17)$$

directly to the gradient:

$$\nabla_{\theta} J = -X^T(y - X\theta), \quad (6.18)$$

from which we again obtain the normal equations by setting to zero.

In most situations of practical interest, the number of data points N is larger than the dimensionality k of the input space and the matrix X has full column rank. If this condition holds, then it is easy to verify that $X^T X$ is necessarily invertible and thus we can express θ^* explicitly as follows:

$$\theta^* = (X^T X)^{-1} X^T y. \quad (6.19)$$

Moreover, if we take a second derivatives of J with respect to θ we find that the Hessian matrix of J is given by $X^T X$ (see Appendix XXX). The assumption that $X^T X$ is invertible implies that $X^T X$ is positive definite, and thus the critical point that we have found is a minimum. The solution to the normal equations provides the unique solution to the constraint satisfaction problem.

In Section ?? we discuss the case in which X has less than full column rank, and develop a *regularization* method to handle this case.

In the setting of “batch” presentation of data, in which data are available as a block, we can form the matrix X and the vector y and solve the normal equations. There are two major classes of methods for solving these equations: *direct methods* and *iterative methods*. The former class of methods, of which Gaussian elimination and QR decomposition are classical examples, converge in a finite number of steps. Iterative methods, which converge in a limiting sense, are of interest in the setting of particularly large problems, where direct approaches can be infeasible computationally.

Our next task is to try to understand the link between the two geometries that we have studied—the k -dimensional geometry of Figure 6.1 and the N -dimensional geometry of Figure 6.4. We also want to understand the relationship between the normal equations and the LMS algorithm. In the following section, we forge these links via the derivation of a steepest descent algorithm for solving the normal equations. This algorithm can be viewed as an example—one of many—of an iterative

method for the batch case. Our goal, however, is not to explore iterative solution methods for the batch case (indeed there are more sophisticated methods than steepest descent). Rather, we wish to use the normal equations and their solution via steepest descent as a point of departure for understanding the on-line case.

6.4 Steepest descent and the LMS algorithm

Following the negative of the gradient in Eq. (6.12) we obtain the following *steepest descent* algorithm:

$$\theta^{(t+1)} = \theta^{(t)} + \rho \sum_{n=1}^N (y_n - \theta^{(t)T} x_n) x_n, \quad (6.20)$$

where $\theta^{(t)}$ is the parameter vector at the t th step of the iteration and where ρ is the step size. The algorithm is initialized at an arbitrary vector $\theta^{(0)}$ and iterates until a convergence criterion is met.

The steepest descent algorithm involves a sum over all N input vectors, thus the algorithm is a batch algorithm. Note that this aspect of the algorithm can render it rather inefficient, and this inefficiency motivates us to consider on-line approaches. In particular, if N is large, say in the millions, then the algorithm can spend an inordinate amount of time passing through the training set in order to compute the gradient, at which point it takes a step in the parameter space. Given that one of the motivations for studying iterative algorithms is to be able to handle very large problems, this feature of steepest descent is disconcerting. Note moreover that if the data set is redundant—a common occurrence with large data sets—then it might not be necessary to sum all of the N terms in Eq. (6.20) to obtain an accurate estimate of the direction of the gradient (the magnitude of the gradient is irrelevant because it is being scaled by a constant ρ that is under our control). In such situations, algorithms that take a sum over a subset of the data—a “mini-batch”—can often be significantly more efficient than the full batch algorithm. Indeed, in the limiting case we can view a single term, $-(y_n - \theta^{(t)T} x_n) x_n$, as providing a rough estimate of the direction of the gradient. It may be advantageous to go ahead and follow this rough estimate and make progress in the parameter space rather than waiting to obtain a better estimate. This logic leads to the following algorithm, which adjusts the parameter vector according to the estimated gradient based on a single data point:

$$\theta^{(t+1)} = \theta^{(t)} + \rho(y_n - \theta^{(t)T} x_n) x_n. \quad (6.21)$$

This is of course the LMS algorithm. We see that the LMS algorithm can be viewed as an approximation to the steepest descent algorithm, where the approximation involves replacing the sum obtained in the batch algorithm with a single term. Such an approximation is referred to as a “stochastic gradient” algorithm, where “stochastic” refers to an assumption that the choice of data point (x_n, y_n) is made according to a stochastic process.

Let us emphasize that although LMS can be viewed as an approximation to steepest descent, it is often a much superior algorithm. Because it requires significantly less work per parameter update, it can converge significantly faster than steepest descent.

We are now in a position to learn something more about the convergence of the LMS algorithm. From the normal equations we have a characterization of the vector toward which we expect the LMS algorithm to tend, and from the steepest descent equations we have the possibility of characterizing the path that LMS will be expected to follow on average (under an appropriate stochastic analysis). In particular we may hope to learn something about the maximum possible value of ρ .

We present two analyses—one algebraic and one geometric—that yield the sought-after results. Both analyses involve analyzing the shape of the quadratic cost function J in the neighborhood of its minimum.

6.4.1 An algebraic convergence analysis³

One way to understand the convergence of the steepest descent algorithm in Eq. (6.20) is to unfold the recursion and solve the resulting equation.

In particular, letting $\theta^{(t)}$ represent the parameter vector at the t th iteration of the algorithm, we have:

$$\theta^{(t+1)} = \theta^{(t)} + \rho \sum_{n=1}^N (y_n - \theta^{(t)T} x_n) x_n \quad (6.22)$$

$$= \theta^{(t)} + \rho \sum_{n=1}^N x_n y_n - \rho \sum_{n=1}^N (x_n x_n^T) \theta^{(t)} \quad (6.23)$$

$$= \theta^{(t)} + \rho X^T y - \rho X^T X \theta^{(t)} \quad (6.24)$$

$$= (I - \rho X^T X) \theta^{(t)} + \rho X^T y. \quad (6.25)$$

Expanding the recursion, we have:

$$\theta^{(t+1)} = (I - \rho X^T X) \theta^{(t)} + \rho X^T y \quad (6.26)$$

$$= (I - \rho X^T X) [(I - \rho X^T X) \theta^{(t-1)} + \rho X^T y] + \rho X^T y \quad (6.27)$$

$$= (I - \rho X^T X)^{t+1} \theta^{(0)} + \rho \sum_{i=0}^t (I - \rho X^T X)^i X^T y. \quad (6.28)$$

We now let t go to infinity. Let us assume for now that the first term goes to zero as t goes to infinity—we will then return to this term and derive a condition that ensures that it goes to zero. Thus we have:

$$\theta^{(\infty)} = \rho \sum_{i=0}^{\infty} (I - \rho X^T X)^i X^T y \quad (6.29)$$

$$= \rho (\rho X^T X)^{-1} X^T y \quad (6.30)$$

$$= (X^T X)^{-1} X^T y, \quad (6.31)$$

³The material in this section is optional; it will not be needed in later chapters.

which are the normal equations. We have thus shown that the steepest descent algorithm converges to the minimum of the cost function, under the assumption that the first term in Eq. (6.28) converges to zero.

Let us now consider the matrix power $(I - \rho X^T X)^{t+1}$ as $t \rightarrow \infty$. In general, to show that a matrix power converges to zero we need to show that its largest eigenvalue is less than one in absolute value. Now it is easy to verify that if λ is an eigenvalue of $(I - B)$ for a matrix B , then $1 - \lambda$ is an eigenvalue of B . Thus the absolute values of the eigenvalues of $(I - B)$ are less than one if and only if the absolute values of the eigenvalues of B are between zero and two. Thus we have the condition:

$$0 < \lambda_{\max}[\rho X^T X] < 2, \quad (6.32)$$

where λ_{\max} represents the maximum eigenvalue of a matrix, or equivalently:

$$0 < \rho < 2/\lambda_{\max}[X^T X]. \quad (6.33)$$

This is the condition for convergence; the step size ρ can be no larger than two divided by the maximum eigenvalue of $X^T X$.

6.4.2 A geometric convergence analysis⁴

To get a better understanding of the convergence condition that we have just derived, let us rederive it from a geometric point of view.

Our cost function is a quadratic function in the components θ^i and can be plotted as a set of elliptical contours in the parameter space. In particular, for the example shown earlier in Figure 6.2, the corresponding contours are shown in Figure 6.5. Let us take a moment to understand how to obtain these contours.

We know that the minimum of the cost function is achieved by the vector θ^* that solves the normal equations. Our analysis will be simplified if we choose this optimizing point as the origin of our coordinate system. We choose new coordinates $\phi = \theta - \theta^*$ and express the cost function in these new coordinates:

$$\begin{aligned} J(\phi) &= \frac{1}{2} \sum_{n=1}^N (y_n - \theta^T x_n)^2 \\ &= \frac{1}{2} (y - X\theta)^T (y - X\theta) \\ &= \frac{1}{2} (y - X(\phi + \theta^*))^T (y - X(\phi + \theta^*)) \\ &= \frac{1}{2} (y^T y - \theta^{*T} X^T y + \phi^T X^T X \phi), \end{aligned}$$

where in passing from the third line to the fourth line we have expanded the quadratic expression and used the fact that θ^* solves the normal equations. In the new coordinates we see that the cost function is expressed simply as:

$$J(\phi) = C + \frac{1}{2} \phi^T X^T X \phi, \quad (6.34)$$

⁴The material in this section is optional; it will not be needed in later chapters.

Figure 6.5: The contours of the cost function $J(\theta)$ for the example in Figure 6.2.

where $C = y^T y - \theta^{*T} X^T y$ is a constant.

We now rotate the coordinate system so that the axes point along the major and minor axes of the ellipse. This is achieved by making use of the eigenvectors of $X^T X$. In particular, let A be the matrix whose column vectors are the eigenvectors of $X^T X$. We have:

$$X^T X = A \Lambda A^T, \quad (6.35)$$

where Λ is a diagonal matrix whose elements are the eigenvalues λ_i of $X^T X$. Note also that the fact that $X^T X$ is a symmetric matrix implies that A is orthogonal. Thus we have:

$$A^T X^T X A = \Lambda. \quad (6.36)$$

Now choose new coordinates $\psi = A^T \phi$. We obtain:

$$J(\psi) = C + \frac{1}{2} (A\psi)^T X^T X (A\psi) \quad (6.37)$$

$$= C + \frac{1}{2} \psi^T A^T X^T X A \psi \quad (6.38)$$

$$= C + \frac{1}{2} \psi^T \Lambda \psi. \quad (6.39)$$

This final equation is simply the weighted sum of squares of the components of ψ , with weights given by the eigenvalues λ_i . Setting $J(\psi)$ equal to a constant yields the equation of an ellipsoid.

Let us now express the steepest descent equation in the new coordinates. We write the equation in matrix notation (cf. Eq. (6.24)) as:

$$\theta^{(t+1)} = \theta^{(t)} + \rho(X^T y - X^T X \theta^{(t)}) \quad (6.40)$$

Given that the θ coordinates and the ψ coordinates are related via $\theta = A\psi + \theta^*$, we obtain:

$$A\psi^{(t+1)} = A\psi^{(t)} - \rho(X^T X A\psi^{(t)}), \quad (6.41)$$

where we have used the fact that θ^* solves the normal equations. Premultiplying both sides of this equation by A^T (recalling that A is orthogonal), we obtain:

$$\psi^{(t+1)} = \psi^{(t)} - \rho(A^T X^T X A\psi^{(t)}) \quad (6.42)$$

$$= \psi^{(t)} - \rho\Lambda\psi^{(t)}. \quad (6.43)$$

This equation represents a decoupled set of equations in the components of the ψ vector:

$$\psi^{i(t+1)} = (1 - \rho\lambda_i)\psi^{i(t)}, \quad (6.44)$$

which converges if $(1 - \rho\lambda_i)$ is less than one in absolute value. That is, we require:

$$\|1 - \rho\lambda_i\| < 1, \quad (6.45)$$

which is equivalent to:

$$0 < \rho < 2/\lambda_i. \quad (6.46)$$

Given that this must be true for all λ_i we have recovered the same condition for convergence as obtained in the previous section (Eq. (6.33)).

In the decoupled coordinate system, we see that convergence condition amounts to the condition that if the algorithm hops from one side of an axis of the ellipsoid to the other, it must end up no further away from the axis than when it started. The axis associated with the maximum eigenvalue puts the strongest constraint on the step size.

6.4.3 LMS and stochastic approximation

It is beyond the scope of the book to provide a detailed consideration of the sense in which the LMS algorithm (and related “on-line” algorithms) converges to a solution, and we will content ourselves with providing pointers to the literature on stochastic approximation where such issues are addressed.⁵ To get some sense of the issues involved, however, note that the path taken by the LMS algorithm in the parameter space depends on the particular way in which the training set is ordered. There are many kinds of ordering that may arise practice; typical examples include: (1) the algorithm passes through the training set in a fixed order, (2) varying orderings are used for each pass through the training set, and (3) data points are selected randomly with replacement from the training set. Moreover, (4) in other cases there is no “training set”; rather, the data points arrive as a potentially infinite stream. Another set of issues arises when one considers the meaning of “convergence.” If the step size ρ remains fixed then the algorithm “converges” only in a stochastic sense, and there are several kinds of stochastic convergence that one can consider. It is also possible to consider variants of LMS in which the step size decreases to zero; under certain conditions (certain rates of decrease of the step size) the algorithm can be shown to converge to a point. As should be clear, a full analysis of LMS is a subtle business, and fairly sophisticated mathematical tools are required to do justice to the problem.

⁵See the section on “Historical remarks and bibliography” at the end of the chapter.

6.5 Weighted least squares

In later chapters we will need to solve a generalization of least squares, in which each data point is accompanied by a “weight” w_n . Intuitively, large weights correspond to data points that are “important,” and small weights correspond to data points that are “unimportant.” Let us set up this *weighted least squares* problem and display the corresponding normal equations.

Consider a set of weights w_n for each $n = 1, \dots, N$. Let us incorporate these weights into the cost function as follows:

$$J(\theta) = \frac{1}{2} \sum_{n=1}^N w_n (y_n - \theta^T x_n)^2, \quad (6.47)$$

We can write this cost function in matrix form by defining a diagonal matrix $W \triangleq \text{diag}(w_1, w_2, \dots, w_N)$ and writing:

$$J(\theta) = \frac{1}{2} (y - X\theta)^T W (y - X\theta), \quad (6.48)$$

where we see that the weight matrix W can be viewed as defining a new metric with which to measure errors.

To obtain a solution θ^* , we take the gradient of Eq. (6.48):

$$\nabla_{\theta} J = -X^T W y - X^T W X \theta. \quad (6.49)$$

and set to zero:

$$X^T W X \theta^* = X^T W y, \quad (6.50)$$

These equations are the normal equations for weighted least squares.

6.6 Probabilistic interpretation

Thus far we have avoided making any probabilistic interpretation of the linear model and the least squares cost function. Let us now return to the statistical framework of linear regression in Chapter 5 and endow the terms in the linear model with probability distributions.

In Chapter 5 we augmented the linearity assumption with the assumption that the errors ϵ_n are Gaussian random variables having zero mean and variance σ^2 . This assumption implies that the conditional probability of y_n given x_n is Gaussian with mean $\theta^T x_n$:

$$p(y_n|x_n, \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left\{ -\frac{1}{2\sigma^2} (y_n - \theta^T x_n)^2 \right\}. \quad (6.51)$$

We assumed moreover that the y_n are independent and identically distributed, conditional on x_n . Thus the joint conditional distribution of the data y is obtained by taking the product of the individual conditional probabilities:

$$p(y|x, \theta) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp \left\{ -\frac{1}{2\sigma^2} \sum_{n=1}^N (y_n - \theta^T x_n)^2 \right\}. \quad (6.52)$$

Taking the logarithm and dropping the terms that do not depend on the parameter θ , we obtain the following expression for the log likelihood:

$$l(\theta; x, y) = -\frac{1}{2\sigma^2} \sum_{n=1}^N (y_n - \theta^T x_n)^2. \quad (6.53)$$

This log likelihood is equivalent to the least-squares cost function $J(\theta)$ in Eq. (6.10). In particular, maximizing the log likelihood with respect to θ is equivalent to minimizing the least-squares cost function.

What we have shown is that the assumptions of a Gaussian distribution and IID sampling imply—within a maximum likelihood framework—the minimization of the least-squares cost function. Moreover, the normal equations characterize the maximum likelihood solution to the linear regression problem.

We can view this result as providing support for the likelihood-based approach to parameter estimation. In particular, in imposing probabilistic assumptions on the linear model so as to obtain a likelihood function, we have imposed neither more nor less constraint on the problem than is required to obtain a well-posed deterministic problem in the constraint satisfaction formulation. In particular, in the latter formulation, we need to decide how to measure the magnitudes of the errors and how to combine these magnitudes. These decisions have correspondences in the probabilistic formulation, in particular the Gaussian assumption effectively determines the metric by which we measure the errors, and the IID assumption determines the way in which the errors are combined. Both formulations are useful. In particular the constraint satisfaction perspective has helped us to understand that the linear, IID, and Gaussian assumptions comprise a natural family, essentially reflecting a Euclidean geometry. The probabilistic perspective provides additional insight; in particular, a Gaussian distribution for the errors can be justified via the central limit theorem if it is the case that the error terms ϵ_n are decomposable into sums of many small random terms.

It is also worth noting that in the frequentist approach to estimation we are not restricted to likelihood-based methods. In particular, we can view the least-squares cost function as providing an “estimator” that can be evaluated with the usual frequentist criteria. That is, we can define the least-squares estimator of a parameter as a value that minimizes the least-squares cost function, whether or not the underlying probability model involves a Gaussian assumption. If the underlying model is Gaussian then the least-squares approach and maximum likelihood coincide, but in general they can be viewed as competitors. The fact that least-squares estimates involve the solutions of systems of linear equations is a computational argument in their favor.

Although the geometric perspective provides significant insight in the case of the linear model, the probabilistic, likelihood-based perspective becomes increasingly powerful when we consider various generalizations of the linear model. For example, discrete variables are naturally handled by likelihood-based methods, as are hybrid models that involve combinations of discrete and continuous variables. Moreover, latent variables allow us to build more complex error models and Markov chains allow us to move beyond the IID assumption. Likelihood-based methods will be our focus throughout the remainder of the book.

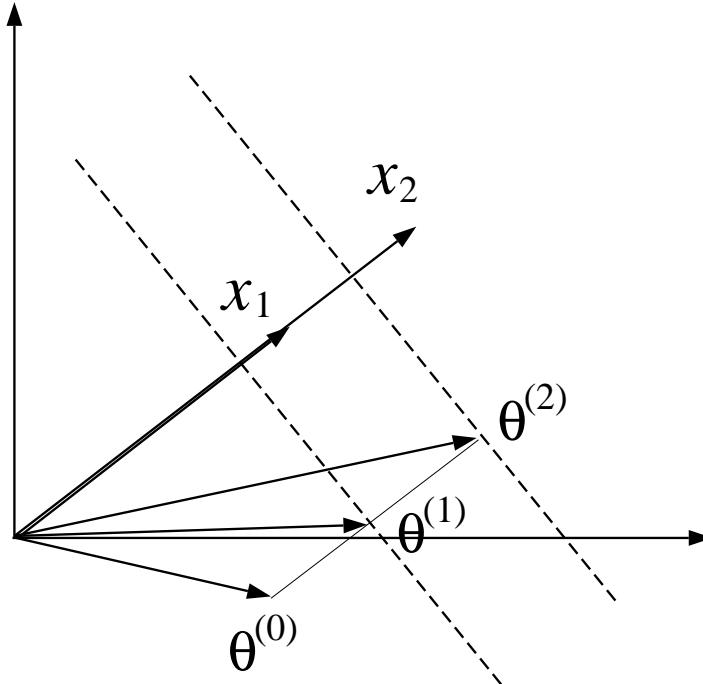


Figure 6.6: The geometry associated with the LMS algorithm in the case of two redundant input vectors x_1 and x_2 . The dashed lines represent lines of solutions corresponding to each of the input vectors. There is a line of least-squares solutions that lies halfway between these two lines. The component of the initial parameter vector $\theta^{(0)}$ that is orthogonal to these lines does not vanish as the algorithm iterates.

6.7 Ridge regression

6.8 Sequential Bayesian methods

6.9 Historical remarks and bibliography

Chapter 7

Linear classification

In this chapter we continue our discussion of elementary building blocks for graphical models, treating the case of a discrete node taking on a finite number of values. As in Chapter 6, our interest is in the conditional relationship between the node Y and a vector of explanatory variables X . We explore a number of possible representations for the conditional probability $p(y | x)$.

What form should our model of $p(y | x)$ take in the case of discrete Y ? If X is also discrete, then we might consider models in which all possible combinations of X and Y are represented in a table. We will indeed consider such a model in this chapter; however, it is important to keep in mind that the size of such a table is exponential in the number of components of X , and we would like to develop models to handle the (commonplace) situation in which this number is large. Moreover, we wish to develop tools that allow for continuous-valued X . In either case a natural first step is to try and mimic what we did with regression, exploiting the simplicity and mathematical convenience of linearity assumptions. It is unclear, however, how to represent the conditional expectation of Y —a number between zero and one for Bernoulli and multinomial variables—within the framework of a linear model. Some sort of nonlinearity seems to be needed, but which nonlinearity? Does introducing such a nonlinearity leave us with any role for linearity?

One way to help organize our thinking on these issues is to recall that we have already seen problems involving discrete Y in Chapter 5. In particular, in our discussion of classification models in that chapter, we found it useful to explore the relationship between two kinds of models: *discriminative models*—in which Y is the child of X —and *generative models*—in which Y is the parent of X . While the former approach represents $p(y | x)$ *explicitly*, the latter approach makes use of Bayes rule to represent the posterior probability $p(y | x)$ *implicitly*, in terms of the class-conditional probability $p(x | y)$ and the prior $p(y)$. Thus we can begin to get ideas for representations of $p(y | x)$ by studying generative models in which Y is a parent of X , and using Bayes rule to invert the model and thereby calculate the corresponding posterior probability $p(y | x)$. This approach will allow us to achieve some of the goals that we alluded to above—it will suggest a certain basic mathematical structure in which linearity plays a role, and it will cope with both discrete-valued and continuous-valued X . Moreover, it will suggest a natural “upgrade path” to more complex models.

In this chapter we retain our assumption from the previous chapter that both X and Y are

observed in our data set. We cast our presentation within the context of classification, where as before we refer to Y and X as the “class label” and the “feature vector,” respectively. We will fill in some of the details that were glossed over in Chapter 5 regarding the parameterization and estimation of generative and discriminative approaches to the classification problem. We present maximum likelihood methods for parameter estimation in both frameworks.

While we place our activity in this chapter within the framework of classification, it is worth noting that there are aspects of classification problems that fall beyond the scope of our discussion. In particular, our goal in this chapter is that of obtaining a model of the conditional probability $p(y|x)$. While $p(y|x)$ is a desirable quantity to model in a classification setting, it is also true that classification involves something more than evaluating a probability—in particular, classification involves making a *decision*. We can threshold the probability distribution $p(y|x)$ to obtain a decision, but this is only one possible way to use this probability; perhaps there are others. Indeed, perhaps there are some decisions which are in some sense more costly than others; our thresholding scheme should be sensitive to such costs. Moreover, we can imagine classification algorithms that do not make use of posterior probability $p(y|x)$ at all; rather they go directly from a data set to a decision rule. Evaluating these alternatives appropriately requires the mathematical framework of *decision theory*. In particular a decision-theoretic approach to classification allows us to specify costs associated with decisions and to evaluate alternative approaches to forming decision rules. We will return to these issues in Chapter 27, where we present a full treatment of decision theory in the graphical model setting. In that discussion we will in fact show that a reasonable first step in classification problems is to obtain a model of the conditional probability $p(y|x)$.

It is also worth noting that there is a flip side to this coin—there are problems other than classification problems for which the methods of this chapter are useful. In particular in Chapter 10 we discuss models that are structurally identical to the models in this chapter, but for which Y is no longer assumed to be observed; that is, for which Y is a latent variable. The results that we obtain here will play an important role in that chapter.

7.1 Linear regression and linear classification

A discrete-valued node can be viewed as a special case of a real-valued node, and this leads one to wonder why we need a separate treatment of discrete nodes. In particular, why not use the regression methods that we developed in Chapter 6 to solve classification problems?

To see some of the problems that arise if we pursue this approach, consider the simple case of a binary classification problem with a scalar-valued feature variable X . Let us represent the class label with a real-valued variable Y , with $Y = 0$ and $Y = 1$ representing the two classes. Figure 7.1 presents an example of such a problem, with the data pairs (x_n, y_n) represented as points in the plane. The linear regression fit to these data is also shown in the figure. Note that even though the data $\{y_n\}$ are restricted to the values zero and one, the fitted line is not restricted to these values. How are we to interpret this line? In Chapter 6 we showed that the linear regression fit is a conditional mean—the expected value of Y conditioned on the observed value of X . For an indicator random variable Y the expected value is the same as the probability that the variable takes on the value 1. The fact that the fitted line in Figure 7.1 strays outside of the range $(0, 1)$

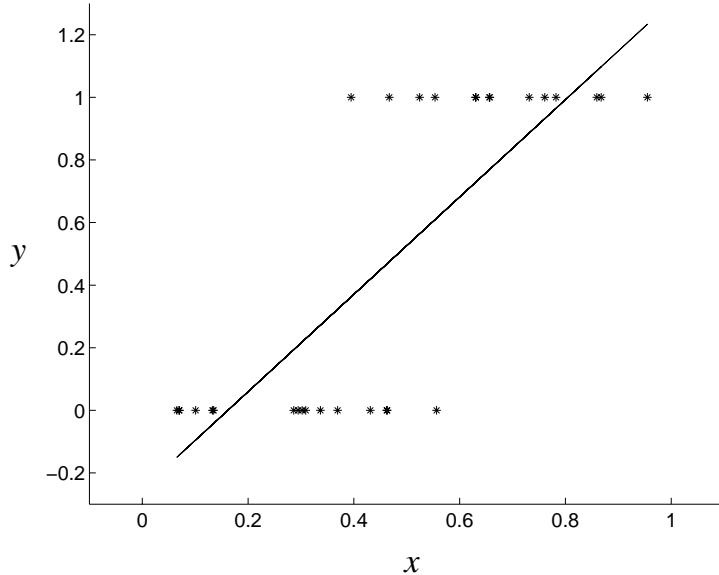


Figure 7.1: Data for a binary classification problem. The abscissa represents the one-dimensional feature vector x , and the ordinate represents the binary class label y , with 0 and 1 representing the two classes. Also shown is the least squares linear regression fit.

makes it difficult to sustain such an interpretation, however, in the setting of binary output data.

Even more serious problems arise when we consider in more detail how the regression fit depends on the data. Suppose in particular that we add the point $(1.5, 1)$ to the data set (see Figure 7.2). The earlier fit (Figure 7.1) yields a fitted value of 2.01 at $x = 1.5$, suggesting, under any reasonable interpretation of this value (e.g., thresholding), that the predicted class label at $x = 1.5$ should be 1. This correctly predicts the class of the new data point, suggesting that the parameters can already accommodate the new data point and need not be changed. Refitting the linear regression, however, changes the slope parameter from 1.55 to 1.23 and the intercept parameter from -0.32 to -0.17 (see Figure 7.2). Moreover, taking the value at which the fit equals 0.5 as the boundary between the two classes, this boundary changes significantly after the introduction of the new data point, leading to changes in the classification of some of the points near the boundary. If we add four additional data points at $x = 1.5$ the boundary moves even further, as shown in Figure 7.2. Given that these new data points are predicted correctly by the original fit, and are far from the boundary, this behavior is disconcerting.

The assumptions underlying linear regression are clearly not met in the classification setting; in particular, the assumption that the variable Y is Gaussian is clearly false. This mismatch between the assumptions and the data is responsible for the problems that we have identified. Once we have made probabilistic assumptions that are appropriate for the classification setting—in particular once we have discarded the Gaussian assumption—we will obtain classification models in which the fitted values behave in an intuitively reasonable manner.

As in Chapter 6 we focus on linear models throughout the current chapter. The notion of

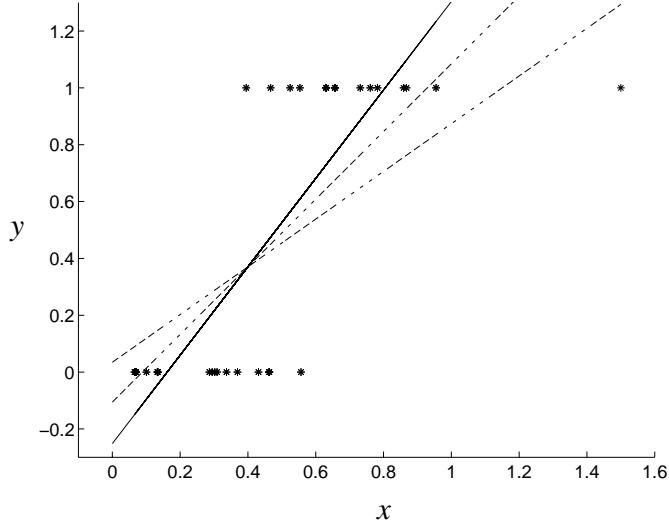


Figure 7.2: Three least squares regression fits. The solid line is the same fit as shown in Figure 7.1, the dash-dot line is the fit to the data with one additional point at $(1.5, 1)$, and the dashed line is the fit to the data with five additional points at $(1.5, 1)$.

“linearity” in the current chapter is, however, different from that in Chapter 6. We postpone the mathematical details until later sections, where in fact we will find that different classification models invoke the linearity assumption in somewhat different ways. All of the classification models that we study, however, can be viewed as providing a partitioning of the feature space into regions corresponding to the class labels. For linear models the boundaries between these regions are hyperplanes (see Figure 7.3).

7.2 Generative models

Figure 7.4 presents three graphical representations of generative classification models. In all three cases the class label node Y is the parent of the feature vector $X = (X_1, X_2, \dots, X_m)$. In Figure 7.4(a), the component features are treated as separate nodes; in this case, the children X_j are assumed to be conditionally independent given Y , as confirmed by the d-separation properties of the graph. This is a simplifying assumption that provides a starting point for our presentation and will be our focus through most of this section. In Figure 7.4(b), we have an alternative model in which the components of the feature vector are interdependent, with specific conditional independencies assumed to hold among specific sets of features. In this model general graphical model machinery must be invoked both to parameterize the class-conditional densities and to learn the values of the parameters. Accordingly we will not treat this model explicitly in this section but will return to it in later chapters once the appropriate machinery is in place. Finally, in Figure 7.4(c), we have a model in which no specific conditional independencies are assumed among the components of the

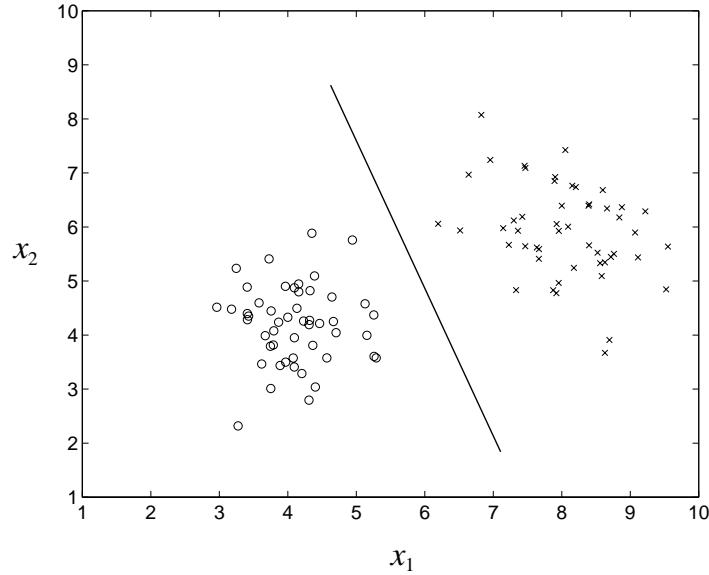


Figure 7.3: A binary classification problem in a two-dimensional feature space. The feature vectors in the training set are plotted as x's and o's for the two classes. Based on the training set, a classifier partitions the feature space into decision regions, one region for each class. In the case of a *linear classifier*, the boundaries between these regions are hyperplanes.

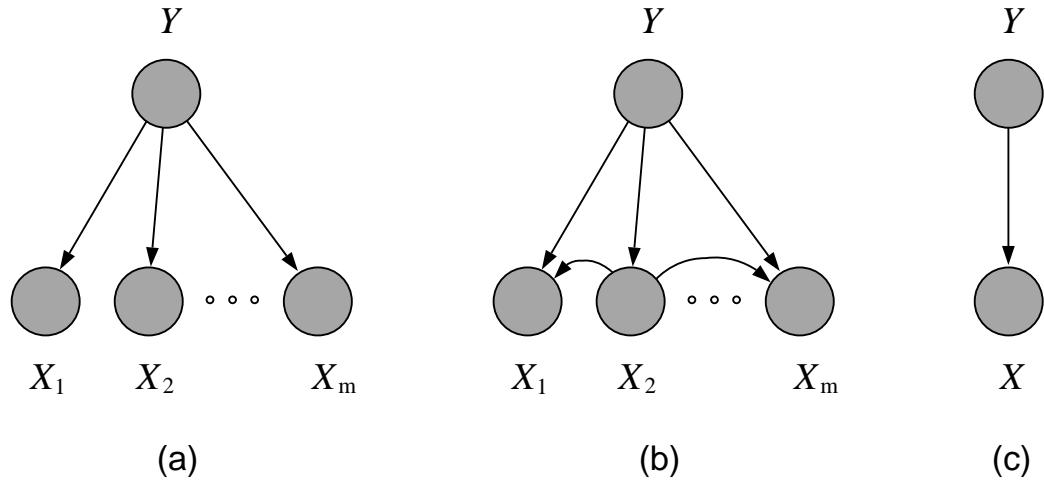


Figure 7.4: Three examples of generative classification models: (a) the case of conditionally independent features, (b) the case of dependent features with some conditional independence assumptions, and (c) the case of no conditional independence assumptions.

feature vector. In this case we represent the feature vector as a single node. This model, despite its simple graphical appearance, is the most general model of the three. We will discuss an example of this model in this section, where a Gaussian assumption for the class-conditional densities will allow us to obtain a simple model despite the absence of conditional independencies.

In all of the examples that we discuss, our goal is twofold: to describe the parametric representation of the posterior probability $p(y | x)$ for particular models, and to present maximum likelihood methods for estimating the parameters of the model from data.

7.2.1 Gaussian class-conditional densities

We begin by discussing the model in Figure 7.4(a) in the setting in which the features are continuous and endowed with Gaussian distributions. We initially treat the case of *binary classification*, in which the class label Y can take on one of two values. The extension to multiple classes is discussed in Section 7.2.1.

The model in Figure 7.4(a) requires a marginal probability for Y and a conditional probability for X given Y . Let $Y \in \{0, 1\}$ be a Bernoulli random variable with parameter π :

$$p(y | \pi) = \pi^y (1 - \pi)^{1-y}. \quad (7.1)$$

Given the conditional independence assumption expressed by the graph, the probability $p(x | y)$ factors into a product over conditional probabilities $p(x_j | y)$. For $Y = 0$, let each X_j have a Gaussian distribution:

$$p(x_j | Y = 0, \theta_j) = \frac{1}{(2\pi\sigma_j^2)^{1/2}} \exp \left\{ -\frac{1}{2\sigma_j^2} (x_j - \mu_{0j})^2 \right\}, \quad (7.2)$$

where μ_{0j} is the j th component of the mean vector for class $Y = 0$. For $Y = 1$ we have:

$$p(x_j | Y = 1, \theta_j) = \frac{1}{(2\pi\sigma_j^2)^{1/2}} \exp \left\{ -\frac{1}{2\sigma_j^2} (x_j - \mu_{1j})^2 \right\}. \quad (7.3)$$

Note that we use θ_j to denote all of the parameters for feature component x_j , including the means μ_{0j} and μ_{1j} , and the variance σ_j^2 . Note also that the variances σ_j^2 are allowed to vary across feature components x_j , but are assumed to be constant between the two classes.

Figure 7.5(a) presents an example of a contour plot of two Gaussians in a two-dimensional feature space for the case in which $\sigma_0^2 = \sigma_1^2$. An example in which the variances are unequal is shown in Figure 7.5(b).

The joint probability associated with the graph in Figure 7.4(a) is as follows:

$$p(x, y | \theta) = p(y | \pi) \prod_{j=1}^m p(x_j | y, \theta_j), \quad (7.4)$$

where $\theta = (\pi, \theta_1, \theta_2, \dots, \theta_m)$.

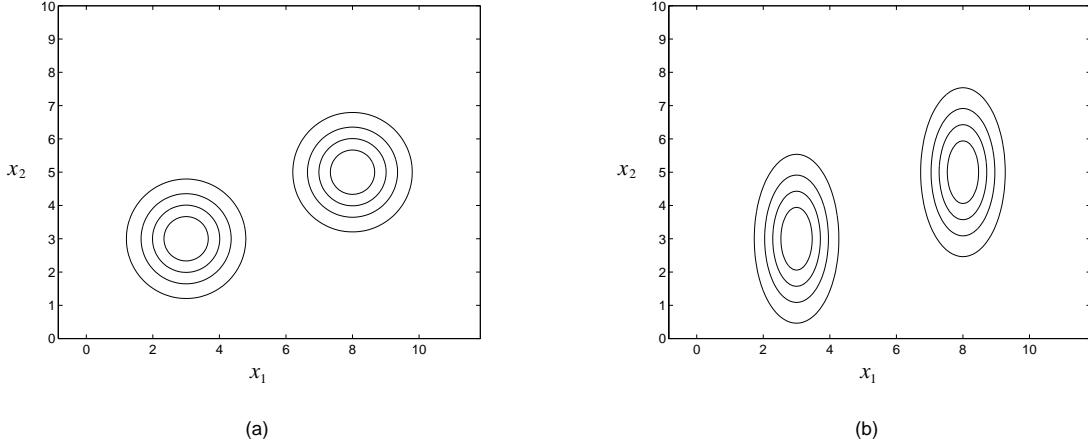


Figure 7.5: (a) A contour plot of Gaussian class-conditional densities for $\sigma_1 = 1$ and $\sigma_2 = 1$. (b) A contour plot for Gaussian class-conditional densities when $\sigma_1 = 0.5$ and $\sigma_2 = 2.0$.

Posterior probability

Let us calculate the posterior probability $p(Y = 1 | x, \theta)$. The algebra is somewhat simplified if we work with matrix notation. Thus let:

$$p(x | y = k, \theta) = \frac{1}{(2\pi)^{1/2} |\Sigma|^{1/2}} \exp \left\{ -\frac{1}{2} (x - \mu_k)^T \Sigma^{-1} (x - \mu_k) \right\}, \quad (7.5)$$

for each of the two classes $k \in \{0, 1\}$, where $\mu_k \triangleq (\mu_{k1}, \mu_{k2}, \dots, \mu_{km})^T$ is the vector of means for the k th Gaussian, and where $\Sigma \triangleq \text{diag}(\sigma_1^2, \sigma_2^2, \dots, \sigma_m^2)$ is a diagonal covariance matrix. We have:

$$\begin{aligned} p(Y = 1 | x, \theta) &= \frac{p(x | Y = 1, \theta)p(Y = 1 | \pi)}{p(x | Y = 1, \theta)p(Y = 1 | \pi) + p(x | Y = 0, \theta)p(Y = 0 | \pi)} \\ &= \frac{\pi \exp\{-\frac{1}{2}(x - \mu_1)^T \Sigma^{-1} (x - \mu_1)\}}{\pi \exp\{-\frac{1}{2}(x - \mu_1)^T \Sigma^{-1} (x - \mu_1)\} + (1 - \pi) \exp\{-\frac{1}{2}(x - \mu_0)^T \Sigma^{-1} (x - \mu_0)\}} \\ &= \frac{1}{1 + \exp\{-\log \frac{\pi}{1-\pi} + \frac{1}{2}(x - \mu_1)^T \Sigma^{-1} (x - \mu_1) - \frac{1}{2}(x - \mu_0)^T \Sigma^{-1} (x - \mu_0)\}} \\ &= \frac{1}{1 + \exp\{-(\mu_1 - \mu_0)^T \Sigma^{-1} x + \frac{1}{2}(\mu_1 - \mu_0)^T \Sigma^{-1} (\mu_1 + \mu_0) - \log \frac{\pi}{1-\pi}\}} \end{aligned} \quad (7.6)$$

$$= \frac{1}{1 + \exp\{-\beta^T x - \gamma\}} \quad (7.7)$$

where the final equation defines parameters β and γ :

$$\beta \triangleq \Sigma^{-1}(\mu_1 - \mu_0) \quad \gamma \triangleq -\frac{1}{2}(\mu_1 - \mu_0)^T \Sigma^{-1}(\mu_1 + \mu_0) + \log \frac{\pi}{1 - \pi}. \quad (7.8)$$

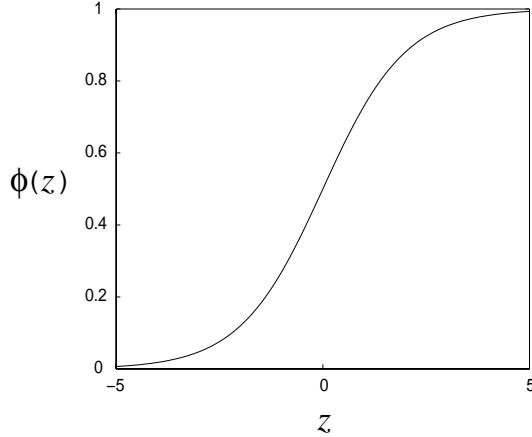


Figure 7.6: A plot of the logistic function.

We see that the posterior probability that $Y = 1$ takes the form:

$$\phi(z) \triangleq \frac{1}{1 + e^{-z}}, \quad (7.9)$$

where $z = \beta^T x + \gamma$ is an affine function of x . The function $\phi(z)$ is a smooth, sigmoid-shaped function known as the *logistic function* (see Figure 7.6).

The fact that the feature vector x enters into the posterior probability via an affine function has an important geometric interpretation; in particular, this implies that the contours of equal posterior probability are lines in the feature space. That is, the term $\beta^T x$ is proportional to the projection of x on β , and this projection is equal for all vectors x that lie along a line orthogonal to β . Consider in particular the case in which the variances σ_j^2 are equal to one; thus let $\Sigma = I$. In this case β is equal to $\mu_1 - \mu_0$, and the contours of equal posterior probability are lines that are orthogonal to the difference vector between the means of the two classes (see Figure 7.7(a)).

We obtain equal values of posterior probability for the two classes when $z = 0$ (because the logistic function in Eq. (7.9) evaluates to 0.5 when $z = 0$). To interpret this result geometrically, consider first the case in which the prior probabilities π and $1 - \pi$ are equal. In this case the term $\log(\pi/(1 - \pi))$ vanishes and we can rewrite z as follows:

$$z = (\mu_1 - \mu_0)^T \left(x - \frac{(\mu_1 + \mu_0)}{2} \right). \quad (7.10)$$

This is equal to zero for vectors x whose projection on $(\mu_1 - \mu_0)$ is equal to the arithmetic average of the two class means. Thus the posterior probabilities for the two classes are equal when x is equidistant from the two means. This corresponds to the solid line in Figure 7.7(a).

The prior probability π enters via the *log odds ratio* $\log(\pi/(1 - \pi))$. This effect of this term can be interpreted as a shift along the abscissa in Figure 7.6. For values of π larger than 0.5 we obtain a shift to the left, which, for a given point in the feature space, corresponds to a larger value of the

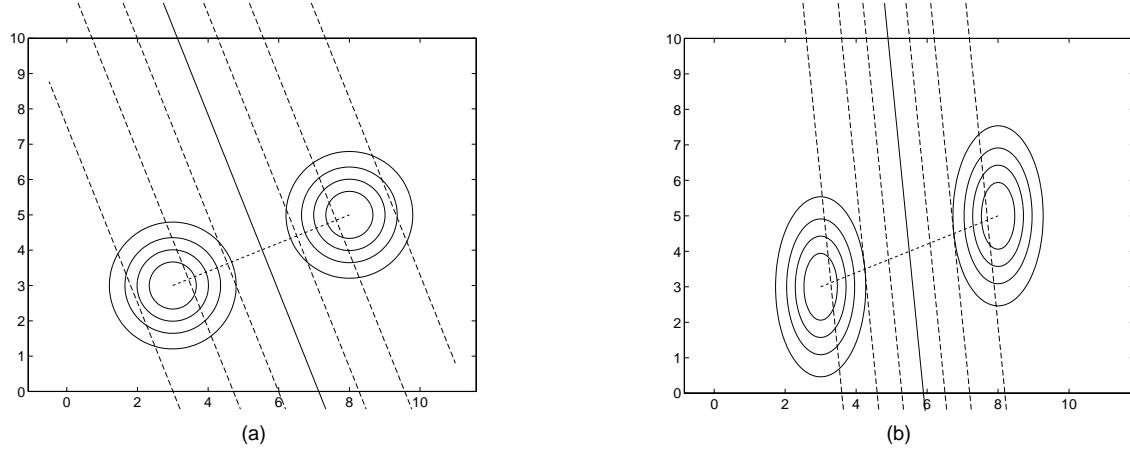


Figure 7.7: (a) The dashed lines and the solid line are contours of equal posterior probability. Note that they are orthogonal to the dotted line connecting the two mean vectors. (b) When $\sigma_1 \neq \sigma_2$, the contours of equal posterior probability are still lines, but they are no longer orthogonal to the difference between the mean vectors.

posterior for class $Y = 1$ (see Figure 7.8(a)). We obtain a shift to the right for π smaller than 0.5 (see Figure 7.8(b)).

Finally, let us consider the case of a general matrix Σ . The contours of equal posterior probability are still lines in the feature space, but in general these lines are no longer orthogonal to the difference vector between the means. If we define new features w via the equation $w \triangleq \Sigma^{-1}x$, however, we obtain the orthogonal geometry of Figure 7.7(a) in the w feature space, which implies an affine geometry in the original feature space. Figure 7.7(b) is an example of this case. Note that the set of vectors that have equal posterior probability for the two classes—the solid line in the figure—are no longer equidistant from the two class means.¹

As in Chapter 6 it is common to suppress the difference between linear and affine functions to simplify our notation. Thus we augment the vector x to include a first component that is equal to 1, and define the augmented parameter vector $\theta \triangleq (\gamma - \log(\pi/(1-\pi)), \beta^T)^T$. Using this notation, we can summarize the results of this section as follows: for Gaussian class-conditional densities, the posterior probability takes the form:

$$p(Y = 1 | x, \theta) = \frac{1}{1 + e^{-\theta^T x}} \quad (7.11)$$

where the parameter vector θ is a function of the means μ_k , the covariance matrix Σ , and the prior probability π .

In summary, we have found that the posterior probability for Gaussian class-conditional densities is the logistic function of a linear function of a feature vector x . We thus have obtained a

¹We can redefine the distance metric, however, basing it on the matrix Σ^{-1} . In this case the points on the solid line are equidistant from the class means. This metric is known as *Mahalanobis distance*; see Exercise ?? for more details.

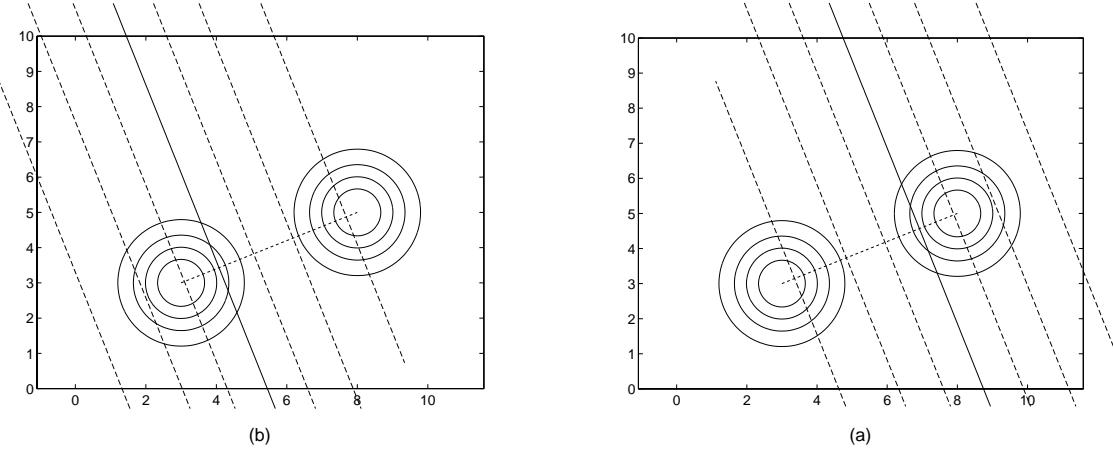


Figure 7.8: The class $Y = 1$ is the upper rightmost of the two Gaussians. (a) When the prior π is greater than 0.5, the contours are shifted to the left, corresponding to a greater posterior probability of $Y = 1$ for a given point in the feature space. (b) When the prior π is less than 0.5, the contours are shifted to the right.

linear classifier—contours of equal posterior probability are lines in the feature space. Inspecting the derivation that yielded this result, we see that the key assumption is that the covariance matrix is the same in the two classes; this leads to a cancellation of the quadratic $x^T \Sigma^{-1} x$ term in the numerator and denominator of the posterior probability. If we retract this assumption and allow different covariance matrices for the two classes, we still obtain a logistic form for the posterior probability, but the argument to the logistic function is now quadratic in x . The corresponding classifier, which has quadratic contours of equal posterior probability, is referred to as a *quadratic classifier*.

Maximum likelihood estimates

In this section we show how to obtain maximum likelihood parameter estimates based on a training set \mathcal{D} composed of N observations: $\mathcal{D} = \{(x_n, y_n); n = 1, \dots, N\}$. This problem has a straightforward solution that makes use of our work on density estimation in Chapter 5. Reasoning intuitively, suppose that we split the training data into two subsets, one in which $y_n = 0$ and the other in which $y_n = 1$. To estimate π we calculate the proportion of the data in the subset corresponding to $y_n = 1$; this is the maximum likelihood estimate of π . Moreover, we obtain separate maximum likelihood estimates of the Gaussian parameters for each of the two classes, pooling the estimates of the variances to take account of the fact that σ_j is the same in the two classes. This intuitively-defined solution is in fact the overall maximum likelihood solution, as we now verify.

We first form the log likelihood:

$$l(\theta | \mathcal{D}) = \log \left\{ \prod_{n=1}^N p(y_n | \pi) \prod_{j=1}^m p(x_{j,n} | y_n, \theta_j) \right\} \quad (7.12)$$

$$= \sum_{n=1}^N \log p(y_n | \pi) + \sum_{n=1}^N \sum_{j=1}^m \log p(x_{j,n} | y_n, \theta_j), \quad (7.13)$$

where we see that we obtain two separate terms, one for the marginal distribution of Y and the other for the conditional distribution of X_j given Y . Maximizing with respect to π involves only the former term, and for π we therefore obtain:

$$\hat{\pi}_{ML} = \arg \max_{\pi} \sum_{n=1}^N \log p(y_n | \pi) \quad (7.14)$$

$$= \arg \max_{\pi} \sum_{n=1}^N \{y_n \log \pi + (1 - y_n) \log(1 - \pi)\}, \quad (7.15)$$

where the latter equation uses Eq. (7.1). As we have seen in Chapter 5 (cf. Eq. (5.37)), the solution to this constrained optimization problem is the sample proportion:

$$\hat{\pi}_{ML} = \frac{\sum_{n=1}^N y_n}{N}, \quad (7.16)$$

where the numerator $\sum_{n=1}^N y_n$ is the count of the number of times that the class $Y = 1$ is observed.

Maximization with respect to the parameters θ_j involves only the second term in Eq. (7.13), which we expand further as:

$$\begin{aligned} & \sum_{n=1}^N \sum_{j=1}^m \log p(x_{j,n} | y_n, \theta_j) \\ &= \sum_{n=1}^N \sum_{j=1}^m \log \{p(x_{j,n} | y_n = 1, \mu_{j1}, \sigma_j)^{y_n} p(x_{j,n} | y_n = 0, \mu_{j0}, \sigma_j)^{1-y_n}\} \end{aligned} \quad (7.17)$$

$$= \sum_{j=1}^m \left\{ \sum_{n=1}^N y_n \log p(x_{j,n} | y_n = 1, \mu_{j1}, \sigma_j) + \sum_{n=1}^N (1 - y_n) \log p(x_{j,n} | y_n = 0, \mu_{j0}, \sigma_j) \right\}. \quad (7.18)$$

Each term in the brackets depends on only one of the parameter vectors $\theta_j = (\mu_{j0}, \mu_{j1}, \sigma_j)$. Thus the problem decomposes into m separate optimization problems, one for each j .

Let us first consider the estimation of μ_{j1} . Plugging in from Eq. (7.3) for $p(x_{j,n} | y_n = 1, \mu_{j1}, \sigma_j)$, and dropping constants, we have:

$$\hat{\mu}_{j1,ML} = \arg \max_{\mu_{j1}} \left\{ -\frac{1}{2} \sum_{n=1}^N y_n (x_{j,n} - \mu_{1j})^2 \right\}. \quad (7.19)$$

This is a weighted least-squares problem, where the “weights” are the binary values y_n . Taking the derivative and setting to zero, we obtain:

$$\hat{\mu}_{j1,ML} = \frac{\sum_{n=1}^N y_n x_{j,n}}{\sum_{n=1}^N y_n}. \quad (7.20)$$

Thus the maximum likelihood estimate is the sample average of the values $x_{j,n}$ for those data points in class $Y = 1$. Similarly, for $\hat{\mu}_{j0}$ we obtain:

$$\hat{\mu}_{j0,ML} = \frac{\sum_{n=1}^N (1 - y_n) x_{j,n}}{\sum_{n=1}^N (1 - y_n)}, \quad (7.21)$$

which is the average of the $x_{j,n}$ for those data points in class $Y = 0$.

Finally, as we ask the reader to verify in Exercise ??, maximization with respect to the variance σ_j^2 yields:

$$\hat{\sigma}_{j,ML}^2 = \frac{\sum_{n=1}^N y_n (x_{j,n} - \hat{\mu}_{j1,ML})^2}{\sum_{n=1}^N y_n} + \frac{\sum_{n=1}^N (1 - y_n) (x_{j,n} - \hat{\mu}_{j0,ML})^2}{\sum_{n=1}^N (1 - y_n)}; \quad (7.22)$$

a pooled estimate of the variance.

Multiway classification

In this section we consider the generalization to multiway classification, in which the class label Y can take on one of K values.

Let Y be a multinomial random variable with components Y^k and parameter vector π . By definition we have:

$$\pi_k = p(Y^k = 1 | \pi). \quad (7.23)$$

For each of the K values of Y , define a Gaussian class-conditional density:

$$p(x | Y^k = 1, \theta) = \frac{1}{(2\pi)^{m/2} |\Sigma|^{1/2}} \exp \left\{ -\frac{1}{2} (x - \mu_k)^T \Sigma^{-1} (x - \mu_k) \right\}, \quad (7.24)$$

where μ_k is the mean associated with the k th class and Σ is a covariance matrix, assumed constant across the K classes. If Σ is diagonal, then the components of X are conditionally independent given the class label Y and the appropriate graphical model is given by Figure 7.4(a). For general Σ , we represent our model as Figure 7.4(c).

The posterior probability of class k is obtained via Bayes rule:

$$p(Y^k = 1 | x, \theta) = \frac{p(x | Y^k = 1, \theta) p(Y^k = 1 | \pi)}{\sum_l p(x | Y^l = 1, \theta) p(Y^l = 1 | \pi)} \quad (7.25)$$

$$= \frac{\pi_k \exp\{-\frac{1}{2} (x - \mu_k)^T \Sigma^{-1} (x - \mu_k)\}}{\sum_l \pi_l \exp\{-\frac{1}{2} (x - \mu_l)^T \Sigma^{-1} (x - \mu_l)\}} \quad (7.26)$$

$$= \frac{\exp\{\mu_k^T \Sigma^{-1} x - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k\}}{\sum_l \exp\{\mu_l^T \Sigma^{-1} x - \frac{1}{2} \mu_l^T \Sigma^{-1} \mu_l + \log \pi_l\}}, \quad (7.27)$$

where the cancellation of the quadratic $x^T \Sigma^{-1} x$ terms again leaves us with exponents that are linear in x . Defining parameter vectors β_k :

$$\beta_k \triangleq \begin{bmatrix} -\mu_k^T \Sigma^{-1} \mu_k + \log \pi_k \\ \Sigma^{-1} \mu_k \end{bmatrix} \quad (7.28)$$

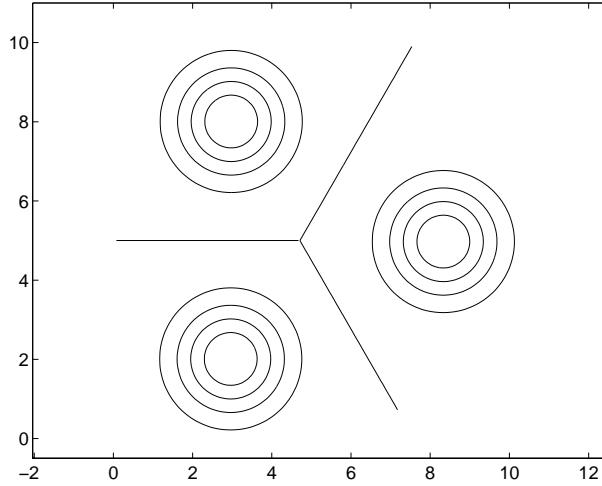


Figure 7.9: Contours of the softmax function. Each line is obtained by setting $\phi_k(z) = \phi_l(z)$ for $k \neq l$. Such a line is a contour of equal posterior probability for classes k and l .

and again simplifying our result by augmenting the vector x to include a first component equal to one, we have:

$$p(Y^k = 1 | x, \theta) = \frac{e^{\beta_k^T x}}{\sum_l e^{\beta_l^T x}}. \quad (7.29)$$

The function $\phi_k(z) \triangleq e^{z_k} / \sum_l e^{z_l}$ is a smooth function known as the *softmax function*.

The softmax function is a generalization of the logistic function and it has a similar geometric interpretation. Indeed we can transfer much of our earlier work to the multiway setting by considering the ratios of posterior probabilities between pairs of classes. In taking the ratio of $p(Y^k = 1 | x, \theta)$ and $p(Y^l = 1 | x, \theta)$, for $k \neq l$, the denominator in the softmax function cancels and we obtain an exponential with exponent $(\beta_k - \beta_l)^T x$. This again involves a projection and thus contours of equal pairwise probability are again lines in the feature space (see Figure 7.9). Moreover, the prior probabilities π again take the form of log odds and act as additive constants in the exponential.

When $\Sigma = \sigma I$, we see from Eq. (7.28) that β_k is proportional to μ_k , and thus the contours of equal probability are again orthogonal to the differences between the class means. For general Σ we obtain the same orthogonal geometry for the transformed coordinates $w \triangleq \Sigma^{-1}x$, which implies an affine geometry for the features x .

The calculation of maximum likelihood estimates for the multiway Gaussian classifier is straightforward and we ask the reader to carry out the calculation in Exercise ???. The results can be summarized as follows: We again divide the data into subsets corresponding to the different values of Y . Separate maximum likelihood estimates of the Gaussian parameters are obtained for each class, and the covariance estimates are pooled. Moreover, the maximum likelihood estimates of π are the proportions of data falling into the K classes.

The classifier that we have presented in this section is again a *linear classifier*. The linearity

again arises from the Gaussian assumption for the class-conditional densities, together with the assumption of a constant covariance matrix.

7.2.2 The naive Bayes classifier

We now turn to the setting of discrete features, in which each feature X_j can take on one of K values. In this setting the graphical model shown in Figure 7.4(a) is often referred to as the “naive Bayes classifier.” We discuss the naive Bayes classifier in this section, calculating the posterior probability and maximum likelihood parameter estimates.

Much of the work in the previous section carries over to the discrete setting. In particular, the joint probability remains the same as before:

$$p(x, y | \theta) = p(y | \pi) \prod_{j=1}^m p(x_j | y, \theta_j). \quad (7.30)$$

We again let Y be a multinomial random variable with components Y^k , defining the probability vector π , where:

$$\pi_k \triangleq p(Y^k = 1 | \pi). \quad (7.31)$$

Finally, treating the variables X_j as multinomial random variables with components X_j^k , where $X_j^k = 1$ for one and only one value of k , we write the class-conditional densities as follows:

$$p(x_1, x_2, \dots, x_m | Y^i = 1, \eta) = \prod_j \prod_k \eta_{ijk}^{x_j^k}, \quad (7.32)$$

where $\eta_{ijk} \triangleq p(x_j^k = 1 | Y^i = 1, \eta)$ is the probability that the j th feature X_j takes on its k th value, for the i th value of the class label Y . Note that the product over k in Eq. (7.32) arises from the definition of multinomial probabilities, and the product over j reflects the assumption that the features are conditionally independent.

Posterior probability

Let us calculate the posterior probability for the naive Bayes classifier. We have:

$$p(Y^i = 1 | x, \eta) = \frac{\pi_i \prod_j \prod_k \eta_{ijk}^{x_j^k}}{\sum_l \pi_l \prod_j \prod_k \eta_{ljk}^{x_j^k}} \quad (7.33)$$

$$= \frac{\exp\{\log \pi_i + \sum_j \sum_k x_j^k \log \eta_{ijk}\}}{\sum_l \exp\{\log \pi_l + \sum_j \sum_k x_j^k \log \eta_{ljk}\}}. \quad (7.34)$$

As in the Gaussian case, this is again a softmax function of a linear combination of the features. We can express this result in the standardized form:

$$p(Y^i = 1 | x, \eta) = \frac{e^{\beta_i^T x}}{\sum_l e^{\beta_l^T x}}, \quad (7.35)$$

with a bit of creativity in the definitions of x and β . In particular, we redefine the vector x by stacking the multinomial vectors x_j vertically. Thus, the components of x are the values x_j^k , where the superscript k varies more rapidly than the subscript j . We also augment the resulting vector to have a first component of one. Similarly, we define β_i as a vector in which the doubly-indexed components $\log \eta_{ijk}$ are arranged, with i fixed and k varying faster than j . We let the first component of β_i be equal to $\log \pi_i$. Given these definitions we obtain Eq. (7.35) as the posterior probability for the naive Bayes model.

Although the feature space is a discrete hypercube in the naive Bayes setting, it is interesting that the classifier is formally the same as the linear discriminant classifier, with log odds playing the role that difference vectors played in the Gaussian case.

In the case of binary classification, we can divide numerator and denominator by the numerator in Eq. (7.34) and obtain the logistic function of a linear function of the features:

$$p(Y = 1 | x, \theta) = \frac{1}{1 + \exp\{-\theta^T x\}} \quad (7.36)$$

for appropriate definitions of θ and x .

Maximum likelihood estimates

Finally, let us calculate the maximum likelihood estimates of the parameters for the naive Bayes classifier. We again assume that we have a training set \mathcal{D} composed of N observations: $\mathcal{D} = \{(x_n, y_n); n = 1, \dots, N\}$.

From Eq. (7.30) we obtain the log likelihood:

$$l(\theta | \mathcal{D}) = \sum_{n=1}^N \log p(y_n | \pi) + \sum_{n=1}^N \sum_{j=1}^m \log p(x_{j,n} | y_n, \eta), \quad (7.37)$$

where for the purposes of this section we define x and y to be the vectors of all observations $x_{j,n}$ and y_n , respectively. The first term again decouples to yield separate maximum likelihood estimates of π . Focusing on the second term, and recalling that the sum over k of the parameters η_{ijk} must equal one, we introduce Lagrange multipliers λ_{ij} and maximize:

$$\tilde{l}(\eta | \mathcal{D}) \triangleq \sum_{n=1}^N \sum_i \sum_j \sum_k x_{j,n}^k y_n^i \log \eta_{ijk} + \sum_i \sum_j \lambda_{ij} \left(1 - \sum_k \eta_{ijk}\right). \quad (7.38)$$

This yields:

$$\frac{\partial \tilde{l}}{\partial \eta_{ijk}} = \frac{\sum_n x_{j,n}^k y_n^i}{\eta_{ijk}} - \lambda_{ij}. \quad (7.39)$$

Setting to zero and summing both sides with respect to k , we have:

$$\lambda_{ij} = \sum_k \sum_n x_{j,n}^k y_n^i \quad (7.40)$$

$$= \sum_n \sum_k x_{j,n}^k y_n^i \quad (7.41)$$

$$= \sum_n y_n^i. \quad (7.42)$$

Finally, substituting back into Eq. (7.39), we obtain:

$$\hat{\eta}_{ijk,ML} = \frac{\sum_n x_{j,n}^k y_n^i}{\sum_n y_n^i}, \quad (7.43)$$

in which the numerator is the number of observations in the i th class for which the j th feature takes on its k th value. The denominator normalizes this count by dividing by the number of observations in the i th class.

7.2.3 The exponential family

For all of the generative classification models studied thus far, the posterior probability takes a simple functional form—a logistic function for the binary problem and a softmax function in the multiway problem. Moreover, for multinomial and Gaussian class-conditional densities (in the latter case with equal, but otherwise arbitrary, class covariance matrices), the contours of equal posterior probability are hyperplanes in the feature space. In fact, as we see in this section, these results are not restricted to multinomial and Gaussian probabilities; but hold for a wide range of class-conditional densities.

The exponential family of probability distributions is a large family that includes the multinomial and Gaussian distributions, as well as a number of other classical distributions such as the binomial, the Poisson, the gamma and the Dirichlet. In Chapter 8 we provide a detailed discussion of the exponential family; here we simply present the functional form of this family, and consider using exponential family distributions as class-conditional densities for classification.

The exponential family is defined as follows:

$$p(x | \eta) = \exp\{\eta^T x - a(\eta)\}h(x), \quad (7.44)$$

where η is a parameter vector. It is a useful exercise to verify that the distributions listed above can all be put in this standard form, for appropriate definitions of the functions $a(\eta)$ and $h(x)$. (We will carry out this exercise in Chapter 8).

Let us now consider a binary classification problem for a generic class-conditional density from the exponential family. We assume that the densities for the two classes are the same, up to the parameter vector η . That is, we let the density for class $Y = 1$ be parameterized by η_1 and let the density for class $Y = 0$ be parameterized by η_0 . Let the prior probabilities be equal for simplicity. We obtain the posterior probability from Bayes rule:

$$p(Y = 1 | x, \eta) = \frac{p(x | Y = 1, \eta)p(Y = 1 | \pi)}{p(x | Y = 1, \eta)p(Y = 1 | \pi) + p(x | Y = 0, \eta)p(Y = 0 | \pi)} \quad (7.45)$$

$$= \frac{\exp\{\eta_1^T x - a(\eta_1)\}h(x)}{\exp\{\eta_1^T x - a(\eta_1)\}h(x) + \exp\{\eta_0^T x - a(\eta_0)\}h(x)} \quad (7.46)$$

$$= \frac{1}{1 + \exp\{-(\eta_0 - \eta_1)^T x - a(\eta_0) + a(\eta_1)\}}. \quad (7.47)$$

Thus we find that the posterior probability is the logistic function of a linear function of x .

Similarly, for the multiway classification problem we have:

$$p(Y^k = 1 | x, \eta) = \frac{p(x | Y^k = 1, \eta_k)p(Y^k = 1 | \pi)}{\sum_l p(x | Y^l = 1, \eta_l)p(Y^l = 1 | \pi)} \quad (7.48)$$

$$= \frac{\exp\{\eta_k^T x - a(\eta_k)\}h(x)}{\sum_l \exp\{\eta_l^T x - a(\eta_l)\}h(x)} \quad (7.49)$$

$$= \frac{\exp\{\eta_k^T x - a(\eta_k)\}}{\sum_l \exp\{\eta_l^T x - a(\eta_l)\}}, \quad (7.50)$$

where again we have assumed equal class priors for simplicity. The result is the softmax function of a linear function of x .

7.3 Discriminative models

In Section 7.2.3 we have seen that a wide range of class-conditional densities all yield the same logistic-linear or softmax-linear form for the posterior probability. This invariance of the functional form of the posterior probability to the specific choice of class-conditional density is good news, because in practice it can be difficult to choose the class-conditional density. This problem is particularly difficult in the case of a high-dimensional feature vector. Consider the Gaussian case. The assumption of a diagonal covariance matrix—corresponding to conditional independence of the features—is often unrealistic. We can allow arbitrary covariance matrices, but this requires us to estimate $O(m^2)$ parameters, which may be prohibitive for large m . Often we would like instead to consider families of covariance matrices that depend on more than m but fewer than m^2 parameters. In some cases there is a natural ordering or grouping of the features (e.g., in the case of time series data or spatial data) that yield natural definitions of such structured covariance matrices. In many other cases, however, there is no obvious way to justify a particular form of structured covariance matrix, and we are left with a choice between the (highly-biased) case of a diagonal covariance matrix and the (highly-variable) case of a full covariance matrix. The fact, however, that all of these choices yield the same linear form for the posterior probability suggests that it may not be necessary to make such a choice. Moreover, the fact that densities other than the Gaussian density yield the same linear classifier suggests that we may not even need to specify the density.

In this section we discuss discriminative models. In discriminative modeling the posterior probability is modeled directly, quite apart from any considerations regarding class-conditional probabilities. Instead of assuming Gaussian or multinomial class-conditional densities and deriving the linearity of the classifier as a consequence, we instead assume linearity at the outset, by assuming that x enters into the model via a linear combination $\theta^T x$. To complete the model, we make an additional assumption regarding the (nonlinear) function that maps from $\theta^T x$ to the posterior

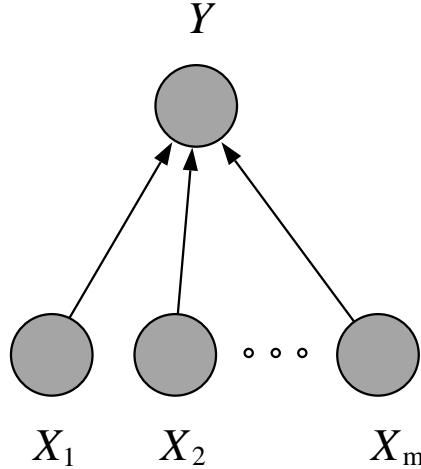


Figure 7.10: The graphical representation of a discriminative classification model.

probability. Taking a hint from the generative setting, we assume a logistic or softmax function at the outset, but we will also explore other possibilities.

The main problem will be that of estimating the parameters of the resulting classifier. Given that we no longer have underlying class-conditional densities, we cannot define the parameters of the classifier in terms of underlying means, covariances, log probabilities or the like. Instead we will have to find a way to estimate the parameters “directly.”

The graphical model that we study in this section is shown in Figure 7.10. Note that in this figure we have treated the components of the feature vector X as separate nodes: X_1, X_2, \dots, X_m . We have done this to emphasize the relationship—as well as the contrast—with the discussion of the generative approach in the previous section. Note in particular that we are not assuming nor implying conditional independence of the features. Indeed, in this section we make no assumptions regarding the marginal probability $p(x)$; our goal is only to model the conditional probability $p(y|x)$. This is of course the same setting as that of regression, and indeed the methods that we discuss in this section are closely related to regression.

7.3.1 Logistic regression

We begin by considering the case of binary classification. The first model that we consider is *logistic regression*, in which the conditional probability $p(y|x)$ is modeled as a function $\phi(\theta^T x)$, where ϕ is the logistic function. This functional form is of course suggested by the generative models in Section 7.2.

The class label Y is a Bernoulli random variable, and the modeling problem is that of determining the probability that Y takes the value one for each input X . Note that this probability, $p(Y = 1|x)$, is the same as the conditional expectation:

$$E(y|x) = 1 \cdot p(Y = 1|x) + 0 \cdot p(Y = 0|x) \quad (7.51)$$

$$= p(Y = 1 | x). \quad (7.52)$$

Thus, as in the case of regression, the goal is that of modeling the conditional expectation of Y given X . In the regression case, we added a Gaussian error term ϵ to the conditional expectation. This approach, however, is clearly inappropriate here given that Y can only take on the discrete values zero and one. Instead, we define $\mu(x) \triangleq p(Y = 1 | x)$ and write the Bernoulli distribution in the following way:

$$p(y | x) = \mu(x)^y (1 - \mu(x))^{1-y}. \quad (7.53)$$

This is the usual definition of the Bernoulli distribution; however, we still need to specify the dependence of the Bernoulli parameter $\mu(x)$ on x .

To complete the specification of the model, we assume that (1) the conditional expectation depends on x via the inner product $\eta(x) \triangleq \theta^T x$, where θ is a parameter vector, and (2) the inner product $\eta(x)$ is converted to a probability scale via the logistic function. Thus we have:

$$\mu(x) = \frac{1}{1 + e^{-\eta(x)}}. \quad (7.54)$$

as the probability model for the conditional expectation $\mu(x) \triangleq p(Y = 1 | x, \theta)$.

Recall that in the current section we simply treat these assumptions as axiomatic—as an attempt to model posterior probabilities in a simple parametric way independently of assumptions regarding class-conditional densities. Figure 7.11 shows an example that helps to suggest the reasonableness of this approach. In this figure it appears to be difficult to choose a model for the class-conditional densities; in particular, a Gaussian assumption does not seem reasonable. It seems significantly less problematic to choose a discriminative model in this case, and indeed the linear boundary implied by the logistic regression model appears to be reasonable. Such examples are by no means uncommon.

Some properties of the logistic function

In this section we collect together several results regarding the logistic function that will be of use in the following section and in several later chapters.

Let us write the logistic function as a map from a variable η to a variable μ :

$$\mu = \frac{1}{1 + e^{-\eta}} \quad (7.55)$$

The logistic function is invertible; thus we can also obtain a map from μ to η :

$$\eta = \log \left(\frac{\mu}{1 - \mu} \right), \quad (7.56)$$

which has the form of a log odds.

This inverse form simplifies the calculation of derivatives. In particular, we have:

$$\frac{d\eta}{d\mu} = \frac{d}{d\mu} \log \left(\frac{\mu}{1 - \mu} \right) \quad (7.57)$$

$$= \frac{1}{\mu(1 - \mu)}, \quad (7.58)$$

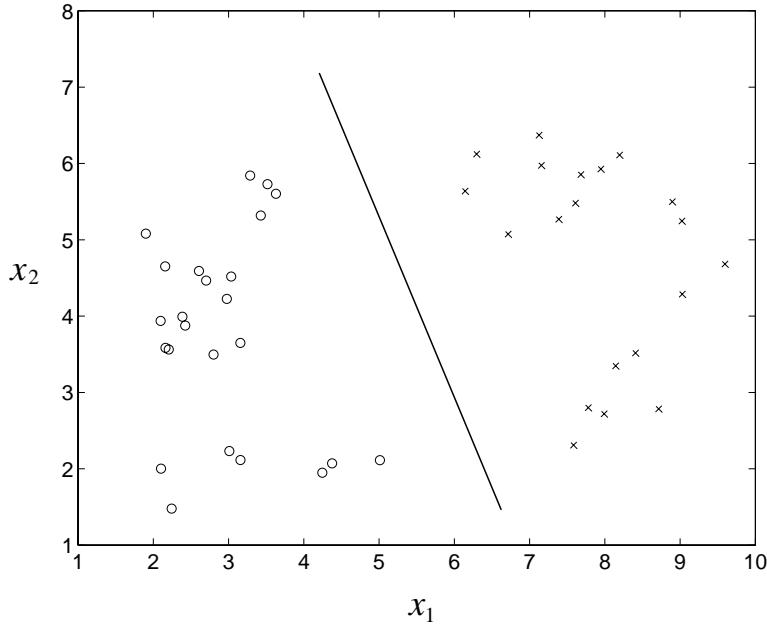


Figure 7.11: An example in which it is difficult to specify the class-conditional densities required for a generative model, but where a linear discriminative boundary between the classes seems reasonable.

from which we obtain:

$$\frac{d\mu}{d\eta} = \mu(1 - \mu). \quad (7.59)$$

This expresses the derivative of the logistic function as a function of μ . We can also use Eq. (7.55) to obtain the derivative as a function of η , but the form in Eq. (7.59) will prove to be more useful.

The likelihood

In this section we begin our discussion of maximum likelihood estimation of the parameters θ based on a training set $\mathcal{D} = \{(x_n, y_n); n = 1, \dots, N\}$. As in our discussion of regression in Chapter 6, we consider batch and on-line methods for parameter estimation.

Let $\eta_n = \theta^T x_n$ and let $\mu_n = 1/(1 + e^{-\eta_n})$ denote the corresponding value of the logistic function, in accordance with our definitions in the previous section. Note that we have omitted the explicit dependence of μ_n on x_n to simplify our notation. Moreover, let η and μ denote the vectors of these values as we range across n ; thus: $\eta = (\eta_1, \eta_2, \dots, \eta_N)$ and $\mu = (\mu_1, \mu_2, \dots, \mu_N)$.

To obtain the likelihood we take the product of N Bernoulli probabilities using Eq. (7.53):

$$p(y_1, \dots, y_N | x_1, \dots, x_N, \theta) = \prod_n \mu_n^{y_n} (1 - \mu_n)^{1-y_n}. \quad (7.60)$$

Taking logarithms yields:

$$l(\theta | \mathcal{D}) = \sum_n \{y_n \log \mu_n + (1 - y_n) \log(1 - \mu_n)\}, \quad (7.61)$$

and it is this expression that we must maximize with respect to θ .² Recall that μ_n is a function of θ whereas y_n is not.

We calculate the gradient of the log likelihood:

$$\nabla_{\theta} l = \sum_n \left(\frac{y_n}{\mu_n} - \frac{1 - y_n}{1 - \mu_n} \right) \frac{d\mu_n}{d\eta_n} x_n \quad (7.62)$$

$$= \sum_n \frac{y_n - \mu_n}{\mu_n(1 - \mu_n)} \mu_n(1 - \mu_n) x_n \quad (7.63)$$

$$= \sum_n (y_n - \mu_n) x_n. \quad (7.64)$$

It is interesting to note that this gradient has the same form as the gradient of the log likelihood for linear regression (cf. Eq. (6.12)). In both cases we obtain a difference between y_n and the conditional expectation μ_n , multiplied by the input x_n .

An on-line estimation algorithm

An on-line estimation algorithm can be obtained by dropping the summation sign and following the stochastic gradient of the log likelihood. Let $\theta^{(t)}$ denote the value of the parameter vector at the t th step of the algorithm. If (x_n, y_n) denotes the data point presented to the algorithm at the t th step, we write:

$$\theta^{(t+1)} = \theta^{(t)} + \rho(y_n - \mu_n^{(t)}) x_n, \quad (7.65)$$

where $\mu_n^{(t)} \triangleq \phi(\theta^{(t)T} x_n)$ and where ρ is a step size.

Note that this on-line algorithm is identical in form to the LMS algorithm differing only in the definition of the conditional expectation. To understand the (important) implications of the difference, let us return to an issue that motivated our development of classification methods. Recall in particular Figure ??, where we considered the effect on linear regression of adding the point $(1.5, 1)$ to the training set. The linear fit is altered significantly by the addition of this point. One way to see this is to note that the error, $(y_n - \mu_n^{(t)})$, in the LMS algorithm is large; thus the algorithm will make a large adjustment to the parameter vector $\theta^{(t)}$. For the on-line logistic regression algorithm in Eq. (7.65), however, $\mu_n^{(t)}$ is near one, given that the logistic function is evaluated in its rightmost tail. As suggested in Figure 7.12, the error, $(y_n - \mu_n^{(t)})$, is therefore essentially zero. Thus, as we see from Eq. (7.65), there is little change in the parameters. In general, points that are already classified correctly do not affect the fit.

²The function in Eq. (7.61) is the *cross-entropy* function. See Appendix XXX for further discussion of the cross-entropy in the context of information theory.

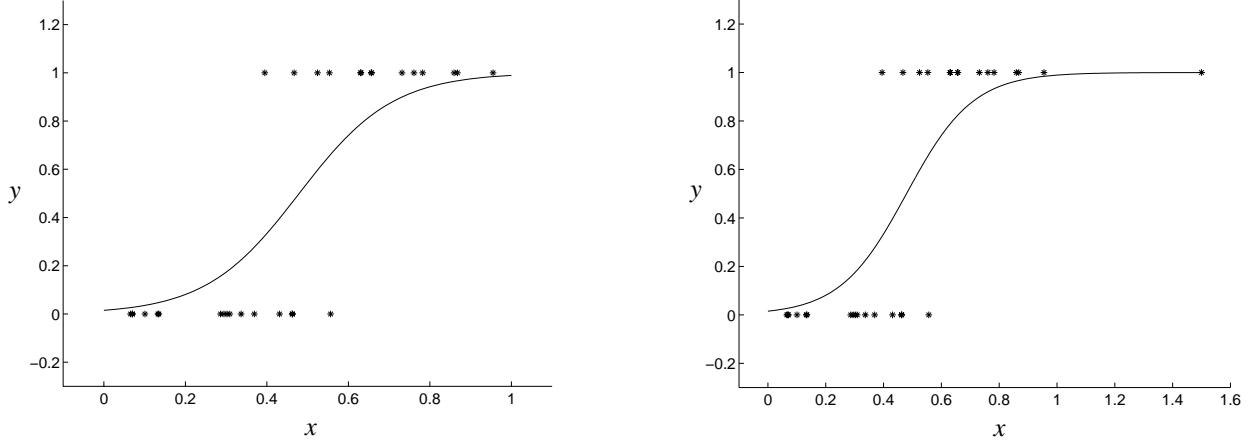


Figure 7.12: (a) The fit of a logistic regression model to the data in Figure 7.1. (b) Adding the point $(1.5, 1)$ to the data set does not change the fit (cf. Figure 7.2).

The iteratively reweighted least squares (IRLS) algorithm

To obtain a batch algorithm we could restore the summation sign in Eq. (7.64) and follow the steepest descent direction, but as in the linear regression case this algorithm has little to recommend it. We instead describe an algorithm, known as the *iteratively reweighted least squares (IRLS)* algorithm, that is closer in spirit to the direct solution of the normal equations.

The IRLS algorithm is a Newton-Raphson algorithm.³ In preparation for deriving the algorithm, let us note that the normal equations can also be viewed, somewhat perversely, from the point of view of the Newton-Raphson algorithm.

Consider a function $J(\theta)$ which is to be minimized with respect to θ . Recall (see Appendix XXX) that the Newton-Raphson algorithm is an iterative algorithm that takes the following general form:

$$\theta^{(t+1)} = \theta^{(t)} - H^{-1} \nabla_{\theta} J, \quad (7.66)$$

where $\nabla_{\theta} J$ and H are the gradient vector and Hessian matrix of $J(\theta)$ respectively (and both are evaluated at $\theta^{(t)}$).

In the case of linear regression, the cost function, $J = \frac{1}{2}(y - X\theta)^T(y - X\theta)$, is a quadratic function of θ . We calculated the gradient of J in Chapter 6, finding:

$$\nabla_{\theta} J = -X^T(y - X\theta). \quad (7.67)$$

Taking another derivative we obtain the Hessian:

$$H = -X^T X. \quad (7.68)$$

³This statement is not entirely accurate, but it is accurate enough for current purposes. See Chapter 8 for further details.

Thus we can apply the Newton-Raphson algorithm to the problem of minimizing J , obtaining:

$$\theta^{(t+1)} = \theta^{(t)} + (X^T X)^{-1} X^T (y - X\theta^{(t)}) \quad (7.69)$$

$$= (X^T X)^{-1} X^T y, \quad (7.70)$$

where we see that the right-hand-side is the solution to the normal equations. Thus Newton-Raphson hops to the solution in a single step, not a surprise given that J is a quadratic function.

In the logistic regression problem, the function to be optimized is the log likelihood, and this function is not quadratic. Nonetheless it is “nearly” quadratic, and we should not be surprised to see that Newton-Raphson for logistic regression has similarities to the linear regression solution. Indeed, as we will see, the similarity is strong.

The function that we wish to optimize is the log likelihood shown in Eq. (7.61). We have already calculated the gradient of the log likelihood in Eq. (7.64). Writing this result in vector notation, we have:

$$\nabla_{\theta} l = \sum_n (y_n - \mu_n) x_n = X^T (y - \mu), \quad (7.71)$$

where we have defined $\mu \triangleq (\mu_1, \mu_2, \dots, \mu_N)^T$. Taking a second derivative, we have:

$$H = - \sum_n \frac{d\mu_n}{d\eta_n} x_n x_n^T \quad (7.72)$$

$$= - \sum_n \mu_n (1 - \mu_n) x_n x_n^T \quad (7.73)$$

$$= -X^T W X, \quad (7.74)$$

where we have defined the diagonal weight matrix:

$$W \triangleq \text{diag}\{\mu_1(1 - \mu_1), \mu_2(1 - \mu_2), \dots, \mu_N(1 - \mu_N)\}, \quad (7.75)$$

Note that the μ_n values depend on the parameter vector θ , thus the weight matrix W depends on θ . We thus will use the notation $W^{(t)}$ to denote the weight matrix at the t th iteration of the algorithm.

Substituting into Eq. (7.66), we obtain:

$$\theta^{(t+1)} = \theta^{(t)} + (X^T W^{(t)} X)^{-1} X^T (y - \mu^{(t)}) \quad (7.76)$$

$$= (X^T W^{(t)} X)^{-1} [X^T W^{(t)} X \theta^{(t)} + X^T (y - \mu^{(t)})] \quad (7.77)$$

$$= (X^T W^{(t)} X)^{-1} X^T W^{(t)} z^{(t)}, \quad (7.78)$$

where we define:

$$z^{(t)} = \eta + [W^{(t)}]^{-1} (y - \mu^{(t)}). \quad (7.79)$$

The algorithm in Eq. (7.78) is the IRLS algorithm.

Inspecting Eq. (7.78) makes it clear why the algorithm is known as the “iteratively reweighted least squares” algorithm. Each iteration of the algorithm involves solving a weighted least-squares

problem (recall Eq. (??)). Moreover, given that the weight matrix W changes at each iteration, the least-squares problem is “iteratively reweighted.”

We can obtain some more insight into the IRLS algorithm, and in particular understand the role played by $z^{(t)}$, if we view the Newton-Raphson algorithm as solving a sequence of linearized problems.

Consider the following (heuristic) argument. For a particular value θ , and a particular vector x_n , let us linearize the logistic function around the “operating point,” $\eta_n = \theta^T x_n$. This linearization allows us to convert the value y_n , which is on a nonlinear scale, “backwards” to a value z_n on the linear scale defined by η_n . In particular, recall that the logistic function can be inverted (cf. Eq. (7.56)) to yield a map from μ_n to η_n . Expanding this inverse function in a first-order Taylor series, we define:

$$z_n \triangleq \eta_n + \frac{d\eta_n}{d\mu_n} (y_n - \mu_n), \quad (7.80)$$

where the derivative is evaluated at η_n , and thus depends implicitly on the parameter vector θ .

This argument suggests using z_n as a surrogate for y_n , in a linearized version of our logistic regression problem. We have another issue to deal with, however, if we wish to use linear regression methods to find parameter estimates: the Bernoulli random variables y_n do not have equal variance. In particular, y_n has variance $\mu_n(1 - \mu_n)$. To deal with this issue, we use weighted least squares. In particular, note that the elements of the weighting matrix W defined in Eq. (7.75) are exactly the Bernoulli variances. Thus we use W as our weight matrix.

We now solve a weighted least squares problem, with data z_n and weight matrix W . Writing the normal equations for this weighted least squares problem, and making the dependence on the iteration number t explicit, we obtain the IRLS iteration in Eq. (7.78).

The Newton-Raphson algorithm is a second-order algorithm and it generally converges rapidly. A small number of iterations of Eq. (7.78) are usually sufficient to obtain convergence of the parameter vector.

7.3.2 Multiway classification

In this section we discuss a generalization of logistic regression to the setting of multiway classification. Recall that in this case the class label Y can take on one of K values.

In Section 7.2.1 we derived the softmax-linear model:

$$p(Y^k = 1 | x, \theta) = \frac{e^{\theta_k^T x}}{\sum_l e^{\theta_l^T x}} \quad (7.81)$$

as the multiway generalization of the logistic-linear model. In that section, the softmax-linear form for the posterior probability was a consequence of our assumption of Gaussian (or more generally, exponential family) class-conditional probabilities. In the current section, however, we adopt a discriminative perspective in which the softmax-linear form is treated as an assumption, and we make no attempt to specify class-conditional probabilities. In this context, we refer to the model in Eq. (7.81) as a *softmax regression* model. As in the case of logistic regression, the main problem that we face is estimating the parameters θ_k “directly,” without making use of an underlying class-conditional model.

We use the notation μ_n^k to denote the posterior probability in Eq. (7.81). We also use $\eta_n^k = \theta_k^T x_n$ to denote the linear component of the softmax-linear model.

Some properties of the softmax function

The softmax function has several properties that are analogs of those of the logistic function that we discussed in Section 7.3.1.

The softmax function can be written as a map from a vector variable η to a vector variable μ . Letting η^i represent the i th component of η and letting μ^i represent the i th component of μ , we write:

$$\mu^i = \frac{e^{\eta^i}}{\sum_k e^{\eta^k}}. \quad (7.82)$$

This function is invertible up to an additive constant. That is, if we add the constant C to each of the components η^i , then the factor e^C cancels in the numerator and denominator of Eq. (7.82), yielding the same value of μ^i . Note in particular that if we take the logarithm of both sides of Eq. (7.82), we obtain the inverse:

$$\eta^i = \log \mu^i + D, \quad (7.83)$$

where $D = \log \sum_k e^{\eta^k}$ is a constant. Any other constant (including zero) will yield an equivalent inverse of the softmax function.

We turn to the calculation of the softmax derivatives. A subtlety in this case is that the derivative of μ_i with respect to η_j is non-zero for $i \neq j$, due to the denominator in Eq. (7.82). The calculation proceeds as follows:

$$\frac{\partial \mu_i}{\partial \eta_j} = \frac{(\sum_k e^{\eta_k}) e^{\eta_i} \delta_{ij} - e^{\eta_i} e^{\eta_j}}{(\sum_k e^{\eta_k})^2} \quad (7.84)$$

$$= \frac{e^{\eta_i}}{\sum_k e^{\eta_k}} \left(\delta_{ij} - \frac{e^{\eta_j}}{\sum_k e^{\eta_k}} \right) \quad (7.85)$$

$$= \mu_i (\delta_{ij} - \mu_j), \quad (7.86)$$

where δ_{ij} is equal to one if $i = j$ and zero otherwise.

Maximum likelihood estimation

In the multiway classification problem the output Y is a multinomial random variable. Recalling that in softmax regression μ_n^k denotes the posterior probability of the k th class for the n th data point, we can write the multinomial probability distribution in the following form:

$$p(y_n | x_n, \theta) = \prod_k \left(\mu_n^k \right)^{y_n^k} \quad (7.87)$$

where $\theta \triangleq (\mu_n^1, \mu_n^2, \dots, \mu_n^K)^T$ is the multinomial parameter vector. The likelihood is the product of N such probabilities. Taking the logarithm, we obtain:

$$l(\theta | \mathcal{D}) = \sum_n \sum_k y_n^k \log \mu_n^k \quad (7.88)$$

as the log likelihood for the multiway classification problem. As in the binary case, this log likelihood has the form of a cross-entropy.

To calculate the gradient of the log likelihood with respect to the parameter vector θ_i , we make use of the intermediate variable $\eta_n^i = \theta_i^T x_n$. Recalling that the derivative of μ_n^k with respect to η_n^i is nonzero because of the shared denominator in the softmax function, we have:

$$\nabla_{\theta_i} l = \sum_n \sum_k \frac{\partial l}{\partial \mu_n^k} \frac{\partial \mu_n^k}{\partial \eta_n^i} \frac{d\eta_n^i}{d\theta_i} \quad (7.89)$$

$$= \sum_n \sum_k \frac{y_n^k}{\mu_n^k} \mu_n^k (\delta_{ik} - \mu_n^i) x_n \quad (7.90)$$

$$= \sum_n \sum_k y_n^k (\delta_{ik} - \mu_n^i) x_n \quad (7.91)$$

$$= \sum_n (y_n^i - \mu_n^i) x_n, \quad (7.92)$$

where we have used the fact that $\sum_k y_n^k = 1$.

The gradient that we have obtained has the same form as the gradient for logistic regression and linear regression! (Recall Eq. (7.64) and Eq. (6.21)). We will see in Chapter 8 that this result is not a coincidence, but reflects a general property of probability distributions in the exponential family.

As in the case of logistic regression and linear regression, we obtain an on-line parameter estimation algorithm by dropping the sum over n in Eq. (7.92). This algorithm is the analog of the LMS algorithm for multiway classification.

It is straightforward to generalize the IRLS algorithm and thereby obtain a batch algorithm for softmax regression. Rather than pursuing that generalization here, we return to the IRLS algorithm in Chapter 8, where we develop a generic IRLS algorithm for the family of generalized linear models, of which softmax regression and logistic regression are examples.

7.3.3 Probit regression

In this section and the remainder of the chapter, we return to binary classification and consider some alternatives to logistic regression.

Although the logistic regression model arises naturally from a generative perspective—as the posterior probability obtained from a wide class of class-conditional probabilities—there are other choices of class-conditional probabilities that do not yield the logistic-linear form for the posterior probability. Thus, even from a generative point of view there is some motivation for exploring alternative representations for the posterior probability. In this section we engage in such an exploration within the discriminative framework, motivating alternative models “directly,” without reference to class-conditional distributions. For simplicity we retain the linearity assumption, and motivate functions other than the logistic function for converting the linear combination $\theta^T x$ to a probability scale.

One natural way to obtain a discriminative classification model is to consider “noisy threshold” models. In particular, we might suppose that a data pair (x, y) is obtained by a process in which some external agent converts the vector x to a scalar value η , defined as a linear combination $\theta^T x$, and compares the resulting value to a threshold. If the value exceeds the threshold, then the label 1 is assigned, otherwise the label 0 is assigned. A probabilistic version of this model can be obtained by assuming that the threshold is stochastic. Thus, let Z be a scalar random variable with a cumulative distribution function $F(z)$. We define:

$$p(Y = 1 | x) = p(Z \leq \eta) = F(\eta). \quad (7.93)$$

Making the further assumption that η is parameterized linearly, as $\eta \triangleq \theta^T x$, we obtain a discriminative classification model:

$$p(Y = 1 | x, \theta) = F(\theta^T x), \quad (7.94)$$

for a given distribution function F .

The logistic regression model can be interpreted as a special case of this model, given that the logistic function, $1/(1 + \exp(-x))$, is a distribution function. There is no particular reason to use a logistic random variable as the noisy threshold model, however. Indeed, given that many natural sources of “noise” have a Gaussian distribution, a common choice is to take Z to be a Gaussian random variable. This choice yields the *probit regression model*. Thus, in the probit model we have:

$$p(Y = 1 | x, \theta) = \Phi(\theta^T x), \quad (7.95)$$

where

$$\Phi(w) = \int_{-\infty}^w \frac{1}{(2\pi)^{1/2}} e^{-\frac{1}{2}\xi^2} d\xi \quad (7.96)$$

is the cumulative distribution function of a Gaussian random variable with zero mean and unit variance.⁴

Figure 7.13 shows a graphical model representation of the probit regression model. In this representation, the threshold variable Z is represented as an explicit latent variable. The graphical model requires a marginal distribution for Z , which in the probit model we take as $\mathcal{N}(0, 1)$, and a conditional distribution for Y , given X and Z . This conditional is a degenerate distribution: Y is equal to zero if $\theta^T x$ is less than Z , and one otherwise.

Figure 7.14 shows a plot of the logistic function and the Gaussian cumulative distribution function. As this plot makes clear, there is not a large difference between the two functions, and indeed probit regression and logistic regression generally give rather similar results.

Probit regression is an instance of the family of generalized linear models that we describe in Chapter 8. Maximum likelihood estimates can be obtained via stochastic gradient descent or the general version of the IRLS algorithm that we present in that chapter.

⁴The assumption of zero mean and unit variance is without loss of generality, because any linear transformation of the features can be absorbed in the parameter vector θ .

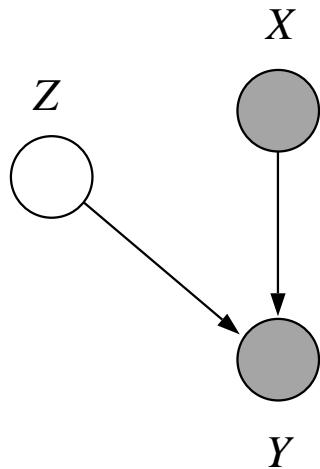


Figure 7.13: A graphical model representation of the probit regression model.

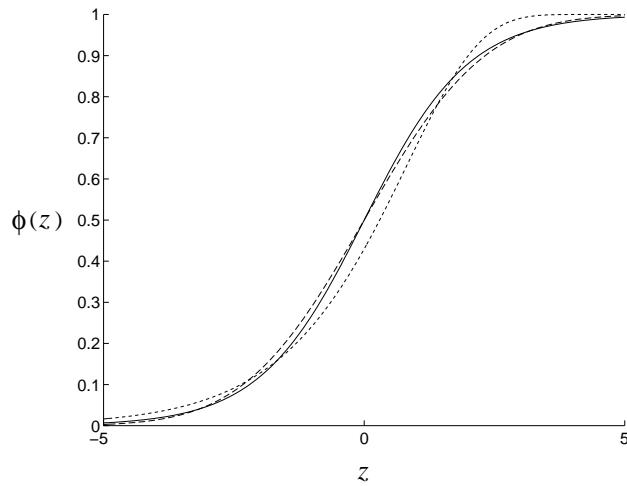


Figure 7.14: Link functions for binary classification. The solid curve is the logistic function (Eq. 7.55), the long-dashed curve is the cumulative Gaussian function (Eq. 7.96), and the small-dashed curve is the complementary log-log function (the inverse of Eq. 7.105).

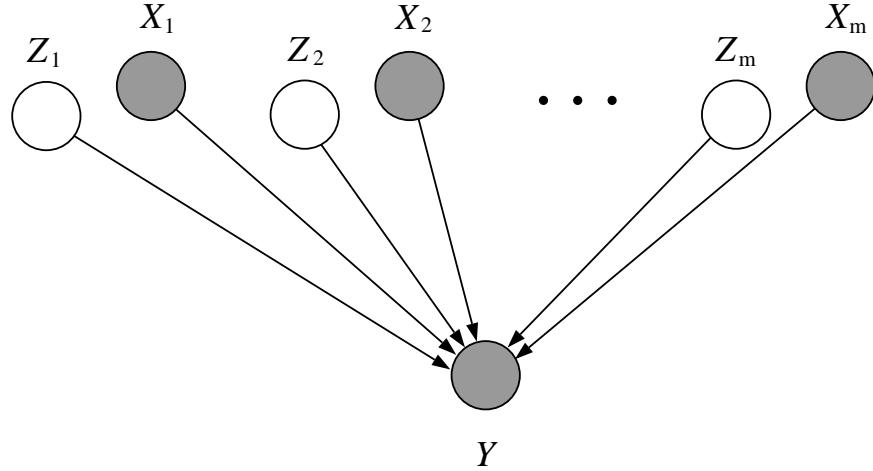


Figure 7.15: A graphical model representation of the noisy-OR model.

7.3.4 The noisy-OR model

A wide range of models can be obtained as “noisy” versions of formulas from propositional logic, in the setting in which the features X_i are binary. In this section we describe an example of this class of models known as the *noisy-OR* model. As with the other models discussed in this chapter the noisy-OR model is a linear classifier.

Let us begin with the Boolean formula:

$$Y = X_1 \vee X_2 \vee \cdots \vee X_m, \quad (7.97)$$

where $X_i \in \{0, 1\}$, for all i . To obtain a “noisy” version of the formula, let us view each variable X_i as encoding a binary “trigger” that can “cause” Y to occur. Eq. (7.97) states that the presence of any single trigger suffices to cause Y to occur. Suppose now that each trigger can “fail” with some probability ξ_i , in that the trigger can be present, but can fail to cause the occurrence of Y . Suppose moreover that the failure probabilities associated with the different triggers are independent. Thus, introducing independent binary random variables Z_i to represent the failure events, we have:

$$Y = \begin{cases} 1 & (X_1 \wedge \neg Z_1) \vee (X_2 \wedge \neg Z_2) \cdots \vee (X_m \wedge \neg Z_m) \\ 0 & \text{otherwise.} \end{cases} \quad (7.98)$$

The graphical model representing this noisy version of the logical OR formula is shown in Figure 7.15.

If we let $\xi_i \triangleq p(z_i = 1)$ denote the Bernoulli parameters associated with the Z_i , we obtain from Eq. (7.98):

$$p(Y = 0 | x, \xi) = \prod_{i=1}^m \{p(z_i = 1)\}^{x_i} = \prod_{i=1}^m \xi_i^{x_i}. \quad (7.99)$$

This formula can be interpreted as stating that the probability of Y *not* occurring is the product of the (independent) failure probabilities associated with those features x_i that are present in the input. That is, if all triggers fail, then Y doesn't occur.

To express the noisy-OR model in a linear form, let us rewrite Eq. (7.99):

$$p(Y = 0 \mid x, \xi) = \exp \left\{ \sum_{i=1}^m x_i \log \xi_i \right\}. \quad (7.100)$$

Letting $\theta_i \triangleq -\log \xi_i$, we obtain our final result:

$$p(Y = 1 \mid x, \theta) = 1 - e^{-\theta^T x} \quad (7.101)$$

for the posterior probability for the noisy-OR model.

7.3.5 Other exponential models

A number of useful classification models are based on the Poisson distribution. Recall that Z is a Poisson random variable with parameter λ if:

$$p(z \mid \lambda) = \frac{\lambda^z e^{-\lambda}}{z!}, \quad (7.102)$$

where z ranges over the nonnegative integers. Poisson variables arise in many contexts, in particular as models of counts of rarely occurring, independent events. For example, in a well-stirred solution that contains a small amount of a virus, the amount of virus in any sample might be a Poisson variable with parameter proportional to the volume of the sample. In such situations, it is often of interest to distinguish between the case in which Z takes on the value zero and the case in which Z takes on a non-zero value. (For example, a model of transmission of viral disease would want to distinguish the case that a sample of the solution contained no viral cells). Defining a binary variable Y that is equal to one in the latter case, we have:

$$p(Y = 1) = 1 - p(Z = 0) = 1 - e^{-\lambda}, \quad (7.103)$$

from Eq. (7.102). If we treat the parameter λ as a linear function of a set of input variables x , we obtain a classification model:

$$p(Y = 1 \mid x, \theta) = 1 - e^{-\theta^T x}. \quad (7.104)$$

This model is identical in form to the noisy-OR model, although the vector x is no longer restricted to be a binary vector.

An awkward aspect of the model in Eq. (7.104) is that the linear combination $\theta^T x$ must be restricted to lie between zero and infinity if we are to obtain a posterior probability that lies between zero and one. To remove this restriction, it is convenient to reparameterize the model so that the argument λ is the exponential function of some underlying variable η . We obtain a linear classification model if we assume that the underlying variable η is linear in x :

$$p(Y = 1 \mid x, \theta) = 1 - e^{-e^{\theta^T x}}. \quad (7.105)$$

An appealing feature of this model is that there are no longer any restrictions on θ . In fact, in situations involving Poisson variables it is often natural to measure the effect of the variables x on a logarithmic scale. In particular, in the example of the viral solution, x might measure the fraction of some diluting agent in the solution.

The model in Eq. (7.105) is referred to as the *complementary log-log model*. (The terminology refers to the inverse of the nonlinear function in Eq. (7.105)). Figure 7.14 includes a plot of the nonlinearity in this model. Note again the similarity to the logistic function.

7.4 Summary

We have presented a number of simple probabilistic models for discrete variables within the framework of binary and multiway classification problems. We discussed generative models, in which the discrete variable is a parent of the feature variables. We also discussed discriminative models, in which the discrete variable is a child of the feature variables. We also focused on some of the relationships between generative and discriminative models.

Maximum likelihood estimates are readily obtained in both cases. In the case of a generative model, maximum likelihood estimation essentially reduces to density estimation. That is, we find estimates of the class-conditional densities separately for each of the classes. In the discriminative setting, we model the class label as a Bernoulli or multinomial variable, which yields a cross entropy for the log likelihood. The IRLS algorithm can be used to maximize this log likelihood in the batch setting. We also presented a stochastic gradient algorithm for the on-line setting, noting the close relationship to the LMS algorithm.

All of the models that we have presented in this chapter are linear classifiers. That is, in all cases the input variable x enters into the model via a linear combination $\eta = \theta^T x$. In the generative setting this linear form was a consequence of the particular kinds of class-conditional densities that we assumed. In the discriminative setting we assumed the linear form at the outset.

Generative and discriminative models have complementary strengths and weaknesses. The generative approach allows knowledge about class-conditional densities to be exploited. If this knowledge is indeed reflective of the true data-generation process, then the generative approach can be more *efficient* than a corresponding discriminative model, in the sense that it will tend to require fewer data points. On the other hand, discriminative approaches tend to be more *robust* than generative approaches, making use of weaker assumptions regarding class-conditional densities. Note also that the discriminative framework presents a straightforward “upgrade path” toward the development of nonlinear classifiers—we can retain the logistic and softmax functions, but replace the linear combination $\eta = \theta^T x$ with a nonlinear function (see Chapter 25).

7.5 Historical remarks and bibliography

Chapter 8

The exponential family and generalized linear models

In this chapter we extend the scope of our modeling toolbox to accommodate a variety of additional data types, including counts, time intervals and rates. We introduce the exponential family of distributions, a family that includes the Gaussian, binomial, multinomial, Poisson, gamma, Rayleigh and beta distributions, as well as many others. We consider both unconditional and conditional models involving this family.

Much of our discussion is focused on the conditional setting, in which we have a directed model, $X \rightarrow Y$, with X and Y observed, and with Y having an exponential family distribution for each value of X . To parameterize this conditional distribution we introduce a class of models known as *generalized linear models (GLIM's)*. GLIM's are a general category of models that include linear regression and linear classification models as special cases. As in those models, GLIM's retain an important role for linearity, while introducing appropriate nonlinearities so as to cope with the idiosyncrasies of the particular exponential family distribution at hand. GLIM's have the dual virtue of systematizing the work that we have done thus far and showing how to extend that work to handle a wide range of additional data types.

At first blush this chapter may appear to involve a large dose of mathematical detail, but appearances shouldn't deceive—most of the detail involves working out examples that show how the exponential family and GLIM's relate to more familiar material. The real message of this chapter is the simplicity and elegance of exponential family and GLIM methods. Once the new ideas are mastered, it is often easier to work within the general exponential family and GLIM frameworks than with specific instances.

8.1 The exponential family

A probability density in the exponential family takes the following general form:

$$p(x | \eta) = h(x) \exp\{\eta^T T(x) - A(\eta)\} \quad (8.1)$$

for a parameter vector η , often referred to as the *natural parameter*, and for given functions T , a , and h . The function $T(X)$ is referred to as a *sufficient statistic*; the reasons for this nomenclature are discussed below. The form of the function $h(x)$ is not of fundamental importance; it simply reflects the underlying measure with respect to which $p(x | \eta)$ is a density. Of rather more importance is the function $A(\eta)$. Integrating Eq. (8.1) with respect to x , we have:

$$A(\eta) = \log \int h(x) \exp\{\eta^T T(x)\} dx \quad (8.2)$$

where we see that $A(\eta)$ can be viewed as the logarithm of a normalization factor. The set of η for which this integral is finite is referred to as the *natural parameter space*.

It is also common to write the exponential family distribution in the following way:

$$p(x | \eta) = \frac{1}{Z(\eta)} h(x) \exp\{\eta^T T(x)\}, \quad (8.3)$$

which is equivalent to if we let $A(\eta) = \log Z(\eta)$. Although we focus on Eq. (8.1) throughout this chapter, we will also make use of Eq. (8.72) in later chapters.

8.1.1 Examples

The Bernoulli distribution

The probability mass function of a Bernoulli random variable X is given as follows:

$$p(x | \pi) = \pi^x (1 - \pi)^{1-x} \quad (8.4)$$

$$= \exp \left\{ \log \left(\frac{\pi}{1 - \pi} \right) x + \log(1 - \pi) \right\}. \quad (8.5)$$

where our trick, here and throughout the chapter, is to take the exponential of the logarithm of the original distribution. Thus we see that the Bernoulli distribution is an exponential family distribution with:

$$\eta = \frac{\pi}{1 - \pi} \quad (8.6)$$

$$T(x) = x \quad (8.7)$$

$$A(\eta) = -\log(1 - \pi) = \log(1 + e^\eta) \quad (8.8)$$

$$h(x) = 1. \quad (8.9)$$

Note moreover that the relationship between η and π is invertible. Solving Eq. (8.6) for π , we have:

$$\pi = \frac{1}{1 + e^{-\eta}}, \quad (8.10)$$

which is the logistic function.

The Poisson distribution

The probability mass function of a Poisson random variable is given as follows:

$$p(x | \lambda) = \frac{\lambda^x e^{-\lambda}}{x!}. \quad (8.11)$$

Rewriting this expression we obtain:

$$p(x | \lambda) = \frac{1}{x!} \exp\{x \log \lambda - \lambda\}. \quad (8.12)$$

Thus the Poisson distribution is an exponential family distribution, with:

$$\eta = \log \lambda \quad (8.13)$$

$$T(x) = x \quad (8.14)$$

$$A(\eta) = \lambda = e^\eta \quad (8.15)$$

$$h(x) = \frac{1}{x!}. \quad (8.16)$$

Moreover, we can obviously invert the relationship between η and λ :

$$\lambda = e^\eta. \quad (8.17)$$

The Gaussian distribution

The (univariate) Gaussian distribution can be written as follows:

$$p(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{1}{2\sigma^2}(x - \mu)^2\right\} \quad (8.18)$$

$$= \frac{1}{\sqrt{2\pi}} \exp\left\{\frac{\mu}{\sigma^2}x - \frac{1}{2\sigma^2}x^2 - \frac{1}{2\sigma^2}\mu^2 - \ln \sigma\right\}. \quad (8.19)$$

This is in the exponential family form, with:

$$\eta = \begin{bmatrix} \mu/\sigma^2 \\ -1/2\sigma^2 \end{bmatrix} \quad (8.20)$$

$$T(x) = \begin{bmatrix} x \\ x^2 \end{bmatrix} \quad (8.21)$$

$$A(\eta) = \frac{\mu^2}{2\sigma^2} + \ln \sigma = -\frac{\eta_1^2}{4\eta_2} - \frac{1}{2} \ln(-2\eta_2) \quad (8.22)$$

$$h(x) = \frac{1}{\sqrt{2\pi}}. \quad (8.23)$$

Note in particular that the univariate Gaussian distribution is a two-parameter distribution and that its sufficient statistic is a vector.

The multivariate Gaussian distribution can also be written in the exponential family form; we leave the details to Exercise ?? and Chapter 13.

The multinomial distribution

As a final example, let us consider the multinomial distribution. Let $X = (X_1, X_2, \dots, X_m)$ be a collection of integer-valued random variables representing event counts, where X_i represents the count of the number of times the i th event occurs in a set of n independent trials. Let π_i represent the probability of the i th event occurring in any given trial. We have:

$$p(x | \pi) = \frac{n!}{x_1! x_2! \cdots x_m!} \pi_1^{x_1} \pi_2^{x_2} \cdots \pi_m^{x_m}, \quad (8.24)$$

as the probability mass function for such a collection.

Following the strategy of our previous examples, we rewrite the multinomial distribution as follows:

$$p(x | \pi) = \exp \left\{ \sum_{i=1}^m x_i \ln \pi_i \right\}. \quad (8.25)$$

While this shows that the multinomial distribution is in the exponential family, there are some troubling aspects to this expression. In particular it appears that the $A(\eta)$ term is equal to zero. As we will be seeing (in Section 8.1.2), one of the principal virtues of the exponential family form is that moments can be calculated by taking derivatives of $A(\eta)$; thus, the disappearance of this term is unsettling.

Our problem is caused by the fact that the parameters satisfy a linear constraint, namely: $\sum_{i=1}^m \pi_i = 1$. Let us define an exponential family to be of *full rank* if the parameters satisfy no such constraint—technically we assume that an m -dimensional parameter space contains an open rectangle of dimension m . (In the case of the multinomial the parameters lie on an $m - 1$ dimensional simplex, and thus the parameter space does not contain a rectangle of dimension m). To achieve a full rank representation for the multinomial, we parameterize the distribution using the first $m - 1$ components of π :

$$p(x | \pi) = \exp \left\{ \sum_{i=1}^m x_i \ln \pi_i \right\} \quad (8.26)$$

$$= \exp \left\{ \sum_{i=1}^{m-1} x_i \ln \pi_i + \left(1 - \sum_{i=1}^{m-1} x_i \right) \ln \left(1 - \sum_{i=1}^{m-1} \pi_i \right) \right\} \quad (8.27)$$

$$= \exp \left\{ \sum_{i=1}^{m-1} \ln \left(\frac{\pi_i}{1 - \sum_{i=1}^{m-1} \pi_i} \right) x_i + \ln \left(1 - \sum_{i=1}^{m-1} \pi_i \right) \right\}. \quad (8.28)$$

where we have used the fact that $\pi_m = 1 - \sum_{i=1}^{m-1} \pi_i$.

From this representation we obtain:

$$\eta_i = \ln \left(\frac{\pi_i}{1 - \sum_{i=1}^{m-1} \pi_i} \right) = \ln \left(\frac{\pi_i}{\pi_m} \right) \quad (8.29)$$

for $i = 1, \dots, m - 1$. For convenience we also can define η_m ; Eq. (8.29) implies that if we do so we must take $\eta_m = 0$.

As in the other examples of exponential family distributions, we can invert Eq. (8.29) to obtain a mapping that expresses π_i in terms of η_i . Taking the exponential of Eq. (8.29) and summing we obtain:

$$\pi_i = \frac{e^{\eta_i}}{\sum_{j=1}^m e^{\eta_j}}, \quad (8.30)$$

which is the softmax function.

Finally, from Eq. (8.28) we obtain:

$$A(\eta) = -\ln \left(1 - \sum_{i=1}^{m-1} \pi_i \right) = \ln \left(\sum_{i=1}^m e^{\eta_i} \right) \quad (8.31)$$

as the log normalization factor for the multinomial.

8.1.2 Moments

An appealing feature of the exponential family representation is that we can obtain moments of the distribution by taking derivatives of the log normalization function $A(\eta)$. Before establishing this fact, let us consider an example.

Recall that in the case of the Bernoulli distribution we have $A(\eta) = \log(1 + e^\eta)$. Taking a first derivative yields:

$$\frac{dA}{d\eta} = \frac{e^\eta}{1 + e^\eta} \quad (8.32)$$

$$= \frac{1}{1 + e^{-\eta}} \quad (8.33)$$

$$= \mu, \quad (8.34)$$

which is the mean of a Bernoulli variable.

Taking a second derivative yields:

$$\frac{d^2 a}{d\eta^2} = \frac{d\mu}{d\eta} \quad (8.35)$$

$$= \mu(1 - \mu), \quad (8.36)$$

which is the variance of a Bernoulli variable.

We now show that in general the first derivative of $A(\eta)$ is equal to the mean of $T(X)$. We treat the case of scalar η for simplicity; the (straightforward) extension to vector η is considered in Exercise ???. Calculating the first derivative of $A(\eta)$ yields:

$$\frac{dA}{d\eta} = \frac{d}{d\eta} \left\{ \log \int \exp\{\eta T(x)\} h(x) dx \right\} \quad (8.37)$$

$$= \frac{\int T(x) \exp\{\eta T(x)\} h(x) dx}{\int \exp\{\eta T(x)\} h(x) dx} \quad (8.38)$$

$$= \int T(x) \exp\{\eta^T T(x) - A(\eta)\} h(x) dx \quad (8.39)$$

$$= ET(X). \quad (8.40)$$

Thus we see that the first derivative of $A(\eta)$ is equal to the mean of the sufficient statistic.

Let us now take a second derivative:

$$\frac{d^2 a}{d\eta^2} = \int T(x) \exp\{\eta T(x) - A(\eta)\}(T(x) - a'(\eta))h(x)dx \quad (8.41)$$

$$= \int T(x) \exp\{\eta T(x) - A(\eta)\}(T(x) - ET(X))h(x)dx \quad (8.42)$$

$$= \int T^2(x) \exp\{\eta T(x) - A(\eta)\}h(x)dx - ET(X) \int T(x) \exp\{\eta T(x) - A(\eta)\}h(x)dx \\ = ET^2(x) - (ET(X))^2 \quad (8.43)$$

$$= \text{Var}[T(x)], \quad (8.44)$$

and thus we see that the second derivative of $A(\eta)$ is equal to the variance of the sufficient statistic.

Example

Let us calculate the moments of the univariate Gaussian distribution. Recall the form taken by $A(\eta)$:

$$A(\eta) = -\frac{\eta_1^2}{4\eta_2} - \frac{1}{2} \ln(-2\eta_2), \quad (8.45)$$

where $\eta_1 = \mu/\sigma^2$ and $\eta_2 = -1/2\sigma^2$.

Taking the derivative with respect to η_1 yields:

$$\frac{\partial A}{\partial \eta_1} = \frac{\eta_1}{2\eta_2} \quad (8.46)$$

$$= \frac{\mu/\sigma^2}{1/\sigma^2} \quad (8.47)$$

$$= \mu, \quad (8.48)$$

which is the mean of X , the first component of the sufficient statistic.

Taking a second derivative with respect to η_1 yields:

$$\frac{\partial^2 A}{\partial \eta_1^2} = -\frac{1}{2\eta_2} \quad (8.49)$$

$$= \sigma^2, \quad (8.50)$$

which is the variance of X .

Given that X^2 is the second component of the sufficient statistic, we can also compute the variance by calculating the partial of a with respect to η_2 . Moreover, we can calculate third moments by computing the mixed partial, and fourth moments by taking the second partial with respect to η_2 (see Exercise ??).

8.1.3 The moment parameterization

In the previous section we have seen that it is possible to obtain the mean, $\mu \triangleq ET(X)$, as a function of the canonical parameter η :

$$\mu = \frac{dA}{d\eta}. \quad (8.51)$$

It turns out that this relationship is invertible.

To see this, note from Eq. (8.44) that the second derivative of $A(\eta)$ is a variance and hence positive. This implies that $A(\eta)$ is a convex function. For a convex function there is necessarily a one-to-one relationship between the argument to the function and the first derivative of the function. Hence the mapping from η to μ is invertible.

We will represent the inverse mapping as $\eta = \psi(\mu)$ in the remainder of the chapter.

This argument implies that a distribution in the exponential family can be parameterized not only by η —the canonical parameterization—but also by μ —the *moment parameterization*. Many distributions are traditionally parameterized using the moment parameterization; indeed, in Section 8.1.1 our starting point was the moment parameterization for each of the examples. We subsequently reparameterized these distribution using the canonical parameterization. We also computed the mapping from η to μ in each case, recovering some familiar functions, including the logistic function and the softmax function. We will return to this topic in Section 8.2 when we discuss generalized linear models.

8.1.4 Sufficiency

In this section we discuss the important concept of *sufficiency*. Sufficiency characterizes what is essential in a data set, or, alternatively, what is inessential and can therefore be thrown away. While the notion of sufficiency is broader than the exponential family, the ties to the exponential family are close, and it is natural to introduce the concept here.

A *statistic* is a function of a random variable. In particular, let X be a random variable and let $T(X)$ be a statistic. Suppose that the distribution of X depends on a parameter θ . The intuitive notion of sufficiency is that $T(X)$ is sufficient for θ if there is no information in X regarding θ beyond that in $T(X)$. That is, having observed $T(X)$, we can throw away X for the purposes of inference with respect to θ . Let us make this notion more precise.

Sufficiency is defined in somewhat different ways in the Bayesian and frequentist frameworks. Let us begin with the Bayesian approach, which is arguably more natural. In the Bayesian approach, we treat θ as a random variable, and are therefore licensed to consider conditional independence relationships involving θ . We say that $T(X)$ is sufficient for θ if the following conditional independence statement holds:

$$\theta \perp\!\!\!\perp X \mid T(X). \quad (8.52)$$

We can also write this in terms of probability distributions:

$$p(\theta \mid T(x), x) = p(\theta \mid T(x)). \quad (8.53)$$

Thus, as shown graphically in Figure 8.1(a), sufficiency means that θ is independent of X , when

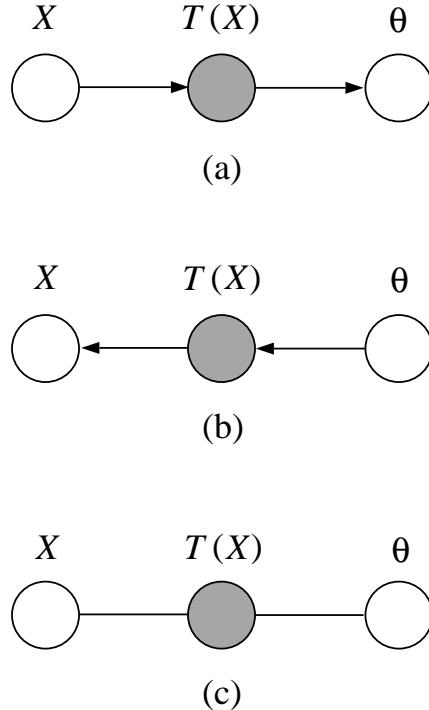


Figure 8.1: Graphical models whose conditional independence properties capture the notion of sufficiency in three equivalent ways.

we condition on $T(X)$. This captures the intuitive notion that $T(X)$ contains all of the essential information in X regarding θ .

To obtain a frequentist definition of sufficiency, let us consider the graphical model in Figure 8.1(b). This model expresses the same conditional independence semantics as Figure 8.1(a), asserting that θ is independent of X conditional on $T(X)$, but the model is parameterized in a different way. From the factorized form of the joint probability we obtain:

$$p(x | T(x), \theta) = p(x | T(x)). \quad (8.54)$$

This expression suggests a frequentist definition of sufficiency. In particular, treating θ as a label rather than a random variable, we define $T(X)$ to be sufficient for θ if the conditional distribution of X given $T(X)$ is not a function of θ .

Both the Bayesian and frequentist definitions of sufficiency imply a factorization of $p(x | \theta)$, and it is this factorization which is generally easiest to work with in practice. To obtain the factorization we use the undirected graphical model formalism. Note in particular that Figure 8.1(c) expresses the same conditional independence semantics as Figure 8.1(a) and Figure 8.1(b). Moreover, from Figure 8.1(c), we know that we can express the joint probability as a product of potential functions ψ_1 and ψ_2 :

$$p(x, T(x), \theta) = \psi_1(T(x), \theta)\psi_2(x, T(x)), \quad (8.55)$$

where we have absorbed the constant of proportionality Z in one of the potential functions. Now $T(x)$ is a deterministic function of x , which implies that we can drop $T(x)$ on the left-hand side of the equation. Dividing by $p(\theta)$ we therefore obtain:

$$p(x | \theta) = g(T(x), \theta)h(x, T(x)), \quad (8.56)$$

for given functions g and h . Although we have motivated this result by using a Bayesian calculation, the result can be utilized within either the Bayesian or frequentist framework. Its equivalence to the frequentist definition of sufficiency is known as the Neyman factorization theorem.

8.1.5 Sufficiency and the exponential family

An important feature of the exponential family is that one can obtain sufficient statistics by inspection, once the distribution is expressed in the standard form. Recall the definition:

$$p(x | \eta) = h(x) \exp\{\eta^T T(x) - A(\eta)\}. \quad (8.57)$$

From Eq. (8.56) we see immediately that $T(X)$ is a sufficient statistic for η .

8.1.6 IID sampling

The reduction obtainable by using a sufficient statistic is particularly notable in the case of IID sampling. Suppose that we have a collection of N independent random variables, $X = (X_1, X_2, \dots, X_N)$, characterized by the same exponential family density. Taking the product, we obtain the joint density:

$$p(x | \eta) = \prod_{n=1}^N h(x_n) \exp\{\eta^T T(x_n) - A(\eta)\} = \left(\prod_{n=1}^N h(x_n) \right) \exp \left\{ \eta^T \sum_{n=1}^N T(x_n) - NA(\eta) \right\}. \quad (8.58)$$

From this result we see that X is itself an exponential distribution, with sufficient statistic $\sum_{n=1}^N T(x_n)$.

For several of the examples we discussed earlier (in Section 8.1.1), including the Bernoulli, the Poisson, and the multinomial distributions, the sufficient statistic $T(X)$ is equal to the random variable X . For a set of N IID observations from such distributions, the sufficient statistic is equal to $\sum_{n=1}^N x_n$. Thus in this case, it suffices to maintain a single value, the sum of the observations. The individual data points can be thrown away.

For the univariate Gaussian the sufficient statistic is the pair $T(X) = (X, X^2)$, and thus for N IID Gaussians it suffices to maintain the sum $\sum_{n=1}^N x_n$, and the sum of squares $\sum_{n=1}^N x_n^2$.

8.1.7 Maximum likelihood estimates

In this section we show how to obtain maximum likelihood estimates in exponential family distributions. We obtain a generic formula which generalizes our earlier work on density estimation in Chapter 5.

Consider an IID data set, $\mathcal{D} = (x_1, x_2, \dots, x_N)$. From Eq. (8.58) we obtain the following log likelihood:

$$l(\eta | \mathcal{D}) = \log \left(\prod_{n=1}^N h(x_n) \right) + \eta^T \left(\sum_{n=1}^N T(x_n) \right) - NA(\eta). \quad (8.59)$$

Taking the gradient with respect to η yields:

$$\nabla_\eta l = \sum_{n=1}^N T(x_n) - N \nabla_\eta A(\eta), \quad (8.60)$$

and setting to zero gives:

$$\nabla_\eta A(\hat{\eta}) = \frac{1}{N} \sum_{n=1}^N T(x_n). \quad (8.61)$$

Finally, defining $\mu \triangleq E[T(x)]$, and recalling Eq. (8.40), we obtain:

$$\hat{\mu}_{ML} = \frac{1}{N} \sum_{n=1}^N T(x_n) \quad (8.62)$$

as the general formula for maximum likelihood estimation in the exponential family.

It should not be surprising that our formula involves the data only via the sufficient statistic $\sum_{n=1}^N T(x_n)$. This gives operational meaning to sufficiency—for the purpose of estimating parameters we retain only the sufficient statistic.

For distributions in which $T(X) = X$, which include the Bernoulli distribution, the Poisson distribution, and the multinomial distribution, our result shows that the sample mean is the maximum likelihood estimate of the mean.

For the univariate Gaussian distribution, we see that the sample mean is the maximum likelihood estimate of the mean and the sample variance is the maximum likelihood estimate of the variance. For the multivariate Gaussian we obtain the same result, where by “variance” we mean the covariance matrix.

8.1.8 Maximum likelihood and the Kullback-Leibler divergence

In this section we point out a simple relationship between the maximum likelihood problem and the Kullback-Leibler (KL) divergence. This relationship is general; it has nothing to do specifically with the exponential family. We discuss it in the current chapter, however, because we have a hidden agenda. Our agenda, to be gradually revealed in Chapters 9, 11 and 20, involves building a number of very interesting and important relationships between the exponential family and the Kullback-Leibler (KL) divergence. By introducing a statistical interpretation of the KL divergence in the current chapter, we hope to hint subliminally at deeper connections to come.

To link the KL divergence and maximum likelihood, let us first define the *empirical distribution*, $\tilde{p}(x)$. This is a distribution which places a point mass at each data point x_n in our data set \mathcal{D} . We

have:

$$\tilde{p}(x) \triangleq \frac{1}{N} \sum_{n=1}^N \delta(x, x_n), \quad (8.63)$$

where $\delta(x, x_n)$ is a Kronecker delta function in the continuous case. In the discrete case, $\delta(x, x_n)$ is simply a function that is equal to one if its arguments agree and equal to zero otherwise.

If we integrate (in the continuous case) or sum (in the discrete case) $\tilde{p}(x)$ against a function of x , we evaluate that function at each point x_n . In particular, the log likelihood can be written this way. In the discrete case we have:

$$\sum_x \tilde{p}(x) \log p(x | \theta) = \sum_x \frac{1}{N} \sum_{n=1}^N \delta(x, x_n) \log p(x | \theta) \quad (8.64)$$

$$= \frac{1}{N} \sum_{n=1}^N \sum_x \delta(x, x_n) \log p(x | \theta) \quad (8.65)$$

$$= \frac{1}{N} \sum_{n=1}^N \log p(x_n | \theta) \quad (8.66)$$

$$= \frac{1}{N} l(\theta | \mathcal{D}). \quad (8.67)$$

Thus by computing a cross-entropy between the empirical distribution and the model, we obtain the log likelihood, scaled by the constant $1/N$. We obtain an identical result in the continuous case by integrating.

Let us now calculate the KL divergence between the empirical distribution and the model $p(x | \theta)$. We have:

$$D(\tilde{p}(x) \| p(x | \theta)) = \sum_x \tilde{p}(x) \log \frac{\tilde{p}(x)}{p(x | \theta)} \quad (8.68)$$

$$= \sum_x \tilde{p}(x) \log \tilde{p}(x) - \sum_x \tilde{p}(x) \log p(x | \theta) \quad (8.69)$$

$$= + \sum_x \tilde{p}(x) \log \tilde{p}(x) - \frac{1}{N} l(\theta | \mathcal{D}). \quad (8.70)$$

The first term, $\sum_x \tilde{p}(x) \log \tilde{p}(x)$, is independent of θ . Thus, the minimizing value of θ on the left-hand side is equal to the maximizing value of θ on the right-hand side.

In other words: *minimizing the KL divergence to the empirical distribution is equivalent to maximizing the likelihood*. This simple result will prove to be very useful in our later work.

8.1.9 Conjugacy and Bayesian estimates

[Section not yet written].

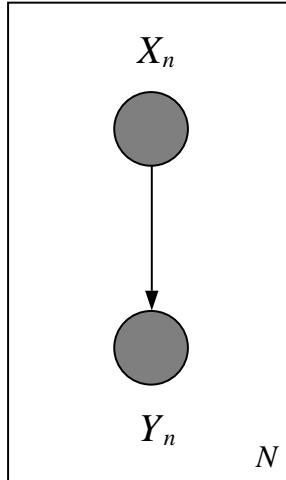


Figure 8.2: The graphical model representation of a generalized linear model.

8.2 Generalized linear models

We now turn to problems involving a pair of variables, X and Y , where both X and Y are assumed to be observed (see Figure 8.2). As in the linear regression and (discriminative) linear classification models that we discussed in Chapters 6 and 7, we focus on the conditional relationship between X and Y .

A common feature of both the linear regression and discriminative linear classification models is a particular choice of representation for the conditional expectation of Y . Letting μ denote the modeled value of the conditional expectation, we can summarize the structural component of both types of models by writing:

$$\mu = f(\theta^T x). \quad (8.71)$$

In the case of linear regression the function $f(\cdot)$ is the identity function. For the linear classification models, we studied a variety of possible choices for $f(\cdot)$, including the logistic function (for logistic regression) and the cumulative Gaussian (for probit regression).

To complete the model specification, we endow Y with a particular conditional probability distribution, having μ as a parameter. For linear regression, this distribution is Gaussian, whereas for the linear classification models, this distribution is Bernoulli (for the binary case) or multinomial (for the multiway case).

The *generalized linear model (GLIM)* framework extends these ideas beyond the Gaussian, Bernoulli and multinomial settings to the more general exponential family. A GLIM makes three assumptions regarding the form of the conditional probability distribution $p(y | x)$:

- The observed input x is assumed to enter into the model via a linear combination $\xi = \theta^T x$,
- The conditional mean μ is represented as a function $f(\xi)$ of the linear combination ξ , where f is known as the *response function*,

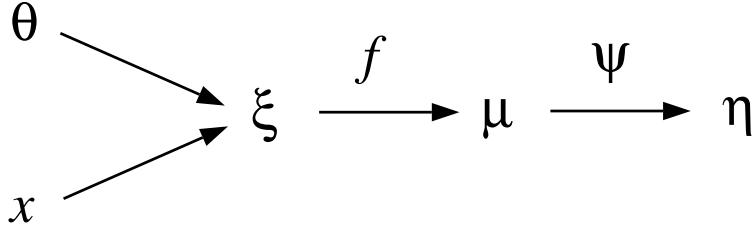


Figure 8.3: A diagram summarizing the relationships between the variables in a GLIM model.

- The observed output y is assumed to be characterized by an exponential family distribution with conditional mean μ .

These assumptions are summarized diagrammatically in Figure 8.3. Note that the diagram includes the mapping from μ to η , which we denote as $\eta = \psi(\mu)$. This mapping allows us to use the canonical parameterization to represent the exponential family distribution for Y .

Within the GLIM framework, it is convenient to work with a slight variation on the exponential family theme. In particular, in the GLIM framework we assume that the conditional distribution of Y takes the following form:

$$p(x | \eta, \phi) = h(x, \phi) \exp \left\{ \frac{\eta^T x - A(\eta)}{\phi} \right\}, \quad (8.72)$$

where we have augmented the representation in Eq. (8.1) to include an explicit *scale parameter* ϕ . Many exponential family distributions, including the Gaussian and the gamma, are naturally expressed in this form. In particular, we can write the Gaussian distribution as follows:

$$p(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp \left\{ -\frac{1}{2\sigma^2}(x - \mu)^2 \right\} \quad (8.73)$$

$$= \left(\frac{1}{\sqrt{2\pi}\sigma^2} \exp \left\{ -\frac{x^2}{2\sigma^2} \right\} \right) \exp \left\{ \frac{\mu x - \mu^2/2}{\sigma^2} \right\}. \quad (8.74)$$

As we have seen, we can equivalently bundle the parameters μ and σ^2 into a single parameter vector, and express the Gaussian as a two-parameter exponential family distribution. The scale-parameter form is, however, often more natural. Note, moreover, that although the Gaussian yields a two-parameter exponential family when we bundle the canonical parameter and the scale parameter, in general we do not require that $p(x | \eta, \phi)$ is expressible as a two-parameter exponential family. This gives some useful flexibility.

Note also that for the purposes of GLIM modeling we drop the $T(\cdot)$ function in the exponential family representation. Thus we focus on distributions for which the observable Y is itself a sufficient statistic.

There are two principal choice points in the specification of a GLIM: (1) the choice of exponential family distribution, and (2) the choice of the response function $f(\cdot)$.

The choice of exponential family distribution is generally rather strongly constrained by the nature of the data Y . Thus, class labels are naturally represented by Bernoulli or multinomial distributions, counts by the Poisson distribution, intervals by the exponential or gamma distributions, etc.

This leaves us with the choice of the response function as the principal degree of freedom in the specification of a GLIM. There are constraints that we generally want to impose on this function, reflecting constraints on the conditional expectation. For example, in the case of the Bernoulli and multinomial distributions, the conditional expectation must lie between 0 and 1, and this suggests that we should choose a response function whose range is $(0, 1)$. Similarly, for a gamma distribution, the random variable is nonnegative, and we should presumably choose a response function whose range is $(0, \infty)$. Such constraints only give rough guidance, however, and in general for any given distribution there are many possible choices of response function. There is, however, a particular response function—the *canonical response function*—that is uniquely associated with a given exponential family distribution and has some appealing mathematical properties. In particular, if we assume that $\xi = \eta$, or equivalently that $f(\cdot) = \psi^{-1}(\cdot)$, we obtain the canonical response function. Note that the function $\psi(\cdot)$ is determined once we have chosen a particular exponential family density. Thus if we decide to use the canonical response function the choice of the exponential family density completely determines the GLIM.

We will explore some of the properties of the canonical response function in the remainder of this chapter. It should be emphasized, however, that the canonical response function is by no means the universally best choice for all problems. Indeed, as we have already seen in the case of the classification models, different choices of response function can be appropriate in different situations, reflecting different underlying assumptions about the way that the data are generated. We might view the canonical response function as a reasonable default.

The first point to note about the canonical response function is that it automatically passes a sanity check with regards to the constraints on its range. That is, the modeled values $\mu = f(\eta)$ are guaranteed to be possible values of the conditional expectation. To see this, note that:

$$f(\eta) = \psi^{-1}(\eta) = a'(\eta) = E[Y | \eta]. \quad (8.75)$$

Thus, for any value η such that $a'(\eta)$ exists, we see that $f(\eta)$ is equal to the conditional mean of an exponential family distribution in which η is the canonical parameter.

Some examples of canonical response functions are provided in the following table; these have been collected from the examples in Section 8.1 and from the exercises.

8.2.1 Maximum likelihood estimation

In this section we write down the likelihood for generalized linear models and present on-line and batch methods for maximizing the likelihood. We restrict ourselves to scalar Y in order to simplify the presentation; the results go through for vector Y with straightforward notational alterations (see Exercise ??).

Consider an IID data set, $\mathcal{D} = \{(x_n, y_n); n = 1, \dots, N\}$. Taking the logarithm of a product of N copies of the exponential family distribution in Eq. (8.72), we obtain the following log likelihood

tb

Model	Canonical response function
Gaussian	$\mu = \eta$
Bernoulli	$\mu = 1/(1 + e^{-\eta})$
multinomial	$\mu_i = \eta_i / \sum_j e^{\eta_j}$
Poisson	$\mu = e^\eta$
gamma	$\mu = -\eta^{-1}$

Figure 8.4: The canonical response functions for several exponential family distributions.

for GLIM models:

$$l(\theta | \mathcal{D}) = \log \left(\prod_{n=1}^N h(y_n) \exp\{\eta_n y_n - A(\eta_n)\} \right) \quad (8.76)$$

$$= \sum_{n=1}^N \log h(y_n) + \sum_{n=1}^N (\eta_n y_n - A(\eta_n)), \quad (8.77)$$

where $\eta_n = \psi(\mu_n)$, $\mu_n = f(\xi_n)$ and $\xi_n = \theta^T x_n$.

In the case of the canonical response function, for which $\eta_n = \theta^T x_n$, the log likelihood simplifies:

$$l(\theta | \mathcal{D}) = \sum_{n=1}^N \log h(y_n) + \sum_{n=1}^N (\theta^T x_n y_n - A(\eta_n)) \quad (8.78)$$

$$= \sum_{n=1}^N \log h(y_n) + \theta^T \sum_{n=1}^N x_n y_n - \sum_{n=1}^N A(\eta_n). \quad (8.79)$$

From this expression we can draw an important conclusion—*the sum $\sum_{n=1}^N x_n y_n$ is a sufficient statistic for θ* . This sum has a fixed, finite dimension (the dimension of the vector x_n), for any value of N . This has the very practical consequence that we can allocate a fixed amount of storage for collecting the information needed to estimate θ , whatever the sample size N . This is an important motivation for considering canonical response functions.

Let us now calculate the gradient of the log likelihood:

$$\nabla_\theta l = \sum_{n=1}^N \frac{dl}{d\eta_n} \nabla_\theta \eta_n \quad (8.80)$$

$$= \sum_{n=1}^N (y_n - a'(\eta_n)) \nabla_\theta \eta_n \quad (8.81)$$

$$= \sum_{n=1}^N (y_n - \mu_n) \frac{d\eta_n}{d\mu_n} \frac{d\mu_n}{d\xi_n} x_n. \quad (8.82)$$

For the canonical response function, we have $\eta_n = \xi_n$, and thus the derivatives cancel, leaving us with the following simple expression for the log likelihood:

$$\nabla_{\theta} l = \sum_{n=1}^N (y_n - \mu_n) x_n. \quad (8.83)$$

This expression has the appealing feature that the parameter vector θ and the “error” $(y_n - \mu_n)$ are on the same scale.

An on-line algorithm

A general on-line estimation algorithm can be obtained by following the stochastic gradient of the log likelihood function. Consider first the case of the canonical response function. Given an estimate $\theta^{(t)}$ at the t th iteration of the algorithm, we obtain:

$$\theta^{(t+1)} = \theta^{(t)} + \rho(y_n - \mu_n^{(t)}) x_n, \quad (8.84)$$

where $\mu_n^{(t)} = f(\theta^{(t)T} x_n)$ and where ρ is a step size.

We have obtained an algorithm that is formally identical to the LMS algorithm. Moreover, the geometry of the LMS algorithm that we discussed in Chapter 6 carries over to this more general setting. That is, as in Chapter 6, the on-line algorithm steps in the direction of the input vector x_n , weighted by the prediction error $(y_n - \mu_n^{(t)})$. The specific GLIM model makes its appearance only through the definition of the conditional expectation μ_n .

If we do not use the canonical response function, then the gradient also includes the derivatives of $f(\cdot)$ and $\psi(\cdot)$. These can be viewed as scaling coefficients that alter the step size ρ , but otherwise leave the general LMS form intact. Thus we have obtained a result worth remembering—the LMS-like algorithm in Eq. (8.84) is the generic stochastic gradient algorithm for models throughout the GLIM family.

A batch algorithm

In our discussion of logistic regression (Section 7.3.1) we introduced the *iteratively reweighted least squares (IRLS) algorithm*—a Newton-Raphson algorithm for batch estimation of parameters. The algorithm goes through with essentially no change to the general GLIM setting. For completeness we present the algorithm and its derivation here.

We will assume the canonical response function, and indicate the changes that are needed to accommodate noncanonical response functions at the end of the section.

We begin by writing the gradient in vector notation:

$$\nabla_{\theta} l = \frac{1}{\phi} \sum_n (y_n - \mu_n) x_n = \frac{1}{\phi} X^T (y - \mu), \quad (8.85)$$

where X is the design matrix whose rows are the vector x_n^T , y is defined as the $N \times 1$ vector whose components are the values y_n , and similarly μ is the $N \times 1$ vector whose components are the values μ_n .

Taking a second derivative, we calculate the Hessian matrix:

$$H = -\frac{1}{\phi} \sum_n \frac{d\mu_n}{d\eta_n} x_n x_n^T \quad (8.86)$$

$$= -\frac{1}{\phi} X^T W X, \quad (8.87)$$

where we have defined the diagonal weight matrix:

$$W \triangleq \text{diag} \left\{ \frac{d\mu_1}{d\eta_1}, \frac{d\mu_1}{d\eta_2}, \dots, \frac{d\mu_1}{d\eta_N} \right\}, \quad (8.88)$$

where $d\mu_n/d\eta_n$ can be computed by calculating the second derivative of $A(\eta_n)$.

Note that the weights depend on the parameter vector θ , and thus the weight matrix W depends on θ . We use the notation $W^{(t)}$ to denote the weight matrix at the t th iteration of the algorithm. Similarly, we use the notation $\mu^{(t)}$ to denote the value of μ at the t th iteration.

The Newton-Raphson algorithm is obtained by multiplying the gradient by the inverse Hessian matrix and subtracting the result from the current parameter vector:

$$\theta^{(t+1)} = \theta^{(t)} + (X^T W^{(t)} X)^{-1} X^T (y - \mu^{(t)}) \quad (8.89)$$

$$= (X^T W^{(t)} X)^{-1} \left[X^T W^{(t)} X \theta^{(t)} + X^T (y - \mu^{(t)}) \right] \quad (8.90)$$

$$= (X^T W^{(t)} X)^{-1} X^T W^{(t)} z^{(t)}, \quad (8.91)$$

where we define:

$$z^{(t)} = \eta + [W^{(t)}]^{-1} (y - \mu^{(t)}). \quad (8.92)$$

The algorithm in Eq. (8.91) is the IRLS algorithm.

If we extend the derivation to handle noncanonical response functions, we find that the Hessian matrix has another term (see Exercise ??). Including this term yields the Newton-Raphson algorithm for noncanonical response functions. There is, however, an alternative approach. If we use the *expected Hessian* in place of the Hessian in the Newton-Raphson update formula, we obtain an alternative algorithm known as the *Fisher scoring method*.¹ This algorithm is a simplification in the case of noncanonical response functions—the extra term that appears in the Hessian contains the factor $(y - \mu)$, and this term therefore vanishes when we take expectations (see Exercise ??). Thus, the Fisher scoring method takes the form shown in Eq. (8.91) in all cases. It is generally the preferred way to implement the IRLS algorithm.

8.3 Historical remarks and bibliography

¹Note that the expectation of the Hessian matrix is the Fisher information matrix.

Chapter 9

Completely Observed Graphical Models

The models that we have discussed until now have, for the most part, involved a single node or a single node and its parents. We have seen how to find maximum likelihood estimates for a variety of models of this form.

In the current chapter, we learn that the techniques that we have developed extend with essentially no additional labor to the entire class of directed graphical models, under the assumption of *complete observations*. Thus, if we assume that our data set assigns values to all of the random variables in the model—i.e., that there are no *latent variables*—then we can find maximum likelihood estimates of parameters in a straightforward way, essentially by solving the problem separately at each node. The parameter estimation problem “decouples.” The underlying reason for this appealing result is the product form of the joint probability distribution.

We also discuss the problem of maximum likelihood parameter estimation for undirected graphical models. In this case the parameter estimation problem decouples only for a special class of graphical models, known as *decomposable models*. For general undirected models the parameter estimation problem does not decouple, the essential reason being the presence of the global normalization factor Z . Nonetheless, we will show that there is a local characterization of maximum likelihood estimates for general undirected models, and we present algorithms for finding these estimates.

In Section 9.4, we discuss the problems that arise when we remove the restriction to completely observed models. The material in this section sets the stage for our treatment of latent variable models in Chapters 10 through 15.

Our focus in this chapter is on the likelihood function and maximum likelihood estimation. We do, however, provide a short discussion of Bayesian estimation in the completely observed setting in Section ??.

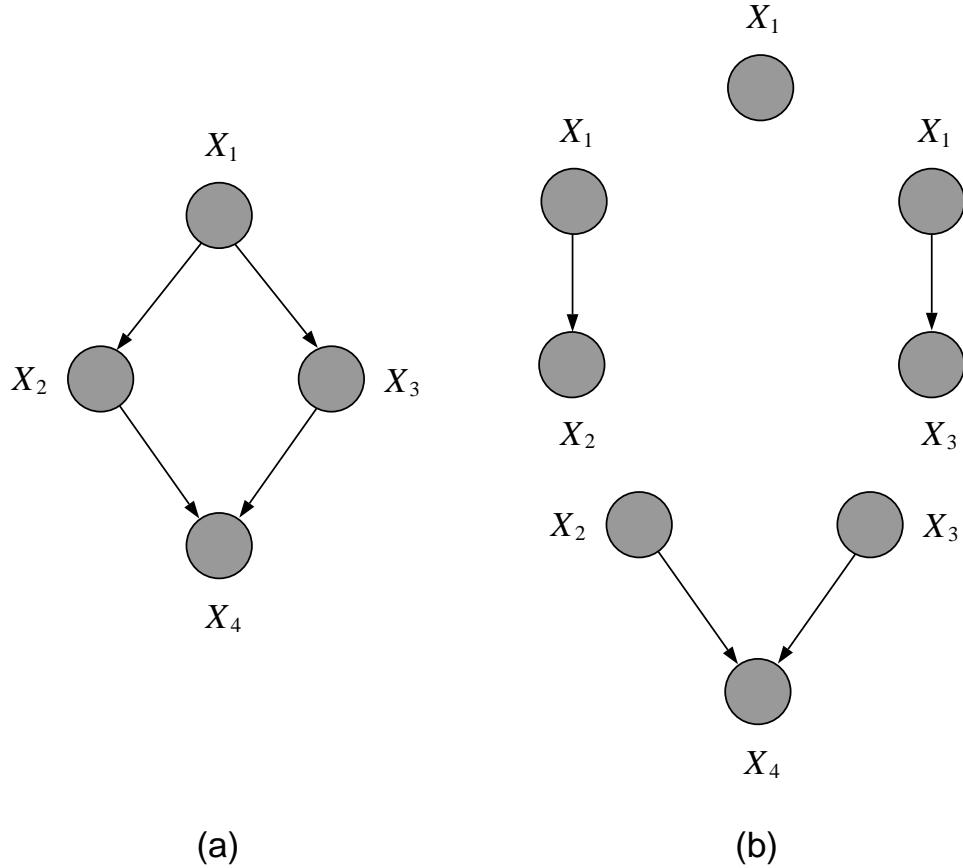


Figure 9.1: (a) A directed graphical model. (b) Solving the maximum likelihood problem in (a) is equivalent to solving separate maximum likelihood problems for each node conditioned on its parents.

9.1 The basic idea

Before jumping into the details, let us consider a simple example. The graphical model shown in Figure 9.1(a) has the following joint probability distribution:

$$p(x | \theta) = p(x_1 | \theta_1)p(x_2 | x_1, \theta_2)p(x_3 | x_1, \theta_3)p(x_4 | x_2, x_3, \theta_4). \quad (9.1)$$

Taking the logarithm of this distribution we have:

$$\log p(x|\theta) = \log p(x_1|\theta_1) + \log p(x_2|x_1, \theta_2) + \log p(x_3|x_1, \theta_3) + \log p(x_4|x_2, x_3, \theta_4). \quad (9.2)$$

From this expression, we see that the parameters, θ_i , appear in different terms, and thus the maximization of the log probability with respect to a given θ_i can be carried out independently of the other maximizations.

Figure 9.1(b) provides a graphical depiction of this fact. The problem of finding the maximum likelihood parameters for the model in Figure 9.1(a) breaks into separate maximum likelihood problems, one for each node in the graph. As suggested in Figure 9.1(b), if we solve these separate maximum likelihood problems, we have solved the overall maximum likelihood problem.

9.2 Directed models

This chapter marks our first attempt to treat rather general families of graphical models. Although our results are rather straightforward, we will need to be a bit fussier with regards to notation than we have been in earlier chapters. This will allow us to state our results in a simple and general way, and will prepare the ground for later chapters.

Let \mathcal{G} be a directed graph, where \mathcal{V} is the set of nodes and \mathcal{E} the set of edges of the graph. We associate a random vector X with the graph, where the components of the vector are indexed by the nodes in the graph. Thus, X_u denotes the random variable associated with node $u \in \mathcal{V}$, and x_u denotes a realization of this random variable.

Recall that we also allow subsets of \mathcal{V} to serve as indices; thus, X_C refers to the set of components indexed by a subset $C \subseteq \mathcal{V}$. The vector X will itself sometimes be written $X_{\mathcal{V}}$.

We define a probability model for a directed graph via a set of local conditional probability distributions. Thus, to each node $u \in \mathcal{V}$ we associate a local conditional probability distribution $p(x_u | x_{\pi_u}, \theta_u)$, where π_u denotes the set of indices of the parents of u and where θ_u is a parameter vector. The overall probability associated with the graph \mathcal{G} is a product of these local conditional probabilities:

$$p(x_{\mathcal{V}} | \theta) = \prod_{u \in \mathcal{V}} p(x_u | x_{\pi_u}, \theta_u), \quad (9.3)$$

where $\theta = (\theta_1, \theta_2, \dots, \theta_m)$.

We treat the case of complete observations in this chapter. A *complete observation* is an assignment of values to all of the random variables $X_{\mathcal{V}}$ in the model.

In many problems the data are assumed to consist of a set of N independent, identically distributed (IID) observations. Such an assumption requires no special treatment within the graphical model formalism—we simply replicate a basic graphical structure N times. The result is itself a graphical model. It is important, however, to be able to continue to refer to the probability model associated with a single observable vector $X_{\mathcal{V}}$, apart from any considerations of the sampling mechanism, and also to be able to refer to the overall model that assigns probability to the IID replicates of $X_{\mathcal{V}}$. Let us continue to use the notation $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and $p(x_{\mathcal{V}} | \theta)$ to refer to the probability model associated with a single observable vector $X_{\mathcal{V}}$. We also construct an augmented graphical model, $\mathcal{G}^{(N)} = (\mathcal{V}^{(N)}, \mathcal{E}^{(N)})$, that incorporates the IID sampling assumption—this graph consists of N disconnected replicates of \mathcal{G} . The nodes $\mathcal{V}^{(N)}$ in this augmented graph are indexed using a pair of labels, (u, n) , where $u \in \mathcal{V}$ designates a node in the underlying graphical model \mathcal{G} , and where $n \in \{1, 2, \dots, N\}$ designates the replication number (see Figure 9.2).

We again allow subsets of indices to be used wherever single indexes are used. Thus, (C, n) denotes the n th replicate of the set C , for $C \subseteq \mathcal{V}$. In particular, (\mathcal{V}, n) denotes the n th replicate of

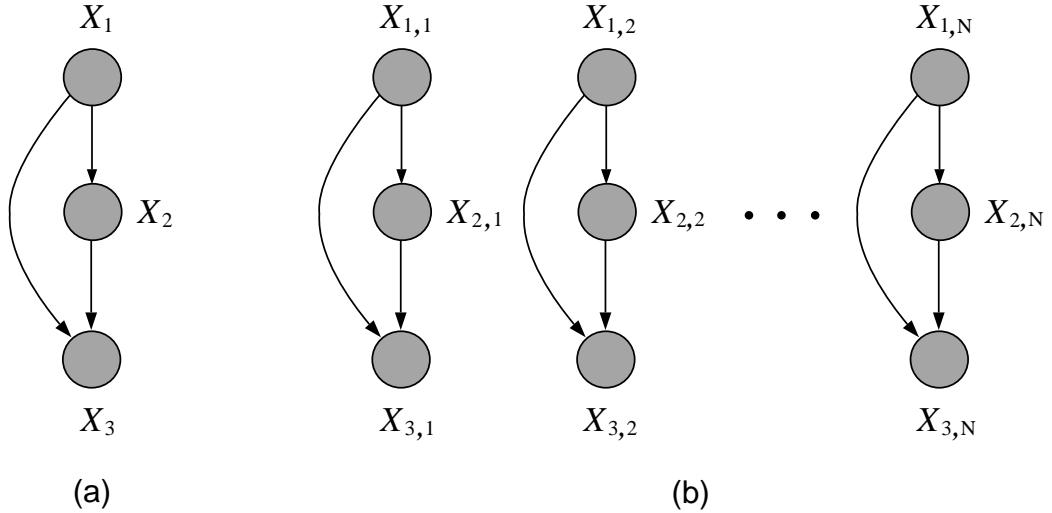


Figure 9.2: (a) An example of a graphical model \mathcal{G} , where $\mathcal{V} = \{1, 2, 3\}$. The set of random variables in the model is represented as $X_{\mathcal{V}} = (X_1, X_2, X_3)$, and $x_{\mathcal{V}} = (x_1, x_2, x_3)$ represents a realization of these variables. (b) An example of the graph $\mathcal{G}^{(N)}$, obtained by making N replicates of \mathcal{G} . Each random variable in this graph is indexed with two indices, (u, n) , the first denoting the underlying node in \mathcal{G} , and the second denoting the replication number. The n th complete observation is an assignment of values to all of the nodes in the n th replicate, and is denoted $x_{\mathcal{V},n} = (x_{1,n}, x_{2,n}, x_{3,n})$.

the entire set of observable nodes \mathcal{V} . This notation allows us to write:

$$\mathcal{D} = (x_{\mathcal{V},1}, x_{\mathcal{V},2}, \dots, x_{\mathcal{V},N}) \quad (9.4)$$

as our representation for the entire set of *observed data* in the completely observed setting.

The local conditional probability associated with a node (u, n) in $\mathcal{G}^{(N)}$ is defined to be the local conditional probability associated with the index u in the underlying graphical model \mathcal{G} . We thus obtain the following probability model for the graph $\mathcal{G}^{(N)}$:

$$p(\mathcal{D} \mid \theta) = \prod_n p(x_{\mathcal{V},n} \mid \theta) \quad (9.5)$$

$$= \prod_n \prod_u p(x_{u,n} \mid x_{\pi_u, n}, \theta_u), \quad (9.6)$$

where Eq. (9.5) is just the independence assumption, but it is also worth noting that it follows from the usual construction of the joint probability of a graph as a product over local conditional probabilities; here the graph is formed from a disconnected set of subgraphs indexed by n .

Taking the logarithm of Eq. (9.6) we obtain the log likelihood:

$$l(\theta; \mathcal{D}) = \sum_n \sum_u \log p(x_{u,n} | x_{\pi_u, n}, \theta_u); \quad (9.7)$$

Note that the log likelihood is a sum of a number of terms, each of which refers to only one of the parameter vectors θ_u . In estimating θ_u we can ignore all terms that involve $\theta_{u'}$, for $u' \neq u$. Note, moreover, that those terms involving θ_u refer only to x_u and x_{π_u} , which are observations of node u and its parents π_u . Thus, for the purposes of estimating the parameter vector θ_u , we need only focus on the data associated with the node u and its parents—we can ignore the data associated with the other nodes in the graph. In other words, the local subset of observations $\{x_{u,n}, x_{\pi_u,n}\}_{n=1}^N$ associated with these nodes is *sufficient* for θ_u .

We can often say more. In particular, if the observations $\{x_{u,n}, x_{\pi_u,n}\}_{n=1}^N$ can themselves be summarized by finite-dimensional sufficient statistics, as they can if each local conditional probability model, $p(x_u | x_{\pi_u}, \theta_u)$, is an exponential family distribution, then the entire estimation problem can be reduced to a collection of finite-dimensional sufficient statistics. In essence, we can reduce our problem from observations associated with the graph $\mathcal{G}^{(N)}$ to statistics associated with the graph \mathcal{G} , and from there to statistics associated with single nodes and their parents. In the next section we illustrate this reduction in the context of discrete graphical models.

9.2.1 Discrete models

Let us work out the sufficient statistics for a general graphical model in which all nodes are discrete. In doing so, we exemplify the general results that we have just discussed and also introduce a few “tricks of the trade” that will be useful in the following section as well as in later chapters.

Counts and marginal counts

We begin by introducing a notation for counts, which are the sufficient statistics for multinomial random variables. For a given configuration $x_{\mathcal{V}}$, let $m(x_{\mathcal{V}})$ denote the number of times that $x_{\mathcal{V}}$ is observed among the observations in the dataset \mathcal{D} . (There are only a finite number of possible configurations $x_{\mathcal{V}}$, and each data point $x_{\mathcal{V},n}$ must be one of these configurations). We can represent this count as a sum:

$$m(x_{\mathcal{V}}) \triangleq \sum_n \delta(x_{\mathcal{V}}, x_{\mathcal{V},n}), \quad (9.8)$$

where $\delta(x_{\mathcal{V}}, x_{\mathcal{V},n})$ is one if its arguments are equal and zero otherwise.

We can also define “marginal counts”—the counts associated with subsets of nodes. For any given subset C , let $m(x_C)$ denote the number of times that configuration x_C is observed in the data set. This is obtained by computing:

$$m(x_C) \triangleq \sum_{x_{\mathcal{V} \setminus C}} m(x_{\mathcal{V}}), \quad (9.9)$$

which can be viewed as a “marginalization” operation.

As an example, suppose that $\mathcal{V} = \{1, 2, 3\}$. In this case, $m(x_{\mathcal{V}})$ can be represented as a three-dimensional table. Suppose that the parent of X_2 is X_1 . To compute the marginal count $m(x_1, x_2)$, we sum over x_3 :

$$m(x_1, x_2) = \sum_{x_3} m(x_1, x_2, x_3), \quad (9.10)$$

which is a two-dimensional table. To compute the marginal count $m(x_1)$, a one-dimensional table, we sum over x_2 and x_3 :

$$m(x_1) = \sum_{x_2, x_3} m(x_1, x_2, x_3) = \sum_{x_2} m(x_1, x_2). \quad (9.11)$$

Note also that if we sum over all three variables we obtain the scalar N , the total number of observations.

A particular subset of interest is the subset consisting of a node u and its parents π_u —the *family* associated with node u . Letting $\phi_u \triangleq \{u\} \cup \pi_u$ denote this family, we have:

$$m(x_{\phi_u}) \triangleq \sum_{x_{\mathcal{V} \setminus \phi_u}} m(x_{\mathcal{V}}). \quad (9.12)$$

That is, $m(x_{\phi_u})$ is the count of the number of times a node u takes on a specific value and its parents π_u take on a specific configuration. Study the expression under the summation sign carefully. We are taking the sum over all nodes in \mathcal{V} other than u and its parents π_u .

The joint probability

Let us turn to the representation of the joint probability distribution in the discrete case. We begin by discussing the tabular case, in which a separate parameter is associated with each possible joint configuration of a node and its parents. In particular, we define the parameter vector $\theta_v(x_{\phi_v})$ to be a nonnegative, multidimensional table indexed by the joint configuration of v and π_v . The normalization condition requires:

$$\sum_{x_v} \theta_v(x_{\phi_v}) = \sum_{x_v} \theta_v(x_v, x_{\pi_v}) = 1, \quad (9.13)$$

where we recall that $\phi_v \triangleq \{v\} \cup \pi_v$ is the family associated with node v .

Given such a normalized table we define:

$$p(x_v | x_{\pi(v)}, \theta_v) \triangleq \theta_v(x_{\phi_v}) \quad (9.14)$$

as the local conditional probability of node v . This is a generic tabular representation that places no constraints on the local conditional probabilities beyond the normalization constraint.

Taking the product over v , we obtain the joint probability distribution:

$$p(x_{\mathcal{V}} | \theta) = \prod_v p(x_v | x_{\pi_v}, \theta_v) \quad (9.15)$$

$$= \prod_v \theta_v(x_{\phi_v}), \quad (9.16)$$

as a product of (normalized) potentials.

We take a further product over n to obtain the total probability of an IID data set $\mathcal{D} = (x_{\mathcal{V},1}, x_{\mathcal{V},2}, \dots, x_{\mathcal{V},N})$. At this point, however, we can make our results look neater if we recognize

that some of the observations have the same value. These observations necessarily have the same probability, and thus it is helpful to group them explicitly. To do this, we make use of a trick that should be familiar from our earlier work with the multinomial distribution:

$$p(x_{\mathcal{V},n} | \theta) = \prod_{x_{\mathcal{V}}} p(x_{\mathcal{V}} | \theta)^{\delta(x_{\mathcal{V}}, x_{\mathcal{V},n})}. \quad (9.17)$$

Here the dummy variable $x_{\mathcal{V}}$ ranges across configurations of the nodes rather than across data points. Using this representation we write the joint probability as follows, working in the log domain for convenience:

$$\log p(\mathcal{D} | \theta) = \log \left(\prod_n p(x_{\mathcal{V},n} | \theta) \right) \quad (9.18)$$

$$= \sum_n \log \left(\prod_{x_{\mathcal{V}}} p(x_{\mathcal{V}} | \theta)^{\delta(x_{\mathcal{V}}, x_{\mathcal{V},n})} \right) \quad (9.19)$$

$$= \sum_n \sum_{x_{\mathcal{V}}} \delta(x_{\mathcal{V}}, x_{\mathcal{V},n}) \log p(x_{\mathcal{V}} | \theta) \quad (9.20)$$

$$= \sum_{x_{\mathcal{V}}} m(x_{\mathcal{V}}) \log p(x_{\mathcal{V}} | \theta). \quad (9.21)$$

Note that the sum over n has disappeared; we have in essence reduced our representation of joint probability from a function on the graph $\mathcal{G}^{(N)}$ to a function on the graph \mathcal{G} . Continuing the derivation, we have:

$$\log p(\mathcal{D} | \theta) = \sum_{x_{\mathcal{V}}} m(x_{\mathcal{V}}) \log p(x_{\mathcal{V}} | \theta) \quad (9.22)$$

$$= \sum_{x_{\mathcal{V}}} m(x_{\mathcal{V}}) \log \left(\prod_v \theta_v(x_{\phi_v}) \right) \quad (9.23)$$

$$= \sum_{x_{\mathcal{V}}} m(x_{\mathcal{V}}) \sum_v \log \theta_v(x_{\phi_v}) \quad (9.24)$$

$$= \sum_v \sum_{x_{\phi_v}} \sum_{x_{\mathcal{V} \setminus \phi_v}} m(x_{\mathcal{V}}) \log \theta_v(x_{\phi_v}) \quad (9.25)$$

$$= \sum_v \sum_{x_{\phi_v}} \left(\sum_{x_{\mathcal{V} \setminus \phi_v}} m(x_{\mathcal{V}}) \right) \log \theta_v(x_{\phi_v}) \quad (9.26)$$

$$= \sum_v \sum_{x_{\phi_v}} m(x_{\phi_v}) \log \theta_v(x_{\phi_v}). \quad (9.27)$$

This result expresses the logarithm of the joint probability as a sum of terms defined on the families $\{\phi_v\}$.

Taking the exponential of both sides of Eq. (9.27), we see that we have an exponential family distribution for \mathcal{D} , with $m(x_{\phi_v})$ as the sufficient statistics and $\log \theta_v(x_{\phi_v})$ as the natural parameters.

Note also the decoupling discussed earlier. The parameters $\theta_v(x_{\phi_v})$ appear in separate terms which can be optimized independently of each other. In particular, to estimate $\theta_v(x_{\phi_v})$, we maximize $m(x_{\phi_v}) \log \theta_v(x_{\phi_v})$ with respect to $\theta_v(x_{\phi_v})$. Adding a Lagrangian term to handle the normalization constraint in Eq. (9.13), we obtain (see Exercise ??):

$$\hat{\theta}_{v,ML}(x_{\phi_v}) = \frac{m(x_{\phi_v})}{m(x_{\pi_v})} = \frac{m(x_v, x_{\pi_v})}{m(x_{\pi_v})}. \quad (9.28)$$

The result has an intuitively-appealing form—the maximum likelihood estimate is the ratio of the count of the number of times a node and its parents are jointly in a specific configuration to the count of the number of times its parents are in that configuration. These estimates are formed independently at each of the nodes in the graph.

Tabular representations are useful for graphs with small families, but quickly become unwieldy when the number of parents grows (the size of a table is exponential in the number of parents). For large families we generally wish to constrain $p(x_v | x_{\pi_v}, \theta_v)$ in some way. The generalized linear models (GLIMs) studied in Chapter 8 provide one example of such a constrained representation. Moreover, for GLIMs we have an efficient algorithm—the IRLS algorithm—for obtaining parameter estimates. The decoupling of the log likelihood implies that if each local conditional model is a GLIM model, then we solve the maximum likelihood problem for the graph as a whole by running the IRLS algorithm separately at each node.

In general, in the case of completely observed data, any solution to the problem of estimating parameters at a single node (conditional on its parents) applies immediately to general directed graphs.

9.3 Undirected models

Undirected models are in certain respects more flexible than their directed counterparts. Whereas in directed models the potentials are restricted to conditional distributions that model the dependence of a variable on its parents, potentials in undirected models are unnormalized functions defined on arbitrary subsets of nodes. In domains such as vision and information retrieval, which involve large collections of variables that do not necessarily have a natural ordering, undirected models are often the preferred choice.

Such flexibility comes at a cost, however. In particular, undirected graphical models require an explicit global normalization factor—the $1/Z$ factor in the definition of the joint probability. This global normalization factor couples the parameters and complicates the parameter estimation problem. There is, however, a special class of undirected models—the decomposable models—for which the parameter estimation problem decouples, despite the presence of the global normalization factor. For general undirected models the problem does not decouple, but, nonetheless, effective parameter estimation algorithms that exploit the structure of the graph are available. We describe some of the basic alternatives in this section, and treat the topic of parameter estimation for undirected models in more detail in Chapter 20.

Much of the notation for undirected models transfers over without change from the directed setting. Let $\mathcal{G}(\mathcal{V}, \mathcal{E})$ be an undirected graph, where \mathcal{V} is the set of nodes and \mathcal{E} the set of edges of

the graph. We again associate a random vector $X_{\mathcal{V}}$ with the graph and refer to random vectors associated with subsets of the nodes using the notation X_C , for $C \subseteq \mathcal{V}$. When the observed data are IID replicates, we again use a second index for the replicates. Thus $X_{C,n}$ denotes the n th replicate of the subset C , and the observed data are denoted:

$$\mathcal{D} = (x_{\mathcal{V},1}, x_{\mathcal{V},2}, \dots, x_{\mathcal{V},N}), \quad (9.29)$$

where the index \mathcal{V} reflects our focus on complete data.

We parameterize an undirected graphical model via a set of clique potentials $\psi_C(x_C)$, for $C \in \mathcal{C}$, where \mathcal{C} is a set of cliques. Note that we do not necessarily assume that \mathcal{C} contains all of the cliques in the graph, nor that the cliques in \mathcal{C} are maximal. Given such a set of clique potentials, we define the joint probability, $p(x_{\mathcal{V}} | \theta)$, as follows:

$$p(x_{\mathcal{V}} | \theta) = \frac{1}{Z} \prod_C \psi_C(x_C), \quad (9.30)$$

where $\theta = \{\psi_C(x_C), C \in \mathcal{C}\}$ is the collection of parameters, and where Z is the normalization factor:

$$Z = \sum_{x_{\mathcal{V}}} \prod_C \psi_C(x_C), \quad (9.31)$$

obtained by summing (or integrating in the continuous case) over all configurations $x_{\mathcal{V}}$.

9.3.1 Discrete models

In order to simplify our discussion we specialize to discrete random variables for the remainder of this section (see Chapter 20 for a discussion of continuous-valued undirected models).

Recall our notation for counts. The number of times that configuration $x_{\mathcal{V}}$ is observed in a dataset \mathcal{D} is represented as follows:

$$m(x_{\mathcal{V}}) \triangleq \sum_n \delta(x_{\mathcal{V}}, x_{\mathcal{V},n}), \quad (9.32)$$

and

$$m(x_C) \triangleq \sum_{x_{\mathcal{V} \setminus C}} m(x_{\mathcal{V}}) \quad (9.33)$$

is the marginal count for clique C . Note in particular that

$$N = \sum_{x_{\mathcal{V}}} m(x_{\mathcal{V}}) \quad (9.34)$$

is the total number of observations.

To express the log likelihood in terms of counts (the sufficient statistics for discrete models) we proceed as in the directed case, introducing the dummy variable $x_{\mathcal{V}}$:

$$p(x_{\mathcal{V},n} | \theta) = \prod_{x_{\mathcal{V}}} p(x_{\mathcal{V}} | \theta)^{\delta(x_{\mathcal{V}}, x_{\mathcal{V},n})}, \quad (9.35)$$

and writing the probability of the observed data as follows:

$$p(\mathcal{D} | \theta) = \prod_n p(x_{\mathcal{V},n} | \theta) \quad (9.36)$$

$$= \prod_n \prod_{x_{\mathcal{V}}} p(x_{\mathcal{V}} | \theta)^{\delta(x_{\mathcal{V}}, x_{\mathcal{V},n})}. \quad (9.37)$$

This trick allows us to write the log likelihood in terms of the marginal counts:

$$l(\theta; \mathcal{D}) = \log p(\mathcal{D} | \theta) \quad (9.38)$$

$$= \sum_n \sum_{x_{\mathcal{V}}} \delta(x_{\mathcal{V}}, x_{\mathcal{V},n}) \log p(x_{\mathcal{V}} | \theta) \quad (9.39)$$

$$= \sum_{x_{\mathcal{V}}} m(x_{\mathcal{V}}) \log p(x_{\mathcal{V}} | \theta) \quad (9.40)$$

$$= \sum_{x_{\mathcal{V}}} m(x_{\mathcal{V}}) \log \left(\frac{1}{Z} \prod_C \psi_C(x_C) \right) \quad (9.41)$$

$$= \sum_{x_{\mathcal{V}}} m(x_{\mathcal{V}}) \sum_C \log \psi_C(x_C) - \sum_{x_{\mathcal{V}}} m(x_{\mathcal{V}}) \log Z \quad (9.42)$$

$$= \sum_C \sum_{x_C} m(x_C) \log \psi_C(x_C) - N \log Z \quad (9.43)$$

We see that the marginal counts $m(x_C)$, for $C \in \mathcal{C}$, are the sufficient statistics for our model. This is reminiscent of the directed case, where the cliques \mathcal{C} were the families $\{\phi_v\}$. Note, however, an important difference between the undirected log likelihood and its directed counterpart—the appearance of the term $N \log Z$.

9.3.2 Maximum likelihood estimation

To find maximum likelihood estimates we take derivatives of the log likelihood and set to zero. Unfortunately, due to the $N \log Z$ term, such a calculation yields a coupled, nonlinear set of equations in which the parameters appear implicitly. It is not obvious how (in general) to solve these equations to obtain explicit estimates for the parameters. All is not lost, however; the implicit equations do reveal an interesting local property of the maximum likelihood estimates. Moreover, as we discuss in Section 9.3.4, this property provides the inspiration for an iterative algorithm for finding the parameter estimates.

Let us thus proceed to calculating the derivatives. In these calculations note that $\psi_C(x_C)$ is the independent variable, where both the clique C and the configuration x_C have been fixed (in the discrete case, this picks out the cell indexed by x_C in the table representing the potential function on clique C).

The derivative of the first term in the log likelihood, Eq. (9.43), with respect to $\psi_C(x_C)$ is obtained immediately; it is equal to $m(x_C)/\psi_C(x_C)$. We turn to $\log Z$:

$$\frac{\partial \log Z}{\partial \psi_C(x_C)} = \frac{1}{Z} \frac{\partial}{\partial \psi_C(x_C)} \left(\sum_{\tilde{x}} \prod_D \psi_D(\tilde{x}_D) \right) \quad (9.44)$$

$$= \frac{1}{Z} \sum_{\tilde{x}} \delta(\tilde{x}_C, x_C) \frac{\partial}{\partial \psi_C(x_C)} \left(\prod_D \psi_D(\tilde{x}_D) \right) \quad (9.45)$$

$$= \frac{1}{Z} \sum_{\tilde{x}} \delta(\tilde{x}_C, x_C) \prod_{D \neq C} \psi_D(\tilde{x}_D) \quad (9.46)$$

$$= \sum_{\tilde{x}} \delta(\tilde{x}_C, x_C) \frac{1}{\psi_C(\tilde{x}_C)} \frac{1}{Z} \prod_D \psi_D(\tilde{x}_D) \quad (9.47)$$

$$= \frac{1}{\psi_C(x_C)} \sum_{\tilde{x}} \delta(\tilde{x}_C, x_C) p(\tilde{x}) \quad (9.48)$$

$$= \frac{p(x_C)}{\psi_C(x_C)}, \quad (9.49)$$

where we have dropped the explicit reference to θ in our notation. With this result in hand, we obtain the derivative of the log likelihood:

$$\frac{\partial l}{\partial \psi_C(x_C)} = \frac{m(x_C)}{\psi_C(x_C)} - N \frac{p(x_C)}{\psi_C(x_C)}, \quad (9.50)$$

where we can assume without loss of generality that $\psi_C(x_C) > 0$ (see Exercise ??).

Setting Eq. (9.50) equal to zero, we obtain:

$$\hat{p}_{ML}(x_C) = \frac{1}{N} m(x_C). \quad (9.51)$$

Defining the *empirical distribution* $\tilde{p}(x) \triangleq m(x)/N$, so that $\tilde{p}(x_C) \triangleq m(x_C)/N$ is a marginal under the empirical distribution, we can rewrite the result as:

$$\hat{p}_{ML}(x_C) = \tilde{p}(x_C). \quad (9.52)$$

Thus we have the following important characterization of maximum likelihood estimates—for each clique $C \in \mathcal{C}$, the model marginals must be equal to the empirical marginals.

While this result constrains maximum likelihood models, it leaves us somewhat short of our goal. How do we find maximum likelihood estimates of the *parameters*? Eq. (9.52) provides us with a system of equations (by ranging over all $C \in \mathcal{C}$) that constrains the maximum likelihood estimates, but the parameters $\psi_C(x_C)$ themselves appear implicitly in these equations.

It turns out that for some graphs, in particular for a special class of graphs known as *decomposable graphs*, the situation is rather benign. Indeed, for these graphs, the maximum likelihood estimation problem decouples, and we can write down maximum likelihood estimates by inspection.

9.3.3 Decomposable models

Consider the example shown in Figure 9.3, a Markov chain on $\{X_1, X_2, X_3\}$. The probability model can be written as:

$$p(x_V) = \frac{1}{Z} \psi_{12}(x_1, x_2) \psi_{23}(x_2, x_3) \quad (9.53)$$

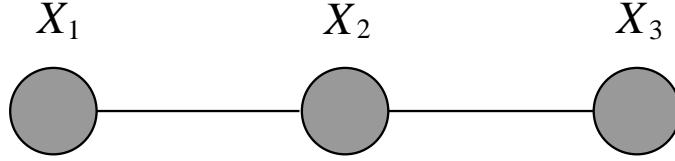


Figure 9.3: A Markov chain.

and the sufficient statistics under IID sampling are the empirical marginals $\tilde{p}(x_1, x_2)$ and $\tilde{p}(x_2, x_3)$.

For a model to be a maximum likelihood model, we require the marginals to equal the empirical marginals. How do we set the parameters so as to achieve this result?

Let us make the following guess:

$$\hat{p}_{ML}(x_1, x_2, x_3) = \frac{\tilde{p}(x_1, x_2)\tilde{p}(x_2, x_3)}{\tilde{p}(x_2)}, \quad (9.54)$$

where $\tilde{p}(x_2) = \sum_{x_1} \tilde{p}(x_1, x_2) = \sum_{x_3} \tilde{p}(x_2, x_3)$. That this is a good guess is readily verified:

$$\hat{p}_{ML}(x_1, x_2) = \sum_{x_3} \hat{p}_{ML}(x_1, x_2, x_3) = \tilde{p}(x_1, x_2) \quad (9.55)$$

$$\hat{p}_{ML}(x_2, x_3) = \sum_{x_1} \hat{p}_{ML}(x_1, x_2, x_3) = \tilde{p}(x_2, x_3), \quad (9.56)$$

and we see that the model marginals are indeed equal to the empirical marginals on the cliques \mathcal{C} .

Moreover, we can also easily match the terms in the guessed distribution to the parameters. For example, we can let:

$$\hat{\psi}_{12,ML}(x_1, x_2) = \tilde{p}(x_1, x_2) \quad (9.57)$$

and

$$\hat{\psi}_{23,ML}(x_2, x_3) = \frac{\tilde{p}(x_2, x_3)}{\tilde{p}(x_2)}, \quad (9.58)$$

which together imply that $Z = 1$. There are obviously many other sets of parameter estimates that yield the same joint distribution; these are all maximum likelihood estimates.

That this approach cannot work in general, however, is shown by the graph in Figure 9.4(a). As we invite the reader to verify in Exercise ??, the analogous ratio of the empirical marginals does not yield a maximum likelihood distribution in this case.

The problem is not simply due to an inability to handle graphs with loops. This can be seen by considering Figure 9.4(b). Here the reader can verify that the guess:

$$\hat{p}_{ML}(x_1, x_2, x_3, x_4) = \frac{\tilde{p}(x_1, x_2, x_3)\tilde{p}(x_2, x_3, x_4)}{\tilde{p}(x_2, x_3)} \quad (9.59)$$

fits the empirical marginals correctly.

There is a subtlety in this latter case, however. To recover parameter estimates from the factored form in Eq. (9.59) we need to have potentials that have the appropriate arguments. That

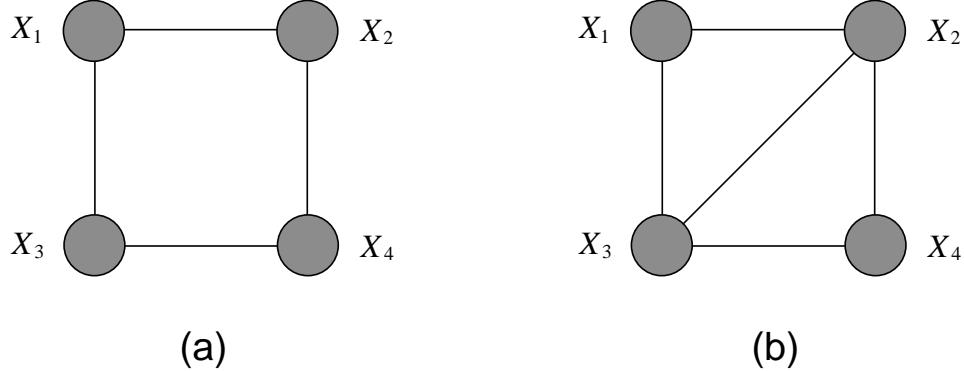


Figure 9.4: (a) A non-decomposable graph. (b) A decomposable graph.

is, in the case of Figure 9.4(b) we require that the model contain potentials $\psi_{123}(x_1, x_2, x_3)$ and $\psi_{234}(x_2, x_3, x_4)$. If the graph is parameterized with potentials that only refer to proper subsets of the maximal cliques, for example, pairwise potentials, then the guess in Eq. (9.59) is not in fact a maximum likelihood distribution because it is not in our model family.

In the remainder of this section, we briefly discuss the underlying concept that makes it possible to write maximum likelihood estimates by inspection in some cases but not in others. We will be brief because an understanding of this concept—that of a decomposable graph—requires additional machinery that we will not have in hand until Chapter 17. For completeness, however, let us define decomposability, and provide a recipe for constructing maximum likelihood estimates for decomposable graphs.

A graph is said to be *decomposable* if it can be recursively subdivided into disjoint sets A , B and S , where S separates A and B , and where S is complete.

The graph in Figure 9.3 is decomposable with $A = X_1$, $B = X_3$ and $S = X_2$. Similarly, the graph in Figure 9.4(b) is decomposable with $A = X_1$, $B = X_4$ and $S = \{X_2, X_3\}$. The reader can verify that the Figure 9.4(a) is not decomposable.

We can find maximum likelihood estimates for decomposable graphs by inspection, but only if the potentials are defined on maximal cliques. That is, our parameterization must be such that the set \mathcal{C} ranges over the maximal cliques in the graph. Given this constraint, the recipe is the following:

- for every clique C , set the clique potential to the empirical marginal for that clique,
- for every non-empty intersection between cliques, associate an empirical marginal with that intersection, and divide that empirical marginal into the potential of one of the two cliques that form the intersection.

For example, for the graph in Figure 9.3, this recipe leads immediately to the estimates given in Eq. (9.57) and Eq. (9.58).

We have barely scratched the surface of the decomposability concept, but we hope to have piqued the reader's interest. To get a further hint of what is to come (in Chapter 17), note that

there is another way to describe the differences between the graphs in Figures 9.3, 9.4(a), and 9.4(b): Only for Figure 9.4(a) is it impossible to eliminate the nodes in the graph without adding extra edges to the graph. This suggests a relationship between decomposability and graph elimination, and suggests (rightly) that decomposability lies with elimination at the core of the relationship between graphs and probabilities.

We now turn to an alternative algorithm for finding maximum likelihood estimates in all undirected graphs, whether decomposable and nondecomposable. In the decomposable case the algorithm turns out to converge in a finite number of iterations, updating each potential once. Indeed, in this case, the algorithm essentially implements the recipe for decomposable graphs that we described above. Thus the algorithm can be viewed as a general solution to the maximum likelihood estimation problem that takes advantage of the decomposable structure in the problem if it is present.

9.3.4 Iterative proportional fitting

A common strategy for solving systems of implicit equations is to “iterate,” hoping that the iterations converge to a “fixed point”—a set of values that solve the original implicit equations. *Iterative proportional fitting (IPF)*, an algorithm for maximum likelihood estimation in undirected models, can be viewed as an example of such a strategy. However, it is also possible to say something stronger about IPF. In general, the iteration of fixed-point equations does not necessarily yield a convergent algorithm; moreover, even if the iteration converges, it is not necessarily the case that the algorithm behaves well in the sense of ascending an objective function at each step. IPF, however, *does* converge, and *does* behave well—the log likelihood is guaranteed to increase or remain the same after each IPF update. The facts about IPF follow from the fact that it is not only a fixed-point algorithm, but that it is also a *coordinate ascent algorithm*.

We begin with a simple, heuristic justification of IPF in terms of fixed-point iteration, and present the more satisfying justification in terms of coordinate ascent later in this section.

To develop an iterative approach to maximum likelihood estimation for undirected models, let us return to the gradient of the log likelihood, but retain the $\psi_C(x_C)$ factors. From Eq. (9.50) we have:

$$\frac{\tilde{p}(x_C)}{\psi_C(x_C)} = \frac{p(x_C)}{\psi_C(x_C)}, \quad (9.60)$$

where $\tilde{p}(x_C) \triangleq m(x_C)/N$ is the empirical marginal. Note that the parameter $\psi_C(x_C)$ appears explicitly in this equation in two places, but also appears implicitly in the marginal $p(x_C)$. We can obtain an iterative algorithm by holding the values of $\psi_C(x_C)$ fixed on the right-hand side of the equation, both in the numerator and the denominator, and solving for the free parameter $\psi_C(x_C)$ on the left-hand side.

In particular, let us denote the set of parameter estimates at iteration t by $\psi_C^{(t)}(x_C)$. Let $p^{(t)}(x)$ denote the joint probability distribution based on these parameter estimates. Rearranging Eq. (9.60), we define the following update for $\psi_C(x_C)$:

$$\psi_C^{(t+1)}(x_C) = \psi_C^{(t)}(x_C) \frac{\tilde{p}(x_C)}{p^{(t)}(x_C)}, \quad (9.61)$$

which is the IPF algorithm. The IPF algorithm cycles through all of the cliques $C \in \mathcal{C}$, applying Eq. (9.61) to each clique in turn; such a cycle constitutes a single iteration of the algorithm.

Before providing an interpretation of IPF in terms of coordinate ascent, let us explore some of the properties of the algorithm.

Properties of the IPF update equation

The IPF update equation in Eq. (9.61) has two interesting properties: (1) the marginal $p^{(t+1)}(x_C)$ is equal to the empirical marginal $\tilde{p}(x_C)$, and (2) the normalization factor Z remains constant across IPF updates.

To establish these properties, let us calculate the marginal probability of x_C for the updated distribution:

$$p^{(t+1)}(x_C) = \sum_{x_{\mathcal{V} \setminus C}} p^{(t+1)}(x) \quad (9.62)$$

$$= \sum_{x_{\mathcal{V} \setminus C}} \frac{1}{Z^{(t+1)}} \prod_D \psi_D^{(t+1)}(x_D) \quad (9.63)$$

$$= \frac{1}{Z^{(t+1)}} \sum_{x_{\mathcal{V} \setminus C}} \psi_C^{(t+1)}(x_C) \prod_{D \neq C} \psi_D^{(t)}(x_D) \quad (9.64)$$

$$= \frac{1}{Z^{(t+1)}} \sum_{x_{\mathcal{V} \setminus C}} \psi_C^{(t)}(x_C) \frac{\tilde{p}(x_C)}{p^{(t)}(x_C)} \prod_{D \neq C} \psi_D^{(t)}(x_D) \quad (9.65)$$

$$= \frac{Z^{(t)}}{Z^{(t+1)}} \frac{\tilde{p}(x_C)}{p^{(t)}(x_C)} \sum_{x_{\mathcal{V} \setminus C}} \frac{1}{Z^{(t)}} \prod_D \psi_D^{(t)}(x_D) \quad (9.66)$$

$$= \frac{Z^{(t)}}{Z^{(t+1)}} \frac{\tilde{p}(x_C)}{p^{(t)}(x_C)} \sum_{x_{\mathcal{V} \setminus C}} p^{(t)}(x) \quad (9.67)$$

$$= \frac{Z^{(t)}}{Z^{(t+1)}} \frac{\tilde{p}(x_C)}{p^{(t)}(x_C)} p^{(t)}(x_C) \quad (9.68)$$

$$= \frac{Z^{(t)}}{Z^{(t+1)}} \tilde{p}(x_C). \quad (9.69)$$

Note that $\tilde{p}(x_C)$ is a proportion and is therefore normalized, and $p^{(t+1)}(x_C)$ is normalized because we have divided explicitly by $Z^{(t+1)}$. Thus, summing both sides of our result with respect to x_C , we have:

$$Z^{(t+1)} = Z^{(t)}. \quad (9.70)$$

We see that the normalization factor Z indeed remains constant across IPF updates. Moreover, now that we know that the factor $Z^{(t)}/Z^{(t+1)}$ is equal to one, we see that we have also derived property (1):

$$p^{(t+1)}(x_C) = \tilde{p}(x_C). \quad (9.71)$$

That is, IPF finds a distribution such that the marginal with respect to the variables x_C is equal to the empirical marginal $\tilde{p}(x_C)$.

This interpretation of IPF accords well with the characterization of the maximum likelihood estimates that we gave in Section 9.3.2. We saw that for the maximum likelihood model, the model marginals are equal to the empirical marginals, for all $C \in \mathcal{C}$. IPF works toward the goal of equal model and empirical marginals, by equating a single model marginal and empirical marginal at a time.

The fact that the normalization factor Z stays constant across IPF iterations has another interesting consequence. As we ask the reader to verify (Exercise ??), the constancy of Z implies that we can write IPF in terms of joint probabilities:

$$p^{(t+1)}(x_{\mathcal{V}}) = p^{(t)}(x_{\mathcal{V}}) \frac{\tilde{p}(x_C)}{p^{(t)}(x_C)}, \quad (9.72)$$

and indeed this is how IPF is often defined.¹ From this equation, we immediately obtain:

$$p^{(t+1)}(x_{\mathcal{V}}) = p^{(t)}(x_{\mathcal{V} \setminus C} | x_C) \tilde{p}(x_C), \quad (9.73)$$

which provides an interpretation of an IPF iteration as retaining the “old” conditional probability $p^{(t)}(x_{\mathcal{V} \setminus C} | x_C)$, while replacing the “old” marginal probability $p^{(t)}(x_C)$ with the “new” marginal $\tilde{p}(x_C)$.

IPF as coordinate ascent

Let us now derive IPF as a coordinate ascent algorithm. A “coordinate” in this setting is a potential function.

To derive a coordinate ascent algorithm, we take the derivative of the log likelihood with respect to the “coordinate” $\psi_C(x_C)$, for fixed C and varying x_C , and solve for the maximizing values of these parameters while holding the remaining potentials fixed. To carry out this calculation, let us return to the calculation of the gradient in Section 9.3.2, pausing the derivation in midstream. From Eq. (9.46) we have:

$$\frac{\partial l}{\partial \psi_C(x_C)} = \frac{m(x_C)}{\psi_C(x_C)} - \frac{N}{Z} \sum_{\tilde{x}} \delta(\tilde{x}_C, x_C) \prod_{D \neq C} \psi_D(\tilde{x}_D). \quad (9.74)$$

Let us take some care here—the parameter $\psi_C(x_C)$ on the right-hand side of this equation is a *variable* whose maximizing value we wish to solve for, where the remaining parameters $\psi_D(\tilde{x}_D)$ are being held fixed. Adding superscripts to the latter to reflect the fact that they are being held fixed at iteration t of our algorithm, the gradient of the log likelihood becomes:

$$\frac{\partial l}{\partial \psi_C(x_C)} = \frac{m(x_C)}{\psi_C(x_C)} - \frac{N}{Z} \sum_{\tilde{x}} \delta(\tilde{x}_C, x_C) \prod_{D \neq C} \psi_D^{(t)}(\tilde{x}_D). \quad (9.75)$$

¹Treating Eq. (9.72) as a definition of the IPF algorithm leaves open the actual implementation of an IPF step in terms of the structural components of the model, and we view Eq. (9.61) as the better definition.

If we were to multiply and divide the second term by the “old” parameter value $\psi_C^{(t)}(\tilde{x}_C)$, the sum would appear to reduce to the marginal $p^{(t)}(x_C)$ much as in our earlier derivation. Setting to zero would yield the IPF update. However, we must not forget that Z also depends on $\psi_C(x_C)$. As $\psi_C(x_C)$ varies, Z varies, and in general the varying value of Z is *not* the correct normalization for the *old* values of the parameters.

We have seen, however, that for a particular value of $\psi_C(x_C)$, namely $\psi_C^{(t+1)}(x_C)$ as defined by the IPF update, we have $Z^{(t+1)} = Z^{(t)}$, which *is* the correct normalization for the old values of the parameters. Taking advantage of this fact, we obtain the following value of the derivative of log likelihood, where we now evaluate the derivative at $\psi_C^{(t+1)}(x_C)$:

$$\frac{\partial l}{\partial \psi_C(x_C)} = \frac{m(x_C)}{\psi_C^{(t+1)}(x_C)} - \frac{N}{Z^{(t+1)}} \sum_{\tilde{x}} \delta(\tilde{x}_C, x_C) \prod_{D \neq C} \psi_D^{(t)}(\tilde{x}_D) \quad (9.76)$$

$$= \frac{m(x_C)}{\psi_C^{(t+1)}(x_C)} - \frac{N}{Z^{(t)}} \sum_{\tilde{x}} \delta(\tilde{x}_C, x_C) \prod_{D \neq C} \psi_D^{(t)}(\tilde{x}_D) \quad (9.77)$$

$$= \frac{m(x_C)}{\psi_C^{(t+1)}(x_C)} - \frac{N}{\psi^{(t)}(x_C)} \sum_{\tilde{x}} \delta(\tilde{x}_C, x_C) \frac{1}{Z^{(t)}} \prod_D \psi_D^{(t)}(\tilde{x}_D) \quad (9.78)$$

$$= \frac{m(x_C)}{\psi_C^{(t+1)}(x_C)} - \frac{N}{\psi^{(t)}(x_C)} p^{(t)}(x_C), \quad (9.79)$$

and we see that the IPF update equation:

$$\psi_C^{(t+1)}(x_C) = \psi^{(t)}(x_C) \frac{\tilde{p}(x_C)}{p^{(t)}(x_C)} \quad (9.80)$$

does indeed set the gradient of the log likelihood to zero, and thus is a coordinate ascent step.

View from the KL divergence

We have shown that the IPF algorithm is coordinate ascent in the log likelihood. A somewhat more elegant derivation of this result can be obtained via the KL divergence.

The KL divergence has a useful decomposition that reflects the decomposition of a joint distribution into the product of a marginal and a conditional. In particular, writing $p(x_A, x_B) = p(x_B | x_A)p(x_A)$, and $q(x_A, x_B) = q(x_B | x_A)q(x_A)$, we have:

$$D(p(x_A, x_B) \| q(x_A, x_B)) = D(p(x_A) \| q(x_A)) + \sum_{x_A} p(x_A) D(p(x_B | x_A) \| q(x_B | x_A)). \quad (9.81)$$

(This result is derived in Appendix XXX).

Recall that the problem of maximizing the likelihood is equivalent to that of minimizing the following KL divergence:

$$D(\tilde{p}(x) \| p(x | \theta)) = \sum_x \tilde{p}(x) \log \frac{\tilde{p}(x)}{p(x | \theta)}, \quad (9.82)$$

where $\tilde{p}(x)$ is the empirical distribution. (The reason that these problems are equivalent is because the (negative) log likelihood and the KL divergence differ by the term $\sum_x \tilde{p}(x) \log \tilde{p}(x)$, which is independent of θ).

Thus we wish to perform coordinate descent in $D(\tilde{p}(x) \parallel p(x | \theta))$. That is, we pick a clique C and adjust the clique potential $\psi_C(x_C)$ so as to minimize the KL divergence. Applying Eq. (9.81), we have:

$$D(\tilde{p}(x) \parallel p(x | \theta)) = D(\tilde{p}(x_C) \parallel p(x_C | \theta)) + \sum_{x_C} \tilde{p}(x_C) D(\tilde{p}(x_{\mathcal{V} \setminus C} | x_C) \parallel p(x_{\mathcal{V} \setminus C} | x_C, \theta)). \quad (9.83)$$

Changes to the clique potential $\psi_C(x_C)$ have no effect on the conditional distribution $p(x_{\mathcal{V} \setminus C} | x_C, \theta)$.² Thus, the second term in Eq. (9.83) is unaltered by changes to $\psi_C(x_C)$, and minimizing the KL divergence reduces to minimizing the first term. This term is minimized by setting the marginal $p(x_C | \theta)$ equal to the empirical marginal $\tilde{p}(x_C)$. But this is exactly what an IPF update achieves. Thus IPF is coordinate descent in $D(\tilde{p}(x) \parallel p(x | \theta))$ or, equivalently, coordinate ascent in the log likelihood.

9.3.5 Gradient ascent

An alternative to IPF is to perform gradient ascent on the log likelihood. In this case, given that the gradient is evaluated only at the current value of the parameters, no subtleties arise.

Evaluating the gradient at $\psi_C^{(t)}(x_C)$, we have:

$$\frac{\partial l}{\partial \psi_C(x_C)} = \frac{m(x_C)}{\psi_C^{(t)}(x_C)} - \frac{N}{\psi^{(t)}(x_C)} p^{(t)}(x_C), \quad (9.84)$$

which leads to the following gradient ascent algorithm:

$$\psi_C^{(t+1)}(x_C) = \psi_C^{(t)}(x_C) + \frac{\rho}{\psi_C^{(t)}(x_C)} \left(\tilde{p}(x_C) - p^{(t)}(x_C) \right), \quad (9.85)$$

where ρ is a step size. We see that the difference between the empirical marginals and the model marginals drives the algorithm.

Gradient ascent has the advantage compared to IPF that all of the parameters can be adjusted simultaneously, although we will present a variant of IPF in Chapter 20 that also allows parameters to be adjusted simultaneously, so this advantage is somewhat diminished. Disadvantages of the gradient ascent approach include the need to choose a step size, and, even more seriously, the fact that the normalization factor Z does not remain constant, but must be recalculated anew after each iteration.

²This is proved by simply writing out the conditional distribution and noting that $\psi_C(x_C)$ cancels in the numerator and denominator; see Exercise ?? for details.

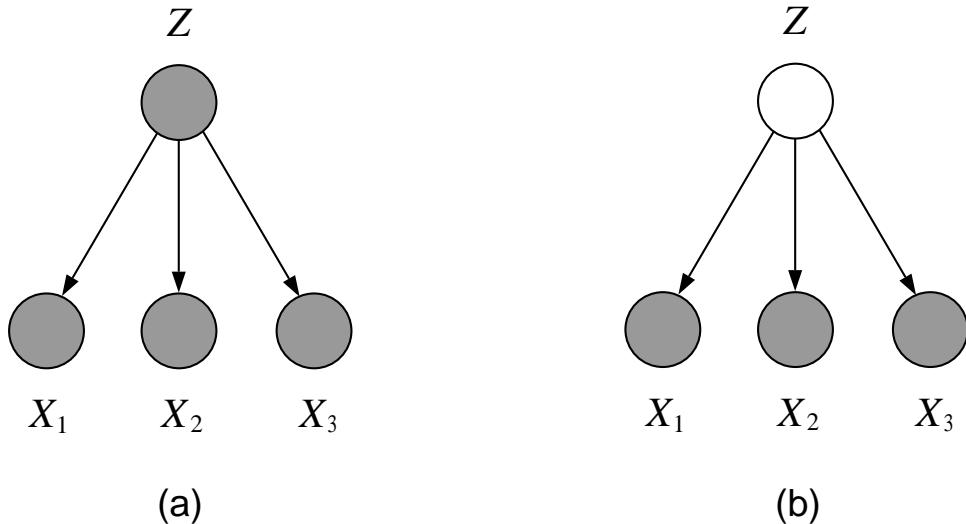


Figure 9.5: (a) The nodes X_i are conditionally independent given Z . (b) When Z is latent the variables X_i are no longer independent and this implies that the parameters are coupled in the log likelihood.

9.4 Latent variables

The algorithms that we have presented in this chapter all rely on the assumption of complete data. In the setting of complete data the likelihood is a product and the log likelihood is therefore a sum, where each of the parameters appear in different terms. This leads to a decoupling of the parameter estimation problem.

When some of the variables are unobserved, that is, when we have *latent variables*, this happy situation breaks down. When there are latent variables, the likelihood is a marginal probability, obtained by summing or integrating over the latent variables. The log likelihood is the logarithm of this sum, and the logarithm is prevented from moving past the sum to act on the product of potentials. The parameter estimation problem does not decouple.

Consider the graphical model shown in Figure 9.5(a). Here the nodes X_i are conditionally independent given Z ; thus, when Z is observed the log likelihood decouples:

$$l(\theta; x, z) = \log p(x, z | \theta) \quad (9.86)$$

$$= \log [p(z | \theta_z) p(x_1 | z, \theta_1) p(x_2 | z, \theta_2) p(x_3 | z, \theta_3)] \quad (9.87)$$

$$= \log p(z | \theta_z) + \log p(x_1 | z, \theta_1) + \log p(x_2 | z, \theta_2) + \log p(x_3 | z, \theta_3). \quad (9.88)$$

If, on the other hand, Z is latent, as shown in Figure 9.5(b), the log likelihood does not decouple:

$$l(\theta; x) = \log \sum_z p(x, z | \theta) \quad (9.89)$$

$$= \log \sum_z [p(z | \theta_z) p(x_1 | z, \theta_1) p(x_2 | z, \theta_2) p(x_3 | z, \theta_3)]. \quad (9.90)$$

Adjusting one parameter, say θ_1 , has an effect on all of the other parameters. This makes good sense probabilistically—our uncertainty about Z should be reflected in a probabilistic dependence among the variables X_i and hence among the estimates of the parameters θ_i . This coupling is unfortunate from a computational point of view, however, in that it complicates the task of parameter estimation.

In Chapter 11 we present a general procedure for dealing with latent variable problems known as the *Expectation-Maximization (EM)* algorithm. The EM algorithm in essence allows us to treat latent variable problems with complete data tools. In particular, the EM algorithm allows us to solve the maximum likelihood problem for the graph in Figure 9.5(b) by solving a sequence of maximum likelihood problems based on the graph in Figure 9.5(a).

To give the briefest hint of what is involved in the EM algorithm, essentially EM makes use of convexity to move the logarithm past the sum in coupled log likelihoods such as Eq. (9.90). This decouples the problem and allows the complete data tools of the current chapter to be brought into play. Exactly how this is done in general is the story for Chapter 11. In the meantime, in the following chapter we work through a specific example of a latent variable problem, developing a special case of the EM algorithm that is particularly simple and intuitive.

9.5 Summary

In the current chapter we have discussed parameter estimation in completely observed graphical models, treating both directed graphs and undirected graphs.

For directed graphs, we saw that the log likelihood decouples into separate terms, one for each parameter, and that the problem of parameter estimation therefore also decouples. Essentially, one collects the sufficient statistics associated with each node and its parents and estimates the parameter vector at that node using those sufficient statistics. This is done independently at each node in the graph. In particular, if the probability model at each node is a generalized linear model, then we can run the IRLS algorithm independently at each node. This is a Newton algorithm on the graph as a whole.

For undirected graphs, we distinguish between decomposable models and nondecomposable models. Decomposable models are the easy case; for decomposable models we can solve for maximum likelihood estimates analytically. We have only briefly outlined the reasons for this in the current chapter; in Chapter 17, we provide a deeper discussion of decomposability, revealing the important relationship between decomposability and the general approach to inference known as the junction tree algorithm. We then return to the problem of parameter estimation in Chapter 21 and nail down the results that we have only sketched here.

For nondecomposable models, we discussed the iterative proportional fitting (IPF) approach to parameter estimation. IPF takes the form of a simple scaling algorithm in which the potentials are multiplied by a ratio of marginal probabilities. We showed that IPF can be viewed as a fixed point algorithm and also as a coordinate ascent algorithm. These themes—iterative algorithms, multiplicative updates of potentials, and coordinate ascent—will appear in several guises in the following chapters, and we will see several IPF-like algorithms as we proceed.

9.6 Historical remarks and bibliography

Chapter 10

Mixtures and conditional mixtures

In this chapter we begin the study of models with *latent* or *hidden* variables. Latent variables are simply random variables whose values are not specified in the observed data—in the graphical model formalism these variables are the unshaded nodes. Our focus in the current chapter is the simple case of latent variables that can take one of a finite set of values.

Let us take a moment to pose the question of why would one include a node in a model if the value of that node cannot be observed in the data. Shouldn’t we include variables in our model only if their values can be observed? One answer to this question is philosophical—surely much human knowledge involves explaining observed data in terms of unobserved concepts.¹ For example, we often introduce *distinctions* into our reasoning in order to simplify relationships between observables. Thus a doctor may group patients into those with a certain “syndrome” and those without, and this grouping may make it easier to understand the relationships between observed symptoms. A biologist may wish to group animals into distinct species, because it may be easier to explain behavioral or physiological patterns within each species than to explain such patterns without the help of the distinction. Although such distinctions may exist only in the mind of the doctor or biologist, at least at the outset, their utility for modeling the data may provide the motivation for further study in which one tries to uncover a “real” physical or biological interpretation of the distinction.

Viewing a “distinction” as a discrete random variable ranging over a finite, unordered set of values leads to the mixture models studied in the current chapter.

In Chapter ?? we study continuous latent variable models in which the latent variable parameterizes a k -dimensional subspace of the d -dimensional input space; here the latent variable achieves a “dimensionality reduction.” Chapter 12 and Chapter 15 discuss models in which latent variables are used in the time series setting to summarize past data; that is, the latent variables are “state variables.” In all of these cases, and in others that we will meet, the general idea is the same—models with latent variables can often be simpler than models without latent variables. In statistical terms we often find that we can get by with fewer parameters using a latent variable model, or we can avail ourselves of simple parametric distributions that have advantageous com-

¹One should not, however, expect the philosophers to have agreed on this. Indeed, the philosophical school of *logical positivism* explicitly denied the meaningfulness of using unobservable concepts in scientific reasoning.

putational or analytical properties. We will not attempt to define “simplicity” more rigorously for now—that is the task of Chapter 26. Instead we proceed by example, describing latent variable models that have been shown to be useful in practice.

In this chapter we discuss two kinds of models based on discrete latent variables—*unconditional mixture models* and *conditional mixture models*. Roughly speaking, unconditional mixture models are used to solve density estimation problems, whereas conditional mixture models are used to solve regression and classification problems. One useful perspective to take on the latent variable methodology in both of these kinds of problems is that it allows us to break problems into subproblems. Thus, in unconditional mixture modeling, for each value of the latent variable we obtain a (presumably simpler) density estimation subproblem. In conditional mixture modeling, for each value of the latent variable we obtain a (presumably simpler) regression or classification subproblem. In general, mixture modeling can be viewed as a “divide-and-conquer” approach to statistical modeling.

10.1 Unconditional mixture models

We begin by discussing unconditional mixture models. While regression and classification models require the observation of (X, Y) pairs, unconditional mixture models make do with observations of X alone.

As we discussed in Chapter 5, mixture models can be used to solve density estimation problems, allowing us to answer questions about whether query vectors are “typical” or “untypical.” This has many applications, including the detection of outliers and the design of algorithms for data compression. Note that in such applications we are not necessarily interested in identifying or interpreting the structure of the probability distribution generating the data; rather we are simply interested in a flexible model that allows us to obtain a good estimate of the probability density.

In other problems, however, we may have a more “structural” interest in the mixture model. In particular, as we discussed in Chapter 5, we may wish to take the point of view that there are “subpopulations” underlying the data. In this setting, mixture modeling is closely linked to classification, in particular to the generative classification models discussed in Chapter 7. Indeed, treating the class label of a generative classification model as a latent variable converts the model into a mixture model. Reserving the term *classification* for the setting in which the labels are in fact observed, we use the term *clustering* for the problem of inferring the labels of data points when such labels are absent in the data. Mixture models provide a popular and widely used methodology for clustering.

In Chapter 5 we presented the following general formulation of an unconditional mixture model (see Figure 10.1). Let Z represent a multinomial random variable with components Z^i . We have:

$$p(x | \theta) = \sum_i p(Z^i = 1 | \pi_i) p(x | Z^i = 1, \theta_i) \quad (10.1)$$

$$= \sum_i \pi_i p(x | Z^i = 1, \theta_i). \quad (10.2)$$

where $\theta = (\pi_1, \dots, \pi_K, \theta_1, \dots, \theta_K)$ and where the π_i are constrained to sum to one. Recall the



Figure 10.1: A mixture model represented as a graphical model. The latent variable Z is a multinomial node taking on one of K values.

terminology—the parameters π_i are referred to as *mixing proportions* and the densities $p(x | Z^i = 1, \theta_i)$ are referred to as *mixture components*.

10.1.1 Gaussian mixture models

Let us begin by discussing the important special case of the Gaussian mixture model. In this model the mixture components are Gaussian distributions with parameters $\theta_i \triangleq (\mu_i, \Sigma_i)$. Note that we allow the covariance to vary across the mixture components. One can also consider models in which the covariance matrices are constrained to be equal.

From Eq. (10.2) we obtain the following probability model for a Gaussian mixture:

$$p(x | \theta) = \sum_i \pi_i \frac{1}{(2\pi)^{m/2} |\Sigma_i|^{1/2}} \exp \left\{ -\frac{1}{2} (x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i) \right\}. \quad (10.3)$$

We will also write this as:

$$p(x | \theta) = \sum_i \pi_i \mathcal{N}(x | \mu_i, \Sigma_i) \quad (10.4)$$

to simplify notation.

Figure 10.2 shows a simple illustration of a Gaussian mixture model, together with a sample from the marginal distribution.

Let us calculate the probability of the latent variable Z conditioned on the observed variable X . This calculation is of obvious interest if we wish to use the mixture model in the clustering setting—the conditional probability of Z can be used to assign X to one of the clusters. We will also find that this conditional probability plays an important role in parameter estimation.

We let τ^i denote the conditional probability that the i th component of Z is equal to one. From Bayes rule we have:

$$\tau^i \triangleq p(Z^i = 1 | x, \theta) \quad (10.5)$$

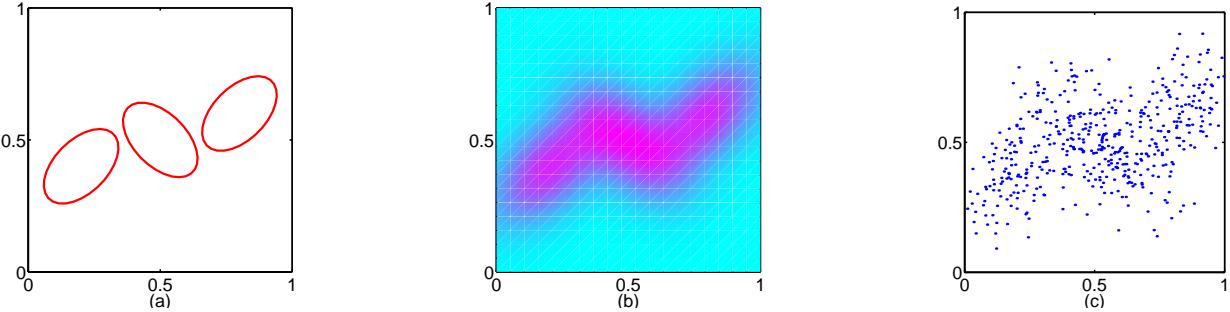


Figure 10.2: Illustration of a mixture of 3 Gaussians in a two-dimensional space showing (a) contours representing one standard deviation for each of the mixture components, (b) the marginal probability density of the mixture distribution, and (c) a sample of 500 points drawn from the marginal distribution.

$$= \frac{p(x | Z^i = 1, \theta_i)p(Z^i = 1 | \pi_i)}{p(x | \theta)} \quad (10.6)$$

$$= \frac{\pi_i \mathcal{N}(x, \mu_i | \Sigma_i)}{\sum_j \pi_j \mathcal{N}(x, \mu_j | \Sigma_j)} \quad (10.7)$$

Note the relationship to the generative classification models of Section 7.2. In particular, if we let the Σ_k be equal, then the quadratic terms cancel and we obtain the linear-softmax function as in that section.

It is common to refer to π_i as a “prior probability” and τ^i as a “posterior probability.” This is a convenient terminology that reflects the fact that these probabilities are linked via Bayes rule. Please note, however, that the use of this terminology is unrelated to whether or not we use Bayesian methods to estimate the parameters θ . Indeed, in this chapter our focus will be maximum likelihood estimation.

Let us now consider the problem of estimating θ from an IID set of observations $\mathcal{D} = \{x_n : n = 1, \dots, N\}$. The model is shown in Figure 10.3, where we see that each data point x_n is accompanied by a multinomial latent variable Z_n that represents the “assignment” of x_n to one of the mixture components. We form the log likelihood:

$$l(\theta | \mathcal{D}) = \sum_n \log p(x_n | \theta) \quad (10.8)$$

$$= \sum_n \log \sum_i \pi_i \mathcal{N}(x_n | \mu_i, \Sigma_i) \quad (10.9)$$

by taking the log of the product of N copies of the probability model in Eq. (10.3). Note the disconcerting fact that the logarithm stops in front of the sum. In all of the models that we have considered up until now, the logarithm acted directly on the basic probability distributions in our model, which, given the exponential family distributions that we have worked with, yielded simple expressions such as squared error or cross entropy. Here the likelihood is a marginal probability.

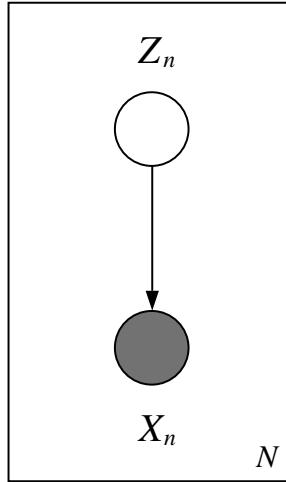


Figure 10.3: The mixture model under an IID sampling assumption.

This prevents the logarithm from acting directly on the component probability distributions and leaves us with a tangled nonlinear function to maximize.

One approach to maximizing this likelihood is to hand Eq. (10.9) to a nonlinear optimization algorithm such as conjugate gradient or Newton-Raphson. In Appendix XXX we provide some details regarding this approach. Our main focus, however, will be on an alternative approach to maximizing the likelihood known as the Expectation-Maximization (EM) algorithm. This algorithm is applicable far beyond the Gaussian mixture setting; indeed, it is applicable to arbitrary graphical models with latent variables. Its important virtue in the graphical model setting is that it allows us to take full advantage of the graphical structure underlying the likelihood; in particular, we will be able to exploit the inference algorithms discussed in Chapter 3 and Chapter 17. By relating the problem of parameter estimation and the problem of efficient inference, the EM algorithm brings together two of our major themes. It will play an important role throughout the book.

In this chapter we provide a heuristic introduction to the EM algorithm for Gaussian mixture models. Chapter 11 provides a rigorous derivation of EM, not only for Gaussian mixture models, but for the general case.

10.1.2 The K-means algorithm

To motivate the EM algorithm for Gaussian mixtures, it is useful to step briefly outside of the Gaussian mixture framework to consider an even simpler approach to clustering.

Recall that we have a set of observations $\mathcal{D} = \{x_n : n = 1, \dots, N\}$. Our goal is to group the data points into a set of K clusters, where we suppose that the value of K is given.

The *K-means algorithm* represents each cluster with a single vector, which we refer to as a “cluster mean.” The basic idea is to assign data points to clusters by finding the nearest cluster mean and assigning the data point to that cluster.

Note that we do not have a probabilistic model in mind, so “cluster mean” is perhaps a poor

terminology. “Cluster centroid” is better; the idea is that if we knew which data points were assigned to the i th cluster, then the cluster mean would be the centroid (the sample average) of those data points.

We are faced with a “chicken-and-egg” problem—if we knew the assignments we could find the means, or if we knew the means we could find the assignments. The basic idea of the K -means algorithm is to make an initial guess for one of these quantities (the means) and iterate back and forth.

The algorithm maintains two kinds of variables—means and assignments. Let μ_i denote the cluster mean for the i th cluster. For each data point x_n let z_n be an indicator vector that represents the assignment of x_n to one of the clusters. Thus, if the x_n is assigned to the i th cluster, we set the component z_n^i equal to one, and all other components of z_n equal to zero.

The K -means algorithm begins by making some initial assignments for the μ_i , for example taking the μ_i to be given by a subset of the data vectors themselves. The algorithm then alternates between two phases. In the first phase, values for the indicator variables z_n^i are evaluated by assigning each data point x_n to the closest mean μ_i (where distance is typically measured using a simple Euclidean metric) so that, for each n ,

$$z_n^i = \begin{cases} 1 & \text{if } i = \arg \min_j \|x_n - \mu_j\|^2 \\ 0 & \text{otherwise.} \end{cases} \quad (10.10)$$

In the second phase, the values of the means are recomputed by taking μ_i to be equal to the sample mean of those vectors x_n which have been assigned to the i^{th} cluster:

$$\mu_i = \frac{\sum_n z_n^i x_n}{\sum_n z_n^i}. \quad (10.11)$$

The two phases of re-assigning data points to clusters and re-computing the cluster means are repeated in turn until there is no further change in the assignments (or until some maximum number of iterations is exceeded). It is easily seen that the K -means algorithm must converge after a finite number of iterations, since there are only a finite number of possible assignments for the set of discrete variables z_n^i and for each such assignment there is a unique value for the $\{\mu_i\}$.

Although we have motivated the K -means algorithm heuristically, the algorithm can also be motivated as the solution to an optimization problem. In particular, it turns out that the algorithm can be viewed as minimizing the *distortion measure* given by:

$$J = \sum_{n=1}^N \sum_{i=1}^K z_n^i \|x_n - \mu_i\|^2. \quad (10.12)$$

As we ask the reader to show in Exercise ??, Eq. (10.10) is obtained by minimizing J with respect to z_n^i while keeping μ_i fixed, while Eq. (10.11) is obtained by minimizing J with respect to μ_i while keeping z_n^i fixed. Thus K -means can be viewed as a *coordinate descent* algorithm.

The K -means algorithm is illustrated using a simple example in Figure 10.4. The data set, shown in plot (a), consists of 40 data points in two dimensions. We now apply the K -means algorithm, with $K = 2$, using the initial mean vectors shown as the red and blue crosses in plot (b).

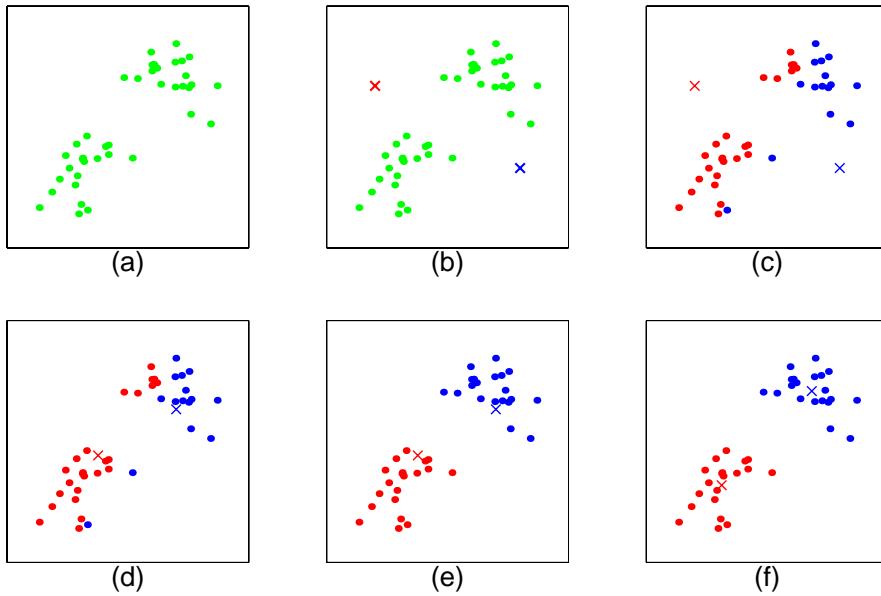


Figure 10.4: Illustration of the K -means algorithm. See the text for a full discussion.

Note that this is in fact a particularly poor initialization and has been chosen in order to provide a clear illustration of the operation of the algorithm. A better initialization would involve selecting random data points as the initial means, and would typically give faster convergence. The first stage of the algorithm involves assigning each data point to one of the two clusters according to its distance to each of the means. This results in the partitioning of the data set shown in plot (c), in which each data point has been colored according to the cluster (red or blue) to which it is assigned. Next we re-compute the mean vectors using the current partitioning, so that the blue mean is reassigned to the mean of the blue-colored data points, and similarly for the red mean, giving the new estimates for the means shown in plot (d). The two phases of the algorithm continue alternately, with repartitioning shown in plot (e), re-assignment in plot (f). On the next re-partitioning the assignment of data points does not change and hence the algorithm has converged.

10.1.3 The EM algorithm

Let us return to the probabilistic framework of mixture models. Adopting the language of the K -means algorithm, let us view the latent variables Z_n as “assignment variables.” These are random variables in the mixture model setting, reflecting our uncertainty about the cluster assignments.

If the Z_n were observed we would have a classification problem in which each data point X_n is assigned a “class label.” The estimate of the mean of the i th Gaussian would simply be the sample mean for the data points in the i th class (cf. Eq. (7.20)):

$$\hat{\mu}_i = \frac{\sum_n z_n^i x_n}{\sum_n z_n^i}. \quad (10.13)$$

This is identical to the K -means update formula, but here we are interpreting the variables z_n^i not as quantities to be manipulated by our algorithm, but rather as observed values of random variables.

Of course, we do not know the values of the Z_n variables. Our approach will be to replace these values with their conditional expectations, conditioning on the data (the x_n values). Recall that we use the notation τ_n^i for these conditional expectations.² We try the following idea—let us replace z_n^i by τ_n^i in Eq. (10.13):

$$\hat{\mu}_i = \frac{\sum_n \tau_n^i x_n}{\sum_n \tau_n^i}. \quad (10.14)$$

Thus, we have replaced a sample mean with a weighted sample mean. Each data point x_n contributes to the estimate of the i th mean in proportion to its posterior probability τ_n^i . The quantity τ_n^i is often referred to as a “soft assignment,” a natural terminology both from the point of view of Eq. (10.14) and the definition of τ_n^i in Eq. (10.7).

We still have a chicken-and-egg problem, however. As seen in Eq. (10.7), the posterior probabilities τ_n^i depend on the parameter estimates, which, according to Eq. (10.14), depend on the posterior probabilities.

Once again, the way out of this chicken-and-egg problem is to start with an initial guess (for the parameters) and to iterate. Given a set of parameters we calculate the posterior probabilities. Given a set of posterior probabilities, we compute new parameter estimates. This is the basic structure of the EM algorithm for Gaussian mixtures.

To clarify, let us augment our notation for τ_n^i to include reference to the iteration number t :

$$\tau_n^{i(t)} = \frac{\pi_i^{(t)} \mathcal{N}(x_n | \mu_i^{(t)}, \Sigma_i^{(t)})}{\sum_j \pi_j^{(t)} \mathcal{N}(x_n | \mu_j^{(t)}, \Sigma_j^{(t)})}, \quad (10.15)$$

where we have also indexed the parameter estimates with a superscript to indicate the iteration number. We now define update equations for all of the parameters—the mixing proportions, the means and the covariance matrices. Motivated by the K -means algorithm we have the following formula for the means:

$$\mu_i^{(t+1)} = \frac{\sum_n \tau_n^{i(t)} x_n}{\sum_n \tau_n^{i(t)}}. \quad (10.16)$$

For the covariance matrices we use an analogous formula:

$$\Sigma_i^{(t+1)} = \frac{\sum_n \tau_n^{i(t)} (x_n - \mu_i^{(t+1)}) (x_n - \mu_i^{(t+1)})^T}{\sum_n \tau_n^{i(t)}}. \quad (10.17)$$

defining the update as a weighted sample covariance, with the posterior probabilities again serving as weights. Finally, viewing $\tau_n^{i(t)}$ as a “soft assignment” of data point x_n to cluster i , it is natural to

²We use the elementary fact that conditional expectations and conditional probabilities are the same for binary-valued variables: $E[Z_n^i | x_n] = p(Z_n^i = 1 | x_n)$.

estimate π_i as the sum of these assignments across the data, divided by the number of data points:

$$\pi_i^{(t+1)} = \frac{1}{N} \sum_n \tau_n^{i(t)}. \quad (10.18)$$

Note that if we sum these estimates $\pi_i^{(t+1)}$ with respect to i we obtain one; thus our “soft counting” has not undercounted or overcounted.

Equations 10.15, 10.18, 10.16, and 10.17 define the EM algorithm for Gaussian mixtures.

The first phase of the algorithm—the calculation of the posterior probability in Eq. (10.15)—is generally referred to as the “Expectation step,” or “E step.” The second phase of the algorithm—the parameter updates in Equations 10.18, 10.16, and 10.17—is generally referred to as the “Maximization step,” or “M step.” The explanation for this choice of terminology will be provided in Chapter 11.

In Figure 10.5 we illustrate the EM algorithm applied to a mixture of Gaussians using the same data set, shown in plot (a), as used to illustrate the K -means algorithm in Figure 10.4. Here a mixture of 2 Gaussians is used, with centers initialized using the same values as for the K -means algorithm, and with covariance matrices initialized to be proportional to the unit matrix. Contours of 1 standard deviation for each of the Gaussian components are shown in plot (b). In plot (c) we show the result of applying the initial E-step, in which points have been colored according to the posterior probabilities for the two components, such that the color ranges from blue to red as the probability $P(\text{blue}|x_n)$ ranges from 1 to 0. We see in plot (c) that some points have a significant probability for belonging to either cluster and so appear purple. Plot (d) shows the result of the first M-step. We see that the mean of, say, the blue Gaussian is moved to the mean of the data set, weighted by the probabilities of each data point belonging to the blue cluster, in other words it moves to the mean of the blue ink. Similarly the covariance of the blue Gaussian becomes the sample covariance of the blue ink, with analogous results for the red component. Subsequent plots show the situation after various numbers L of complete EM cycles. In plot (i) the model is close to the final, converged state. Note that the EM algorithm takes many more iterations to reach (approximate) convergence compared with the K -means algorithm, and that each cycle requires significantly more computation. It is therefore common to run the K -means algorithm in order to find a suitable initialization for a Gaussian mixture model which is subsequently adapted using EM. The covariance matrices can conveniently be initialized to the sample covariances of the clusters found by the K -means algorithm, and the mixing proportions can be set to the fractions of data points assigned to the respective clusters.

10.1.4 Necessary conditions

Although we have defined a simple, intuitively appealing algorithm, it is not yet clear what relationship this algorithm has to the quantity that we are trying to maximize, the log likelihood in Eq. (10.9). In this section and the following section, we take initial steps toward working out this relationship, and in so doing providing a more rigorous justification of the EM algorithm. The full justification will appear in Chapter 11, where we show that the EM algorithm—like the K -means algorithm—is a form of coordinate ascent.

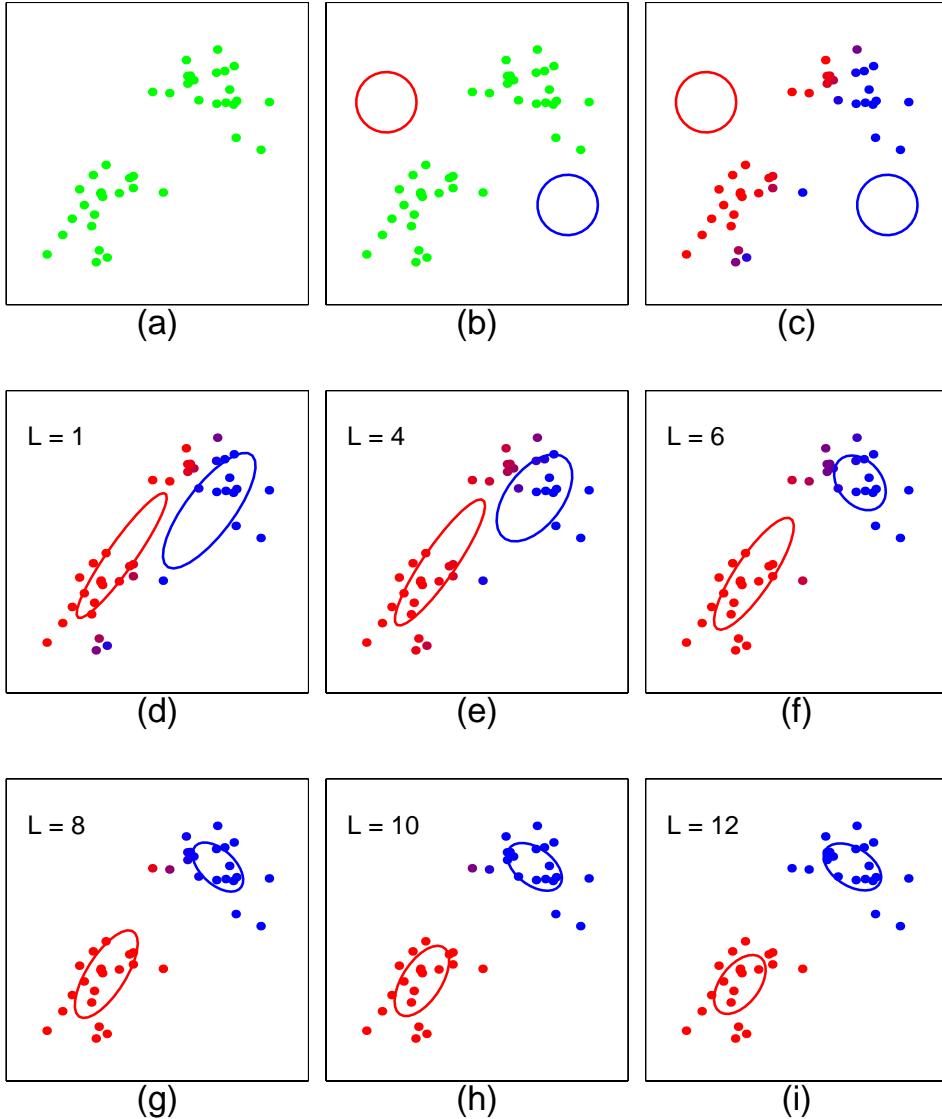


Figure 10.5: Illustration of the EM algorithm using the same data set as used for the illustration of the K -means algorithm in Figure 10.4. The value of L denotes the number of complete EM cycles. See the text for a full discussion.

In this section we write down a set of equations characterizing the stationary points of the log likelihood function. We show that the stationary points can be viewed as fixed points of the EM iteration.

We need to obtain the derivatives of l with respect to the parameters. Let us first take the

derivative with respect to μ_i :

$$\frac{\partial l}{\partial \mu_i} = \frac{\partial}{\partial \mu_i} \left\{ \sum_n \log \sum_i \pi_i \mathcal{N}(x_n | \mu_i, \Sigma_i) \right\} \quad (10.19)$$

$$= \sum_n \frac{\pi_i \mathcal{N}(x_n | \mu_i, \Sigma_i)}{\sum_j \pi_j \mathcal{N}(x_n | \mu_j, \Sigma_j)} \frac{\partial}{\partial \mu_i} \log \mathcal{N}(x_n | \mu_i, \Sigma_i) \quad (10.20)$$

$$= \sum_n \tau_n^i \frac{\partial}{\partial \mu_i} \log \mathcal{N}(x_n | \mu_i, \Sigma_i) \quad (10.21)$$

$$= \sum_n \tau_n^i \Sigma_i^{-1} (x_n - \mu_i). \quad (10.22)$$

Setting to zero yields

$$\mu_i = \frac{\sum_{n=1}^N \tau_n^i x_n}{\sum_{n=1}^N \tau_n^i} \quad (10.23)$$

at a stationary point of the log likelihood.

A very similar calculation yields the following conditions for the covariance matrices:

$$\Sigma_i = \frac{\sum_{n=1}^N \tau_n^i (x_n - \mu_i)(x_n - \mu_i)^T}{\sum_{n=1}^N \tau_n^i} \quad (10.24)$$

and the mixing proportions:

$$\pi_i = \frac{1}{N} \sum_{n=1}^N \tau_n^i, \quad (10.25)$$

where in the latter case we use Lagrange multipliers.

These equations do not of course constitute an explicit solution since the posterior probabilities are themselves functions of the parameters, and so Equations 10.25, 10.23 and 10.24 represent a system of coupled, nonlinear equations. We can, however, attempt to solve these equations iteratively. In particular, given a parameter vector $\theta^{(t)}$, we plug into the right-hand side of Equations 10.25, 10.23 and 10.24 and obtain an updated parameter vector, which we define to be $\theta^{(t+1)}$.

Comparing to Equations 10.16, 10.17, and 10.18, we see that we have derived the EM update equations.

While this derivation of the EM iterations is perhaps preferable to our earlier heuristic arguments, it is still rather heuristic, leaving us with a number of questions regarding convergence. Moreover, the derivation provides us with little insight—the key quantity, the posterior probability τ_n^i , emerges somewhat mysteriously from the algebra. To develop a deeper understanding and to apply the EM algorithm to more general graphical models, we will need some new concepts.

10.1.5 The expected complete log likelihood

In this section we derive the EM equations for the Gaussian mixture model simply and systematically, by introducing a key player in the EM story—the *expected complete log likelihood*. The full

treatment of the role played by the expected complete log likelihood in the EM algorithm will have to wait for Chapter 11, but we provide some initial intuition in this section, paving the way for the general presentation in Chapter 11.

To introduce the key idea, let us pretend for a moment that we are able to observe the latent variables Z_n . This is a pretense, but it will turn out to be a useful pretense. In particular, let us define a (fictional) data set $\mathcal{D}_c = \{(x_n, z_n) : n = 1, \dots, N\}$ that we refer to as the *complete data*.

If we were to actually have such a data set, we would define the following likelihood, which we refer to as the *complete log likelihood*:

$$l_c(\theta | \mathcal{D}_c) = \sum_n \log p(x_n, z_n | \theta) \quad (10.26)$$

$$= \sum_n \log \prod_i [\pi_i \mathcal{N}(x_n | \mu_i, \Sigma_i)]^{z_n^i} \quad (10.27)$$

$$= \sum_n \sum_i z_n^i \log [\pi_i \mathcal{N}(x_n | \mu_i, \Sigma_i)] \quad (10.28)$$

Note the difference between this log likelihood and the original log likelihood for our problem, which we repeat here for convenience:

$$l(\theta | \mathcal{D}) = \sum_n \log \sum_i \pi_i \mathcal{N}(x_n | \mu_i, \Sigma_i). \quad (10.29)$$

In the latter log likelihood, the logarithm is outside of the summation over i , which, as we have remarked before, reflects the fact that the likelihood is a marginal probability. The complete log likelihood, on the other hand, is not a marginal probability, and thus the logarithm is inside the sum. This logarithm acts on the probabilities π_i and $\mathcal{N}(x_n | \mu_i, \Sigma_i)$, leading to the simple maximum likelihood formulas for generative classification that we studied in Chapter 7.

Of course the Z_n variables are not observed. The next step is the key one—as in our earlier discussion let us treat the values z_n in the complete log likelihood as random variables Z_n and take expectations. In calculating these expectations we condition on the observed data x_n , also fixing a particular parameter vector $\theta^{(t)}$. Using the operator notation $\langle \cdot \rangle_{\theta^{(t)}}$ to denote these conditional expectations, we define an important quantity known as the *expected complete log likelihood*:

$$\langle l_c(\theta | \mathcal{D}_c) \rangle_{\theta^{(t)}} = \left\langle \sum_n \sum_i Z_n^i \log [\pi_i \mathcal{N}(x_n | \mu_i, \Sigma_i)] \right\rangle_{\theta^{(t)}} \quad (10.30)$$

$$= \sum_n \sum_i \langle Z_n^i \rangle_{\theta^{(t)}} \log \{\pi_i \mathcal{N}(x_n | \mu_i, \Sigma_i)\} \quad (10.31)$$

$$= \sum_n \sum_i \tau_n^{i(t)} \log \{\pi_i \mathcal{N}(x_n | \mu_i, \Sigma_i)\} \quad (10.32)$$

Comparing Eq. (10.28) and Eq. (10.32), we see that the expected complete log likelihood is obtained from the complete log likelihood by replacing the fictional “observations” z_n^i with the posterior probabilities τ_n^i , where the latter are evaluated using the parameter vector $\theta^{(t)}$.

In general we define the E step of the EM algorithm to be the “calculation of the expected complete log likelihood.” In the Gaussian mixture problem this simply reduces to calculating the posterior probabilities τ_n^i , and it may not be clear why we need the fancier language. In problems with multiple latent variables, however, there are generally interactions to account for, and the preferred method for defining the E step in the general setting is to calculate the expected complete log likelihood.

Moreover, the preferred method for obtaining the M step of an EM algorithm is to maximize the expected complete log likelihood with respect to the parameters. We will explain why this is the case in Chapter 11, but in the meantime let us treat it as a recipe and verify that we obtain the M step updates in Equations 10.18, 10.16, and 10.17 from the expected complete log likelihood in Eq. (10.32).

Let us first consider the update for the means. Collecting together the terms in Eq. (10.32) that depend on μ_i , and denoting the result as $J(\mu_i)$, we obtain:

$$J(\mu_i) = -\frac{1}{2} \sum_n \tau_n^{i(t)} (x_n - \mu_i)^T \Sigma_i^{-1} (x_n - \mu_i). \quad (10.33)$$

We see that we have a weighted least-squares problem. Calculating the derivative of $J(\mu_i)$ with respect to μ_i and setting to zero yields:

$$\mu_i^{(t+1)} = \frac{\sum_n \tau_n^{i(t)} x_n}{\sum_n \tau_n^{i(t)}}, \quad (10.34)$$

which is Eq. (10.16).

Similarly, collecting together the terms that reference the covariance matrix Σ_i , we have:

$$J(\Sigma_i) = -\frac{1}{2} \sum_n \tau_n^{i(t)} \left\{ \log |\Sigma_i| + (x_n - \mu_i)^T \Sigma_i^{-1} (x_n - \mu_i) \right\}. \quad (10.35)$$

This is a weighted variant of the problem of estimating the covariance matrix of a Gaussian. Taking the derivative with respect to Σ_i and setting to zero yields:

$$\Sigma_i^{(t+1)} = \frac{\sum_n \tau_n^{i(t)} (x_n - \mu_i^{(t+1)}) (x_n - \mu_i^{(t+1)})^T}{\sum_n \tau_n^{i(t)}}. \quad (10.36)$$

which is Eq. (10.17).

Finally, the terms in the expected complete log likelihood that reference π are:

$$J(\pi) = \sum_n \sum_i \tau_n^{i(t)} \log \pi_i. \quad (10.37)$$

Adding a Lagrangian term to account for the constraint that the π_i sum to one, taking derivatives and setting to zero yields:

$$\pi_i^{(t+1)} = \frac{1}{N} \sum_n \tau_n^{i(t)}, \quad (10.38)$$

Figure 10.6:

Figure 10.7:

which is Eq. (10.18).

Although this derivation of the EM equations is no more intuitive than our earlier work, it has the virtue of yielding a simple algebraic recipe, both for the E step and the M step. It is also more general than our earlier work; indeed, the derivation that we have carried out here will extend readily to arbitrary graphical models. The key concepts that underly the usefulness of this approach are: (1) the decoupled form of the complete log likelihood, and (2) the linearity of the expectation operator.

This brief discussion suggests an important role for the “expected complete log likelihood,” particularly in providing a simple method for deriving EM update equations, but it still leaves us with a number of questions. How does the maximization of the expected complete log likelihood relate to the maximization of the actual log likelihood, which is after all our goal? We have claimed that the EM algorithm is a coordinate ascent algorithm—how does the expected complete log likelihood emerge in this picture? How general is the algorithm? Will the algorithm converge? These are the topics of Chapter 11. Before turning to these general considerations, however, let us consider another application of mixture model ideas.

10.2 Conditional mixture models

The “divide-and-conquer” approach to statistical modeling applies equally well in the regression and classification domains. In this section we study conditional mixture models, which are the analogs for regression and classification of the unconditional mixture models studied thus far.³

Consider the data set shown in Figure 17.16, where we clearly have a nonlinearity in the mapping from X to Y . We might utilize a rich set of basis functions to allow us to capture this nonlinearity, but it may be difficult to capture the sharp kink in the middle of the figure without requiring an overly large number of basis functions to the left and right of the kink, where the function would seem to be well modeled as a simple linear function.

An alternative way to model these data is to utilize a conditional mixture. Within the conditional mixture framework, we in essence split the problem into two subproblems, each of which can be treated as a simple linear regression. We must find the parameters of these regressions, and we must also decide where to split. Assuming that we can model the splitting decision using a simple parametric model, we may be able to model the overall nonlinear dependence of Y on X using a small number of parameters.

The conditional mixture model is shown as a graphical model in Figure 17.16. The model includes nodes for the observed variables X and Y and also incorporates a node for a multinomial

³Conditional mixture models are also referred to as “mixture of experts” models; where the term “expert” is used to designate a regression, classification, or other generalized regression model.

latent variable Z . The response variable Y is conditioned not only on X but also on the latent variable Z . This latent variable indexes the set of possible regressions of Y on X —for each value of Z , we obtain a possibly different parameterized regression. Note moreover that there is a link from X to Z . It is this dependency that allows us to obtain different regressions in different regions of the input space.

Let us consider how to parameterize each of the nodes in the model. Z is a multinomial variable, and its parent is the input variable X . The fact that the edge between these nodes points from X to Z suggests using the ideas that we discussed in the section on discriminative classification (Section ??) to parameterize the dependency. For example, we may use a softmax regression:

$$p(Z^i = 1 | x, \xi) = \frac{e^{\xi_i^T x}}{\sum_j e^{\xi_j^T x}}, \quad (10.39)$$

where $\xi = (\xi_1, \xi_2, \dots, \xi_M)$ is a parameter vector.

The node Y has X and Z as parents, and thus we have a conditional probability $p(Y | X, Z^i = 1, \theta_i)$. The mathematical form of this conditional probability depends on the nature of the data Y . Let us not specify a particular model at this point, but assume that we will bring to bear the machinery of generalized linear models (GLIM's). For example we could consider a binary classification model in which $p(Y | X, Z^i = 1, \theta_i)$ is a logistic regression model. Note that we have one such model for each value of Z .

As is usual in the regression or discriminative classification setting we treat the observations of X as fixed constants. Thus we do not incorporate a marginal probability for X into our model.

Putting together the pieces, we obtain the following model for the conditional probability of Y given X :

$$p(y | x, \theta) = \sum_i p(Z^i = 1 | x, \xi) p(y | Z^i = 1, x, \theta_i), \quad (10.40)$$

where $\theta = (\xi_1, \dots, \xi_M, \theta_1, \dots, \theta_M)$. This is a conditional mixture model, where both the *mixing proportions*, $p(Z^i = 1 | x, \xi)$, and the *mixture components*, $p(y | Z^i = 1, x, \theta_i)$, are conditional probabilities—both are conditioned on $\{X = x\}$.

As in the unconditional mixtures, a key quantity in conditional mixture modeling is the posterior probability of the latent variable Z . Here the notions of “prior” and “posterior” are relative to the observation of Y ; the variable X is taken to be always observed. Thus, let us define $\pi_i(x, \xi) \triangleq p(Z^i = 1 | x, \xi)$ as the *prior probability* of the i th mixture component, conditioned solely on X . We now define the *posterior probability* $\tau^i(x, y, \theta)$:

$$\tau^i(x, y, \theta) \triangleq p(Z^i = 1 | x, y, \xi) \quad (10.41)$$

$$= \frac{p(Z^i = 1 | x, \xi) p(y | Z^i = 1, x, \theta_i)}{\sum_j p(Z^j = 1 | x, \xi) p(y | Z^j = 1, x, \theta_j)} \quad (10.42)$$

$$= \frac{\pi_i(x, \xi) p(y | Z^i = 1, x, \theta_i)}{\sum_j \pi_j(x, \xi) p(y | Z^j = 1, x, \theta_j)} \quad (10.43)$$

where we see that the prior probability of the i th class is updated by how probable the observation $\{Y = y\}$ is under the i th model.

Figure 10.8:

10.2.1 Examples

Let us consider some specific choices for the mixture components $p(y | Z^i = 1, x, \theta_i)$.

Mixtures of linear regressions

For continuous variables Y it is natural to consider a mixture of linear regressions:

$$p(y | x, \theta) = \sum_i \pi_i(x, \xi) \mathcal{N}(y | \beta_i^T x, \sigma_i^2), \quad (10.44)$$

where we have allowed each linear regression to have a possibly different error variance σ_i^2 .

Figure 17.16 shows a depiction of this model in the case of two mixture components. In this case the variable Z can be taken to be a binary variable and the mixing proportion $\pi(x, \xi) \triangleq p(Z = 1 | x, \xi)$ can be modeled using logistic regression. The logistic curve shown in the figure thus models the input-dependent probability associated with the two mixture components. For negative values of X , the logistic curve assigns small probability to $Z = 1$, thus essentially choosing the regression curve labeled $Z = 0$. For positive values of X , the logistic curve assigns large probability to $Z = 1$, thus essentially choosing the regression curve labeled $Z = 1$. The point at which the logistic function is equal to 0.5 can be viewed as the “split” point.

Let us now consider the geometric interpretation of the posterior probability. As shown in Figure 17.16, conditioning on a specific value of X leaves us with two possible conditional expectations of Y —the conditional expectations associated with each of the two regressions. We have Gaussian distributions around each of these conditional expectations, with variances σ_1^2 and σ_0^2 , respectively. Thus the conditional distribution of Y given X is bimodal—we have a mixture distribution in the output space for each point in the input space. Consider now the point (x, y') in the figure. For this value of x , the prior $\pi(x, \xi)$ is large, corresponding to a choice of the regression curve labeled $Z = 1$. Moreover, the value y' has high probability under this regression model and the corresponding posterior $\tau(x, y', \xi)$ is therefore large. Consider, on the other hand, the point (x, y'') . The prior for this point is the same as before. The value y'' , however, has low probability under the regression model labeled $Z = 1$ and high probability under the regression model labeled $Z = 0$. The posterior $\tau(x, y'', \xi)$ is therefore small, corresponding to a posterior choice of the regression model labeled $Z = 0$.

We see that the posterior probability has much the same interpretation as in the unconditional mixture model setting. That is, we can interpret the posterior probability as a “soft assignment,” but here the assignment process reflects both the prior partitioning of the input space into regions, modeled by $\pi_i(x, \xi)$, and the ability of each of the component regressions to accommodate the observed output y at that given value of x , modeled by $p(y | Z^i = 1, x, \theta_i)$.

Mixtures of logistic regressions

We can readily extend the model of the previous section to mixtures of generalized linear models, thereby accommodating a wide variety of data types. An example is the mixture of logistic regressions:

$$p(y | x, \theta) = \sum_i \pi_i(x, \xi) \mu(\theta_i^T x)^y (1 - \mu(\theta_i^T x))^{1-y}, \quad (10.45)$$

where $\mu(\theta_i^T x)$ is the logistic function: $\mu(\theta_i^T x) = 1/(1 + e^{-\theta_i^T x})$. This model allows us to bring the divide-and-conquer approach to bear on classification problems.

The interpretation of the prior and posterior probabilities is identical in this model to the linear regression case, with the likelihood being a Bernoulli distribution rather than a Gaussian distribution.

10.2.2 Parameter estimation via the EM algorithm

At this point we have parameterized the graphical model in Figure 17.16, and the problem of maximum likelihood parameter estimation can be handled straightforwardly using the tools that we have developed in Section 10.1.3 and Section 10.1.5. Indeed the model is a good exercise of our skills. In this section we write down the log likelihood, the expected complete log likelihood, and the EM algorithm for conditional mixtures.

We assume an IID data set $\mathcal{D} = \{(x_n, y_n) : n = 1, \dots, N\}$. The likelihood is obtained as the sum of the logarithm of the probability model in Eq. (10.40):

$$l(\theta | \mathcal{D}) = \sum_n \log \sum_i \pi_i(x_n, \xi) p(y_n | Z^i = 1, x_n, \theta_i). \quad (10.46)$$

Note the presence of the logarithm outside of the summation; as in the unconditional case, the likelihood is a marginal probability.

The “complete data” is the data set $\mathcal{D}_c = \{(x_n, z_n, y_n) : n = 1, \dots, N\}$, where as before we imagine that we can observe the latent variable Z . The likelihood for N complete observations of the model is:

$$l(\theta | \mathcal{D}_c) = \prod_n \prod_i [\pi_i(x_n, \xi) p(y_n | Z^i = 1, x_n, \theta_i)]^{z_n^i} \quad (10.47)$$

and taking the logarithm yields the complete log likelihood:

$$l_c(\theta | \mathcal{D}_c) = \sum_n \sum_i z_n^i \log [\pi_i(x_n, \xi) p(y_n | Z^i = 1, x_n, \theta_i)], \quad (10.48)$$

where we now see the logarithm inside the summation.

We take the expectation of the complete log likelihood, where we now treat the variables Z_n^i as random variables. The expectation is taken with respect to the conditional probability distribution $p(z | x, y, \theta^{(t)})$. Given that the complete log likelihood is linear in z_n^i , we see that we can obtain the expectation by simply computing:

$$\langle Z_n^i \rangle_{\theta^{(t)}} = p(Z_n^i = 1 | x_n, y_n, \theta^{(t)}) \quad (10.49)$$

$$= \tau^i(x_n, y_n, \theta^{(t)}). \quad (10.50)$$

Thus we see that the E step of the EM algorithm amounts to computing the posterior probabilities $\tau^i(x_n, y_n, \theta^{(t)})$. These can be viewed as our “best guess” of the values of the latent variables Z_n , conditioned on the observed values x_n and y_n , and evaluated at the current value of the parameter vector $\theta^{(t)}$.

To summarize, the expected complete log likelihood takes the following form:

$$l_c(\theta | \mathcal{D}) = \sum_n \sum_i \tau_n^i(t) \log [\pi_i(x_n, \xi) p(y_n | Z^i = 1, x_n, \theta_i)], \quad (10.51)$$

where we write $\tau_n^{i(t)}$ for the posterior probability $\tau^i(x_n, y_n, \theta^{(t)})$, in order to simplify notation.

With the expected complete log likelihood in hand, we can now turn to the M step. Let us first consider maximizing l_c with respect to the parameters ξ . Collecting together the terms that depend on ξ , and referring to the result as $J(\xi)$, we have:

$$J(\xi) = \sum_n \sum_i \tau_n^i(t) \log \pi_i(x_n, \xi). \quad (10.52)$$

This is identical to the log likelihood for the discriminative classification problem (cf. Eq. (??)), where the role of the class labels in that problem (the z_n^i) is now played by the posterior probabilities (the $\tau_n^{i(t)}$). The interpretation is that we “fill in” the values of the latent variables Z_n^i with our “best guess.” Based on these filled-in values we treat the problem of estimating the parameters of the conditional probability $p(Z | x, \xi)$ as a discriminative classification problem. In particular we can use the IRLS algorithm to update these parameters.

It is also straightforward to derive an M step for the parameters θ_i . Collecting together the terms in the expected complete log likelihood that depend on θ_i , and referring to the result as $J(\theta_i)$, we obtain:

$$J(\theta_i) = \sum_n \tau_n^i(t) \log p(y_n | Z^i = 1, x_n, \theta_i). \quad (10.53)$$

For generalized linear models, the log probability in this expression is the logarithm of an exponential family distribution. Each data point, (x_n, y_n) , has an associated “weight,” the posterior probability $\tau_n^i(t)$. Thus we have a weighted maximum likelihood problem to solve. In essence, each data point is “assigned” to one of the mixture components, and the estimation of the parameters of each mixture component is carried out using the data points assigned to that component.

In the case of a mixture of linear regressions, we obtain a set of weighted least squares problems, one for each mixture component. For a mixture of logistic regressions, we have a set of weighted cross-entropies. In general we can treat all of these problems within the IRLS framework—recall our discussion of weighted IRLS in Section ??.

In summary, the EM algorithm for conditional mixtures takes the following form:

- (E step): Calculate the posterior probabilities $\tau_n^{i(t)}$.
- (M step): Use the IRLS algorithm to update the parameters ξ , based on data pairs $(x_n, \tau_n^{i(t)})$.
- (M step): Use the weighted IRLS algorithm to update the parameters θ_i , based on data pairs (x_n, y_n) , with weights $\tau_n^{i(t)}$.

These steps iterate and, as we prove in Chapter 11, climb to a local maximum of the likelihood.

10.2.3 An on-line algorithm

We can obtain some additional insight into the conditional mixture model by developing an on-line estimation algorithm. As we discussed in Chapter 6, the problem here is to derive an update for the parameters based on a single data point.

To derive these updates we take the derivative of the log likelihood with respect to the parameters and delete the summation over n ; this yields the “stochastic gradient.” We omit the algebra, asking the reader to supply the details in Exercise ??.

In the equations below, we use the notation $\mu_n^i(t)$ to denote the conditional expectation of Y given $\{X = x\}$, for the i th mixture component, the n th data point, and letting the parameter vector equal $\theta^{(t)}$. We assume that the canonical link function has been chosen.

Taking the derivative with respect to θ_i , we obtain the following update equation:

$$\theta_i^{(t+1)} = \theta_i^{(t)} + \rho \tau_n^{i(t)} (y_n - \mu_n^i(t)) x_n, \quad (10.54)$$

where ρ is a step size. Similarly, taking the derivative with respect to ξ_i , we obtain:

$$\xi_i^{(t+1)} = \xi_i^{(t)} + \kappa (\tau_n^{i(t)} - \pi_i(x_n, \xi^{(t)})) x_n, \quad (10.55)$$

where κ is a step size.

Both of these equations have natural interpretations. The update of θ_i in Eq. (10.54) has the form of the LMS algorithm, but with the additional feature that the step size is modulated by the posterior probability $\tau_n^{i(t)}$. Thus, if our current “best guess” is that the n th data point should be assigned to the i th mixture component, then we update the parameters in the normal way. If, on the other hand, we do not think that the n th data point should be assigned to the i th mixture component, then the step size is near zero and the parameters are not adjusted.

The update of ξ_i in Eq. (10.54) also has an appealing interpretation. The update again takes the form of the LMS algorithm, where the error is the difference between the posterior probability and the prior probability. In essence we have a classification problem in which the prior probability is a prediction of the class label associated with the mixture components, and the posterior probability is an improved estimate of that label.

Figure 17.16 shows a depiction of these update equations for the case of a mixture of linear regressions. In this case $\mu_n^i(t) = \theta_i^{(tT)} x_n$, and the error which drives the update of θ_i is simply the difference $(y_n - \theta_i^{(tT)} x_n)$. This difference is shown in the figure for both of the regressions. Although the error is larger for the $Z = 0$ regression, the posterior probability associated with this regression is vanishingly small (it is proportional to the exponential of the negative of the square of the error). Thus, as we show in Figure 17.16(b), the parameters associated with the upper ($Z = 1$) curve change significantly, while the parameters associated with the lower ($Z = 0$) curve change little.

Given that the posterior probability associated with the upper curve is near one, the error in Eq. (10.55) has the effect of shifting the logistic curve towards the left. As we show in Figure 17.16(b), this implies that on future presentations of this data point, the prior prediction will be closer to one.

10.2.4 Hierarchical conditional mixtures and decision trees

10.3 Appendix XXX

As an alternative to the EM algorithm, we can use standard nonlinear optimization algorithms such as conjugate gradients. In this appendix we discuss this approach for unconditional mixtures, focusing on the problem of implementing the probabilistic constraints on the parameters.

In evaluating the derivatives we must take account of the requirement for the mixing proportions to satisfy $0 \leq \pi_i \leq 1$ and $\sum_i \pi_i = 1$. Similarly, the covariance matrices Σ_i must remain symmetric and positive-definite.

We can allow for the constraints on the mixing proportions π_i by expressing them as a nonlinear transformation of a corresponding set of unconstrained variables η_i . Specifically, we use the softmax transformation:

$$\pi_i = \frac{\exp(\eta_i)}{\sum_j \exp(\eta_j)}, \quad (10.56)$$

which has the property that the mixing proportions will automatically satisfy the required constraints.

In order to impose the required constraints on the covariance matrix we can represent the inverse⁴ covariance matrix in terms of its Cholesky decomposition:

$$\Sigma^{-1} = A^T A \quad (10.57)$$

where A is an upper diagonal matrix, so that $A_{ij} = 0$ if $i < j$. It is easily seen that there are $d(d + 1)/2$ remaining independent elements in A , corresponding to the number of independent elements in Σ . In computing the expected complete-data log likelihood we need the inverse square root of the determinant of the covariance matrix, which is given by:

$$|\Sigma|^{-1/2} = \prod_{i=1}^d A_{ii}. \quad (10.58)$$

The covariance matrix will be positive definite provided the diagonal elements A_{ii} are positive, which can be ensured by writing them as the exponentials of real values $A_{ii} = \exp(\alpha_i)$.

In summary, if we treat the values of η_i , A_{ij} (for $j > i$), α_i and the components of μ_i as independent, unconstrained real values, the required constraints will be met. The required derivatives of the log likelihood with respect to these unconstrained variables are then easily obtained Exercise ??.

⁴The inverse of a positive definite symmetric matrix is also positive definite and symmetric.

Chapter 11

The EM algorithm

The expectation-maximization (EM) algorithm provides a general approach to the problem of maximum likelihood parameter estimation in statistical models with latent variables. We have already seen two examples of the EM approach at work in the previous chapter. While these examples are revealing ones, it is important to understand that EM applies much more widely. Indeed, the EM approach goes hand-in-glove with general graphical model machinery, taking advantage of the conditional independence structure of graphical models in a systematic way. As such it occupies a central place in the book.

While in principle one can treat ML parameter estimation as a simple matter of passing a likelihood function to a black-box numerical optimization routine, in practice one would like to take advantage of the structure embodied in the model to break the optimization problem into more manageable pieces. EM provides a systematic way to implement such a divide-and-conquer strategy. As we will see, in this chapter and in later chapters, this approach leads to conceptual clarity and simplicity of algorithmic implementation. It also provides a guide to dealing with models in which issues of computational complexity begin to arise. Indeed, EM will provide a guide to dealing with problems in which the mere calculation of the likelihood or its derivatives appear to be intractable computational challenges.

The main goal of this short chapter is to present a general formulation of the EM algorithm. We show that EM is a rather simple optimization algorithm—it is *coordinate ascent* on an appropriately defined function. Thus, both the E step and the M step can be viewed as maximizations in an abstract space. We show how the *expected complete log likelihood* emerges from this perspective; in particular, we show how the maximization operation that defines the E step can also be viewed as an expectation. We also take the coordinate ascent story a bit further, showing that EM can be viewed as an *alternating minimization algorithm*—a special form of coordinate descent in a Kullback-Leibler divergence.

Finally, we sketch how the EM algorithm applies in the general setting of graphical models. Subsequent chapters will provide many examples of applications to graphical models and will fill in the various details appropriate to these special cases.

11.1 Latent variables and parameter estimation

Recall that latent or hidden variables are generally introduced into a model in order to simplify the model in some way. We may observe a complex pattern of dependency among a set of variables $x = (x_1, \dots, x_m)$. Rather than modeling this dependency directly, via edges linking these variables, we may find it simpler to account for their dependency via “top-down” dependency on a latent variable z . In the simplest case, we may find it possible to assume that the x_i are conditionally independent given z , and thus restrict our model to edges between the node z and the nodes x_i .

If the latent variables in the model could be observed, then generally the parameter estimation problem would be simplified as well. Indeed, this is one way of characterizing what we mean by the simplification achieved by introducing latent variables into a model. For example, in the case of the mixture of Gaussians model, if we could observe a class label corresponding to each data point, then we would break the data into classes and estimate the mean and covariance matrix separately for each class. The estimation problem would decouple.

But the latent variables are not observed, and this implies that the likelihood function is a marginal probability, obtained by summing or integrating over the latent variables. Marginalization couples the parameters and tends to obscure the underlying structure in the likelihood function.

The EM algorithm essentially allows us to treat latent variable problems using complete data tools, skirting the fact that the likelihood is a marginal probability and exploiting to the fullest the underlying structure induced by the latent variables. EM is an iterative algorithm, consisting of a linked pair of steps. In the *expectation step (E step)*, the values of the unobserved latent variables are essentially “filled in,” where the filling-in is achieved by calculating the probability of the latent variables, given the observed variables and the current values of the parameters.¹ In the *maximization step (M step)*, the parameters are adjusted based on the filled-in variables, a problem which is essentially no more complex than it would be if the latent variables had been observed.

11.2 The general setting

Let X denote the observable variables, and let Z denote the latent variables. Often, X and Z decompose into sets of independent, identically-distributed (IID) pairs, in particular X can often be written as $X = (X_1, X_2, \dots, X_N)$, where the X_i are IID variables and the observed data, $x = (x_1, x_2, \dots, x_N)$, are the observed values of X . We do not need to make this assumption, however, and indeed we will see many non-IID examples in later chapters. Thus, X represents the totality of observable variables and x is the entire observed dataset. Similarly Z represents the set of all latent variables. The probability model is $p(x, z | \theta)$.

If Z could be observed, then the ML estimation problem would amount to maximizing the quantity:

$$l_c(\theta; x, z) \triangleq \log p(x, z | \theta), \quad (11.1)$$

¹We will see that a better way to express this is that in the E step we compute certain expected sufficient statistics, which in the case of multinomial variables reduces to computing the probability of the latent variables. But let us stick with the intuitive and picturesque language of “filling-in” for now.

which is referred to in the context of the EM algorithm as the *complete log likelihood*. If the probability $p(x, z | \theta)$ factors in some way, such that separate components of θ occur in separate factors, then the operation of the logarithm has the effect of separating the likelihood into terms that can be maximized independently. As we discussed in Chapter 9, this is what we generally mean by “decoupling” the estimation problem.

Given that Z is not in fact observed, the probability of the data x is a marginal probability, and the log likelihood (referred to in this context as the *incomplete log likelihood*) takes the following form:

$$l(\theta; x) = \log p(x | \theta) = \log \sum_z p(x, z | \theta), \quad (11.2)$$

where here as in the rest of the chapter we utilize summation to stand for marginalization—the derivation goes through without change if we integrate over continuous z . The logarithm on the right-hand side is separated from $p(x, z | \theta)$ by the summation sign, and the problem does not decouple. It is not clear how to exploit the conditional independence structure that may be present in the probability model.

Let us not give up the hope of working with the complete log likelihood. Given that Z is not observed, the complete log likelihood is a random quantity, and cannot be maximized directly. But suppose we average over z to remove the randomness, using an “averaging distribution” $q(z | x)$. That is, let us define the *expected complete log likelihood*:

$$\langle l_c(\theta; x, z) \rangle_q \triangleq \sum_z q(z | x, \theta) \log p(x, z | \theta), \quad (11.3)$$

a quantity that is a deterministic function of θ . Note that the expected complete log likelihood is linear in the complete log likelihood and thus should inherit its favorable computational properties. Moreover, if q is chosen well, then perhaps the expected complete log likelihood will not be too far from the log likelihood and can serve as an effective surrogate for the log likelihood. While we cannot hope that maximizing this surrogate will yield a value of θ that maximizes the likelihood, perhaps it will represent an improvement from an initial value of θ . If so then we can iterate the process and hill-climb. This is the basic idea behind the EM algorithm.

We begin the derivation of the EM algorithm by showing that an averaging distribution $q(z | x)$ can be used to provide a lower bound on the log likelihood. Consider the following line of argument:

$$l(\theta; x) = \log p(x | \theta) \quad (11.4)$$

$$= \log \sum_z p(x, z | \theta) \quad (11.5)$$

$$= \log \sum_z q(z | x) \frac{p(x, z | \theta)}{q(z | x)} \quad (11.6)$$

$$\geq \sum_z q(z | x) \log \frac{p(x, z | \theta)}{q(z | x)} \quad (11.7)$$

$$\triangleq \mathcal{L}(q, \theta), \quad (11.8)$$

where the last line defines the function $\mathcal{L}(q, \theta)$, a function that we will refer to as an *auxiliary function*.² In Eq. 11.7 we have used Jensen’s inequality, a simple consequence of the concavity of the logarithm function (see Appendix XXX). What we have shown is that—for an arbitrary distribution $q(z|x)$ —the auxiliary function $\mathcal{L}(q, \theta)$ is a lower bound for the log likelihood.

The EM algorithm is a coordinate ascent algorithm on the function $\mathcal{L}(q, \theta)$. At the $(t+1)$ st iteration, we first maximize $\mathcal{L}(q, \theta^{(t)})$ with respect to q . For this optimizing choice of averaging distribution $q^{(t+1)}$, we then maximize $\mathcal{L}(q^{(t+1)}, \theta)$ with respect to θ , which yields the updated value $\theta^{(t+1)}$. Giving these steps their traditional names, we have:

$$\text{(E step)} \quad q^{(t+1)} = \arg \max_q \mathcal{L}(q, \theta^{(t)}) \quad (11.9)$$

$$\text{(M step)} \quad \theta^{(t+1)} = \arg \max_{\theta} \mathcal{L}(q^{(t+1)}, \theta). \quad (11.10)$$

We will soon explain why the first step can be referred to as an “expectation step.” We will also explain how a procedure based on maximizing a lower bound on the likelihood $l(\theta; x)$ can maximize the likelihood itself.

The first important point to note is that the M step is equivalently viewed as the maximization of the expected complete log likelihood. To see this, note that the lower bound $\mathcal{L}(q, \theta)$ breaks into two terms:

$$\mathcal{L}(q, \theta) = \sum_z q(z|x) \log \frac{p(x, z|\theta)}{q(z|x)} \quad (11.11)$$

$$= \sum_z q(z|x) \log p(x, z|\theta) - \sum_z q(z|x) \log q(z|x) \quad (11.12)$$

$$= \langle l_c(\theta; x, z) \rangle_q - \sum_z q(z|x) \log q(z|x), \quad (11.13)$$

and that the second term is independent of θ . Thus, maximizing $\mathcal{L}(q, \theta)$ with respect to θ is equivalent to maximizing $\langle l_c(\theta; x, z) \rangle_q$ with respect to θ .

Let us now consider the E step, the maximization of $\mathcal{L}(q, \theta^{(t)})$ with respect to the averaging distribution q . This maximization problem can be solved once and for all; indeed, we can verify that the choice $q^{(t+1)}(z|x) = p(z|x, \theta^{(t)})$ yields the maximum. To see this, evaluate $\mathcal{L}(q, \theta^{(t)})$ for this choice of q :

$$\mathcal{L}(p(z|x, \theta^{(t)}), \theta^{(t)}) = \sum_z p(z|x, \theta^{(t)}) \log \frac{p(x, z|\theta)}{p(z|x, \theta^{(t)})} \quad (11.14)$$

$$= \sum_z p(z|x, \theta^{(t)}) \log p(x|\theta^{(t)}) \quad (11.15)$$

$$= \log p(x|\theta^{(t)}) \quad (11.16)$$

$$= l(\theta^{(t)}; x). \quad (11.17)$$

Given that $l(\theta; x)$ is an upper bound for $\mathcal{L}(q, \theta^{(t)})$, this shows that $\mathcal{L}(q, \theta^{(t)})$ is maximized by setting $q(z|x)$ equal to $p(z|x, \theta^{(t)})$.

²Note that $\mathcal{L}(q, \theta)$ is a function of x as well. We omit this dependence, however, to lighten the notation.

There is slightly different way to show this result. We first show that the difference between $l(\theta; x)$ and $\mathcal{L}(q, \theta)$ is a Kullback-Leibler (KL) divergence:

$$l(\theta; x) - \mathcal{L}(q, \theta) = l(\theta; x) - \sum_z q(z|x) \log \frac{p(x, z|\theta)}{q(z|x)} \quad (11.18)$$

$$= \sum_z q(z|x) \log p(x|\theta) - \sum_z q(z|x) \log \frac{p(x, z|\theta)}{q(z|x)} \quad (11.19)$$

$$= \sum_z q(z|x) \log p(x|\theta) - \log \frac{q(z|x)}{p(z|x, \theta)} \quad (11.20)$$

$$= D(q(z|x) \| p(z|x, \theta)). \quad (11.21)$$

In Appendix XXX we show that the KL divergence is nonnegative (a simple consequence of Jensen's inequality), and that the KL divergence is uniquely minimized by letting $q(z|x)$ equal $p(z|x, \theta^{(t)})$. Since minimizing the difference between $l(\theta; x)$ and $\mathcal{L}(q, \theta)$ is equivalent to maximizing $\mathcal{L}(q, \theta)$, we again have our result.

The conditional distribution $p(z|x, \theta^{(t)})$ is an intuitively appealing choice of averaging distribution. Given the model $p(x, z|\theta^{(t)})$, a link between the observed data and the latent variables, the conditional $p(z|x, \theta^{(t)})$ is our "best guess" as to the values of the latent variables, conditioned on the data x . What the EM algorithm does is to use this "best guess" distribution to calculate an expectation of the complete log likelihood. The M step then maximizes this expected complete log likelihood with respect to the parameters to yield new values $\theta^{(t+1)}$. We then presumably have an improved model, and we can now make a "better guess" $p(z|x, \theta^{(t+1)})$, which is used as the averaging distribution in a subsequent EM iteration.

What is the effect of an EM iteration on the log likelihood $l(\theta; x)$? In the M step, we choose the parameters so as to increase a lower bound on the likelihood. Increasing a lower bound on a function does not necessarily increase the function itself, if there is a gap between the function and the bound. In the E step, however, we have closed the gap by an appropriate choice of the q distribution. That is, we have:

$$l(\theta^{(t)}; x) = \mathcal{L}(q^{(t+1)}, \theta^{(t)}), \quad (11.22)$$

by Eq. 11.17, and thus an M-step increase in $\mathcal{L}(q^{(t+1)}, \theta)$ will also increase $l(\theta; x)$.

In summary, we have shown that the EM algorithm is a hill-climbing algorithm in the log likelihood $l(\theta; x)$. The algorithm achieves this hill-climbing behavior indirectly, by coordinate ascent in the auxiliary function $\mathcal{L}(q, \theta)$. The advantage of working with the latter function is that it involves maximization of the expected complete log likelihood rather than the log likelihood itself, and, as we have seen in examples, this is often a substantial simplification.

11.3 EM and alternating minimization

We can put our results in a slightly more elegant form by working with KL divergences rather than likelihoods.

Recall that in Chapter 8 we noted a simple equivalence between maximization of the likelihood and minimization of the KL divergence between the empirical distribution and the model. Let us return to that equivalence, and bound the KL divergence rather than the log likelihood. We have:

$$D(\tilde{p}(x) \parallel p(x|\theta)) = -\sum_x \tilde{p}(x) \log p(x|\theta) + \sum_x \tilde{p}(x) \log \tilde{p}(x) \quad (11.23)$$

$$\leq -\sum_x \tilde{p}(x) \mathcal{L}(q, \theta) + \sum_x \tilde{p}(x) \log \tilde{p}(x) \quad (11.24)$$

$$= -\sum_x \tilde{p}(x) \sum_z q(z|x) \log \frac{p(x, z|\theta)}{q(z|x)} + \sum_x \tilde{p}(x) \log \tilde{p}(x) \quad (11.25)$$

$$= \sum_x \tilde{p}(x) \sum_z q(z|x) \log \frac{\tilde{p}(x)q(z|x)}{p(x, z|\theta)} \quad (11.26)$$

$$= D(\tilde{p}(x)q(z|x) \parallel p(x, z|\theta)). \quad (11.27)$$

We see that the KL divergence between the empirical distribution and the model—the quantity that we wish to minimize—is upper bounded by a “complete KL divergence,” a KL divergence between joint distributions on (x, z) .

The term $\sum_x \tilde{p}(x) \log \tilde{p}(x)$ is independent of q and θ and its inclusion in the problem therefore does not change any of our previous results. In particular, minimizing the complete KL divergence with respect to q and θ is equivalent to maximizing the auxiliary function $\mathcal{L}(q, \theta)$ with respect to these variables. We can therefore reformulate the EM algorithm in terms of the KL divergence. Defining $D(q \parallel \theta) \triangleq D(\tilde{p}(x)q(z|x) \parallel p(x, z|\theta))$ as a convenient shorthand, we have:

$$(\text{E step}) \quad q^{(t+1)}(z|x) = \arg \min_q D(q \parallel \theta^{(t)}) \quad (11.28)$$

$$(\text{M step}) \quad \theta^{(t+1)} = \arg \min_{\theta} D(q^{(t+1)} \parallel \theta) \quad (11.29)$$

We see that EM is a special kind of coordinate descent algorithm—an *alternating minimization* algorithm. We alternate between minimizing over the arguments of a KL divergence.

The alternating minimization perspective and the auxiliary function perspective are essentially the same, and the choice between the two is largely a matter of taste. We will see, however, in Chapter 20, that the alternating minimization view allows us to provide a geometric interpretation of EM as a sequence of projections between manifolds—a perspective reminiscent of our presentation of the LMS algorithm in Chapter 6.

11.4 EM, sufficient statistics and graphical models

[Section not yet written.]

11.5 Historical remarks and bibliography

Chapter 12

Hidden Markov Models

In this chapter we finally relax the assumption of independent, identically distributed (IID) sampling that we have labored under until now. A hidden Markov model (HMM) is a graphical model that is appropriate for modeling sequential data; i.e., data sets in which successive samples are no longer assumed to be independent.

An HMM is a natural generalization of a mixture model; indeed, it is perhaps best viewed as a “dynamical” mixture model. To reflect this point of view, we adjust our terminology somewhat, referring to the “mixture components” of the mixture model as “states.” To see exactly what kind of generalization is involved, let us recall the process of generating IID data under a mixture model, using the new language (cf. Figure 12.1(a)):

- At each step, a state is selected according to the distribution $p(z)$. This selection is made independently of the choice of states at other steps.
- Given the state, a data vector is chosen from a distribution $p(x|z)$.

Within the HMM framework we no longer assume that the states are chosen independently at each step, but rather we assume that the choice of a state at a given step depends on the choice of the state at the previous step. Thus we augment the basic mixture model to include a matrix of *transition probabilities* linking the states at neighboring steps. If there are M states, then this is an $M \times M$ matrix, whose (i, j) th entry represents the probability of transitioning from the i th state at a given step to the j th state at the following step. The process of generating data under the HMM is suggested in Figure 12.1(b), where we have drawn arrows between the probability distributions labeled by the states to suggest the transition probabilities. Other than the introduction of a state transition matrix, the HMM is the same as the simpler mixture model—in particular, given the state at a given step a data vector is generated from a distribution that depends only on that state.

As in any mixture model, the states underlying the data generation process are assumed to be “hidden” from the learner. We envision an HMM-based learning system observing the pattern of data in Figure 12.1(a)—one data point at a time—and interpreting the sequence in terms of the hypothesized states and state transitions of Figure 12.1(b). Just as in the simpler mixture model, the fact that the data form clusters is grist for the HMM mill, allowing the learner to differentiate the states. But while the clustering of the data is necessary for an HMM-based learner to be

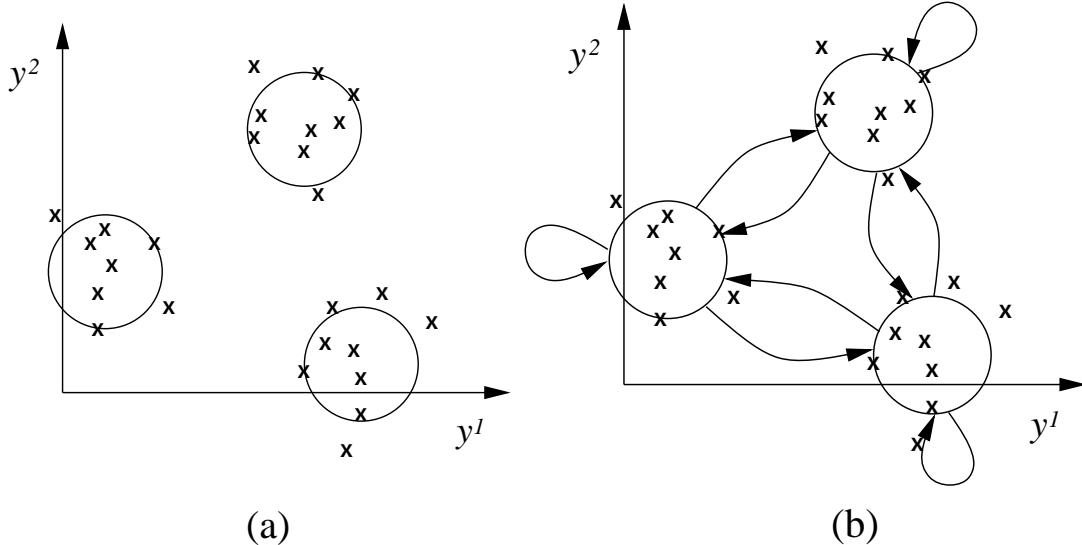


Figure 12.1: (a). A sample point is generated from a mixture model by first selecting a mixture component and then generating a data point from that mixture component. (b) An HMM generalizes the mixture model by allowing the choice of the mixture component at a given step to depend on the choice of the mixture component at the previous step. The arrows in the diagram represent these transitions between the mixture components.

justified in a given problem, it is not sufficient—there should also be regularities in the transitions between clusters.

The inference problem for HMMs involves taking as input the sequence of observed data and yielding as output a probability distribution on the underlying states. Given the dependence between the states, this problem is substantially more complex than the analogous inference problem for mixture models. Nonetheless, it is readily solved. Guided by Bayes rule, we will uncover a simple recursion that neatly computes the desired posterior probabilities. In fact, this algorithm marks an important milestone for us—it begins to suggest the general machinery for propagating probabilities on graphs that we will be our focus in much of the remainder of the book. With HMMs we begin our study of inference in graphical models in earnest.

12.1 The graphical model

The graphical model representation of an HMM is shown in Figure 12.2. As the diagram makes clear, the HMM can be viewed as a linked sequence of mixture models, with the linking occurring at the level of the mixture components, or “states.” We denote the state at time t as q_t , and let y_t represent the observable “output” at time t .¹

¹Throughout the chapter we refer to t as a temporal variable for concreteness; the HMM model is of course applicable to any kind of sequential data.

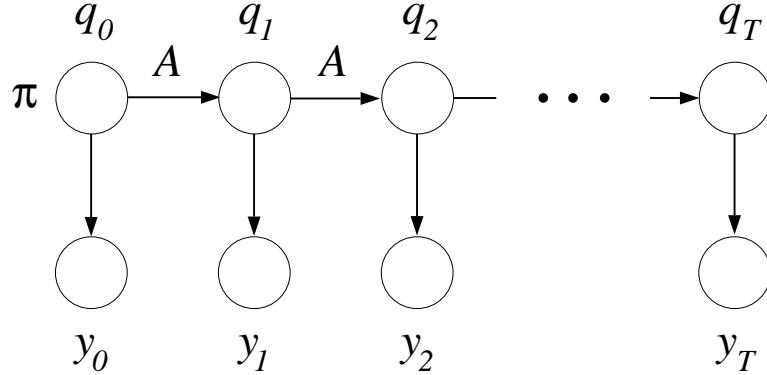


Figure 12.2: The representation of a HMM as a graphical model. Each vertical slice represents a time step. The top node in each slice represents the multinomial q_t variable and the bottom node in each slice represents the observable y_t variable.

We represent the state at time t as a multinomial random variable q_t , with components q_t^i , for $i = 0, \dots, M$. Thus q_t^i is equal to one for a particular value of i and is equal to zero for $j \neq i$. As for the output variables y_t , these variables are always observed in the HMM setting and thus they play a minimal role in the inference problem. We will accordingly leave their type undefined for now (the reader can think of them as multinomial or multivariate Gaussian for concreteness).

From the graphical model we can read off various conditional independencies. The main conditional independency of interest is that obtained by conditioning on a single state node. Conditioning on q_t renders q_{t-1} and q_{t+1} independent; moreover it renders q_s independent of q_u , for $s < t$ and $t < u$. Thus, “the future is independent of the past, given the present.” This statement is also true for output nodes y_s and y_u , again conditioning on the state node q_t .

Note that conditioning on an output node, on the other hand, does not separate nodes in the graph and thus does not yield any conditional independencies. It is not true that the future is independent of the past, given the present, if by “present” we mean the current output.

Indeed, conditioning on *all* of the output nodes fails to separate any of the remaining nodes. That is, given the observable data, we cannot expect any independencies to be induced between the state nodes. Thus we should expect that our inference algorithm must take into account possible dependencies between states at arbitrary locations along the chain. In particular, learning something about the final state node in the chain, q_T (e.g., by observing y_T), can change the posterior probability distribution for the first node in the chain, q_0 . We expect that our inference algorithm will have to propagate information from one end of the chain to the other.

12.2 The parameterization

We now parameterize the HMM by assigning local conditional probabilities to each of the nodes. The first state node in the sequence has no parents; thus we endow this node with an unconditional distribution π , where $\pi^i \triangleq p(q_0^i = 1)$. Each successive state node has the previous state node in the

chain as its (sole) parent; thus we need a $M \times M$ matrix to specify its local conditional probability. We define a *state transition matrix* A , where the (i, j) th entry a_{ij} of A is defined to be the transition probability $p(q_{t+1}^j = 1 | q_t^i = 1)$. Note that we assume that this transition probability is independent of t ; that is, we assume a *homogeneous* HMM. (All of the algorithms that we describe are readily generalized to the case of a varying A matrix, however this case is less common in practice than the homogeneous case).

Each of the output nodes has a single state node as a parent, thus we require a probability distribution $p(y_t | q_t)$. We again assume this distribution to be independent of t . We make no further assumptions regarding the form of $p(y_t | q_t)$ for now; for the purposes of developing the HMM inference algorithms we need only be able to evaluate $p(y_t | q_t)$ for a fixed value of y_t .

The joint probability is obtained as always by taking the product over the local conditional probabilities. Thus, for a particular configuration $(q, y) = (q_0, q_1, \dots, q_T, y_0, y_1, \dots, y_T)$, we obtain the following joint probability:

$$p(q, y) = p(q_0) \prod_{t=0}^{T-1} p(q_{t+1} | q_t) \prod_{t=0}^T p(y_t | q_t). \quad (12.1)$$

To introduce the A and π parameters into this equation, we adopt a notation in which state variables can be used as indices. Thus, when q_t takes on its i th value and q_{t+1} takes on its j th value, we let $a_{q_t, q_{t+1}}$ denote the (i, j) th entry of the matrix A . Formally, this interpretation is achieved via the following definition:

$$a_{q_t, q_{t+1}} \triangleq \prod_{i,j=1}^M [a_{ij}]^{q_t^i q_{t+1}^j}. \quad (12.2)$$

Recall that only one of the components of q_t is one, and thus only one factor in the product on the right-hand side is different from one; this picks out the appropriate entry in the matrix A . Similarly, we define π_{q_0} via:

$$\pi_{q_0} \triangleq \prod_{i=1}^M [\pi_i]^{q_0^i} \quad (12.3)$$

which has the effect of picking out the appropriate entry in the π vector. We use the simple shorthand forms $a_{q_t, q_{t+1}}$ and π_{q_0} throughout the chapter, although the expanded forms in Eqs. 12.2 and 12.3 will also prove useful when we discuss parameter estimation.

Plugging the definitions into the joint probability, we have:

$$p(q, y) = \pi_{q_0} \prod_{t=0}^{T-1} a_{q_t, q_{t+1}} \prod_{t=0}^T p(y_t | q_t). \quad (12.4)$$

This is the parameterized probability distribution in which we wish to do inference.

12.3 The inference problem

There are quite a number of inference problems that are of interest in the setting of the HMM. The general inference problem involves computing the probability of a hidden state sequence q given an

observable output sequence y . Various marginal probabilities are also of interest, in particular the probability of a particular hidden state q_t given the output sequence.

It is also of interest to compute various probabilities conditioned on partial output sequences. In particular, consider the “on-line” problem in which a sequence of outputs y_t arrives and it is desired to compute the probability of the state at time t immediately, without waiting for future data. Computing this probability, $p(q_t|y_0, \dots, y_t)$, is generally called the *filtering problem*.² Another inference problem involves the calculation of $p(q_t|y_0, \dots, y_s)$, where $t > s$. This is referred to as the *prediction problem*. Finally, the problem of calculating a posterior probability based on data up to and including a future time, i.e., $p(q_t|y_0, \dots, y_u)$ for $t < u$, is referred to as the *smoothing problem*.

Let us consider the problem of computing the posterior probability $p(q|y)$ where $y = (y_0, \dots, y_T)$ is the entire observed output sequence at our disposal. Let q be an arbitrary fixed state sequence whose probability we wish to compute. By definition we have $p(q|y) = p(q, y)/p(y)$. The numerator is readily calculated by substituting q and y in Eq. 12.4. What about the denominator $p(y)$?

Calculating the denominator involves taking a sum across all possible values of the hidden states:

$$p(y) = \sum_{q_0} \sum_{q_1} \cdots \sum_{q_T} \pi(q_0) \prod_{t=0}^{T-1} a_{q_t, q_{t+1}} \prod_{t=0}^T p(y_t|q_t, \eta). \quad (12.5)$$

This sum should give us pause. Each state node q_t can take on M values, and we have T state nodes. This implies that we must perform M^T sums, a wildly intractable number for reasonable values of M and T . Is it possible to perform inference efficiently for HMMs?

The way out of our seeming dilemma lies in the factorized form of the joint probability distribution (Eq. 12.4). Each factor involves only one or two of the state variables, and the factors form a neatly organized chain. This suggests that it ought to be possible to move these sums “inside” the product in a systematic way. Moving the sums as far as possible ought to reduce the computational burden significantly. Consider, for example, the sum over q_T . This sum can be brought inside until the end of the chain and applied to the two factors involving q_T . Once this sum is performed the result can be combined with the two factors involving q_{T-1} and the sum over q_{T-1} can be performed. We begin to hope that we can organize our calculation as a recursion.

12.4 Inference

To reveal the recursion behind the HMM inference problem as simply as possible, let us consider an inference problem that is seemingly easier than the full problem. Rather than calculating $p(q|y)$ for the entire state sequence q , we focus on a particular state node q_t and ask to calculate its posterior probability, that is, we calculate $p(q_t|y)$. This posterior probability also has $p(y)$ in its denominator,

²Why “filtering”? The terminology arises from the interpretation of the outputs y_t as providing “noisy” information about the underlying “signal” q_t . The inference problem is then one of “filtering” the noise from the signal. In the linear stochastic systems setting in which this terminology originally arose (cf. Chapter 15), the calculation of quantities such as $p(q_t|y_0, \dots, y_t)$ often had a frequency domain interpretation in which some frequencies are passed and not others. In such a setting the terminology is rather natural. While recognizing the possible unnaturalness of the terminology outside of the linear systems setting, we bow to its wide usage and adopt it here.

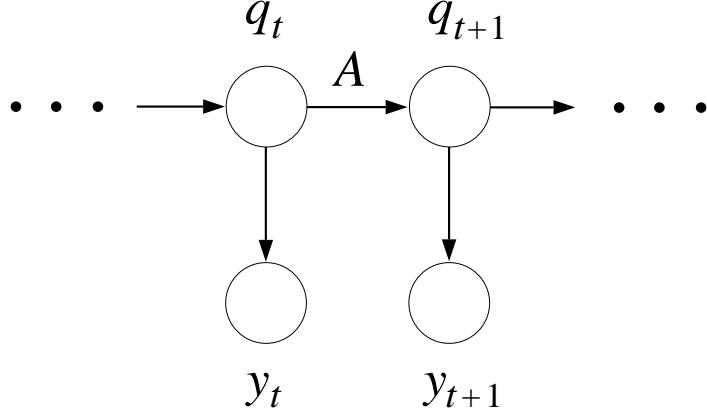


Figure 12.3: A fragment of the graphical model representation of an HMM.

and in fact we can easily adapt our algorithm for computing $p(q_t|y)$ to compute $p(q|y)$ or marginals over substrings of q (see Exercise ??).

We thus turn to the calculation of $p(q_t|y)$. To make progress, we need to take advantage of the conditional independencies in our graphical model, breaking the problem into pieces. To do so we condition on a state node (see Figure 12.3). We reverse the terms q_t and y via an application of Bayes rule, conditioning now on q_t , and use conditional independence:

$$p(q_t|y) = \frac{p(y|q_t)p(q_t)}{p(y)} \quad (12.6)$$

$$= \frac{p(y_0, \dots, y_t|q_t)p(y_{t+1}, \dots, y_T|q_t)p(q_t)}{p(y)}. \quad (12.7)$$

Finally, we regroup the terms and make a definition:

$$p(q_t|y) = \frac{p(y_0, \dots, y_t, q_t)p(y_{t+1}, \dots, y_T|q_t)}{p(y)} \quad (12.8)$$

$$= \frac{\alpha(q_t)\beta(q_t)}{p(y)}, \quad (12.9)$$

where

$$\alpha(q_t) \triangleq p(y_0, \dots, y_t, q_t) \quad (12.10)$$

is the probability of emitting a partial sequence of outputs y_0, \dots, y_t and ending up in state q_t , and

$$\beta(q_t) = p(y_{t+1}, \dots, y_T|q_t) \quad (12.11)$$

is the probability of emitting a partial sequence of outputs y_{t+1}, \dots, y_T given that the system starts in state q_t .

Given that the sum of $p(q_t|y)$ over the possible values of q_t must equal one, we use Eq. 12.9 to obtain:

$$p(y) = \sum_{q_t} \alpha(q_t)\beta(q_t). \quad (12.12)$$

That is, we can obtain the likelihood $p(y)$ by calculating $\alpha(q_t)$ and $\beta(q_t)$ for any t and summing their product.

We make one additional definition: $\gamma(q_t)$ will denote the posterior probability $p(q_t|y)$. Thus:

$$\gamma(q_t) \triangleq \frac{\alpha(q_t)\beta(q_t)}{p(y)}, \quad (12.13)$$

where $p(y)$ is computed once, as the normalization constant for a particular (arbitrary) choice of t .

We have reduced our problem to that of calculating the alphas and the betas. This is a useful reduction because, as we now see, these quantities can be computed recursively.

Let us first consider the alpha variables. Given that $\alpha(q_t)$ depends only on quantities up to time t , and given the Markov properties of our model, we might hope to obtain a recursion between $\alpha(q_t)$ and $\alpha(q_{t+1})$. Indeed, referring to Figure 12.3 to justify the conditional independencies we need, we obtain:

$$\alpha(q_{t+1}) = p(y_0, \dots, y_{t+1}, q_{t+1}) \quad (12.14)$$

$$= p(y_0, \dots, y_{t+1}|q_{t+1})p(q_{t+1}) \quad (12.15)$$

$$= p(y_0, \dots, y_t|q_{t+1})p(y_{t+1}|q_{t+1})p(q_{t+1}) \quad (12.16)$$

$$= p(y_0, \dots, y_t, q_{t+1})p(y_{t+1}|q_{t+1}) \quad (12.17)$$

$$= \sum_{q_t} p(y_0, \dots, y_t, q_t, q_{t+1})p(y_{t+1}|q_{t+1}) \quad (12.18)$$

$$= \sum_{q_t} p(y_0, \dots, y_t, q_{t+1}|q_t)p(q_t)p(y_{t+1}|q_{t+1}) \quad (12.19)$$

$$= \sum_{q_t} p(y_0, \dots, y_t|q_t)p(q_{t+1}|q_t)p(q_t)p(y_{t+1}|q_{t+1}) \quad (12.20)$$

$$= \sum_{q_t} p(y_0, \dots, y_t, q_t)p(q_{t+1}|q_t)p(y_{t+1}|q_{t+1}) \quad (12.21)$$

$$= \sum_{q_t} \alpha(q_t)a_{q_t, q_{t+1}}p(y_{t+1}|q_{t+1}). \quad (12.22)$$

Throughout this derivation the key idea is to condition on a state and then use the conditional independence properties of the model to decompose the equation. This is done in Eqs. 12.16 and 12.22, both of which can be verified via the graphical model fragment. The second key idea is to introduce a variable, in this case q_t , by marginalizing over it (cf. Eq. 12.18). Once q_t is introduced the recursion follows readily.

The computational complexity of each step of the alpha recursion is $O(M^2)$; in particular, for each of the M values of q_{t+1} , we require M multiplications to compute the inner product of $\alpha(q_t)$

with the appropriate column of the A matrix. To compute all of the alpha variables from $t = 1$ to $t = T$ thus requires time $O(M^2T)$.

Note that the algorithm proceeds “forward” in time. The definition of alpha at the first time step yields:

$$\alpha(q_0) = p(y_0, q_0) \quad (12.23)$$

$$= p(y_0|q_0)p(q_0) \quad (12.24)$$

$$= p(y_0|q_0)\pi_{q_0}. \quad (12.25)$$

and these values are used to initialize the recursion.

For the beta variables we obtain a “backward” recursion in which $\beta(q_t)$ is expressed in terms of $\beta(q_{t+1})$, where once again the various steps are justified by making reference to the graphical model fragment in Figure 12.3:

$$\beta(q_t) = p(y_{t+1}, \dots, y_T|q_t) \quad (12.26)$$

$$= \sum_{q_{t+1}} p(y_{t+1}, \dots, y_T, q_{t+1}|q_t) \quad (12.27)$$

$$= \sum_{q_{t+1}} p(y_{t+1}, \dots, y_T|q_{t+1}, q_t)p(q_{t+1}|q_t) \quad (12.28)$$

$$= \sum_{q_{t+1}} p(y_{t+2}, \dots, y_T|q_{t+1})p(y_{t+1}|q_{t+1})p(q_{t+1}|q_t) \quad (12.29)$$

$$= \sum_{q_{t+1}} \beta(q_{t+1})a_{q_t, q_{t+1}}p(y_{t+1}|q_{t+1}). \quad (12.30)$$

Note that the beta recursion is a backwards recursion; that is, we start at the final time step T and proceed backwards to the initial time step.

As for the initialization of the beta recursion, the definition of $\beta(q_T)$ is unhelpful, given that it makes reference to a non-existent y_{T+1} , but we see from applying the recursion once to compute $\beta(q_{T-1})$ that this value will be calculated correctly if we define $\beta(q_T)$ to be a vector of ones. Alternatively, computing $p(y)$ at time T , we have:

$$p(y) = \sum_i \alpha(q_T^i)\beta(q_T^i) \quad (12.31)$$

$$= \sum_i \alpha(q_T^i) \quad (12.32)$$

$$= \sum_i p(y_0, \dots, y_T, q_T^i) \quad (12.33)$$

$$= p(y), \quad (12.34)$$

and we see that the definition makes sense.

If we need only the likelihood $p(y)$, Eq. 12.31 shows us that it is not necessary to compute the betas; a single forward pass for the alphas will suffice. Moreover, Eq. 12.12 tell us that any partial

forward pass up to time t to compute $\alpha(q_t)$, accompanied by a partial backward pass to compute $\beta(q_t)$, will also suffice. To compute the posterior probabilities for all of the states q_t , however, requires us to compute alphas and betas for each time step. Thus we require a forward pass and a backward pass for a complete solution to the inference problem.

12.5 An alternative inference algorithm

The alpha-beta algorithm is not the only way to compute the posterior probabilities of the states. In this section we describe an alternative approach in which the backward phase is a recursion defined directly on the $\gamma(q_t)$ variables. An interesting feature of this algorithm is that the backward phase makes no use of the observations y_t ; only the forward phase uses the observed data. We can throw away the data as we filter.

The algorithm differs from the alpha-beta algorithm only in the backward phase. In the forward direction we run the alpha algorithm as before, calculating the filtered quantities $\alpha(q_t) = p(y_0, \dots, y_t, q_t)$.

To uncover a backward recursion linking the γ_t variables, we refer once again to the graphical model fragment in Figure 12.3. Our goal is to compute $\gamma(q_t) = p(q_t | y_0, \dots, y_T)$. As in our earlier calculations, our main tool for computing such quantities recursively is to condition on a state variable; such conditioning breaks the problem into two pieces. In particular, we condition on q_{t+1} and obtain:

$$p(q_t | q_{t+1}, y_0, \dots, y_T) = p(q_t | q_{t+1}, y_0, \dots, y_t). \quad (12.35)$$

This shows that we can get a conditional probability that depends on all of the data via a conditional probability that depends only on the partial sequence up to t . Moreover, the left-hand side can be readily converted into $\gamma(q_t)$ by multiplying by $p(q_{t+1} | y_0, \dots, y_T)$ —which is $\gamma(q_{t+1})$ by definition—and summing over q_{t+1} . The details are as follows:

$$\gamma(q_t) = \sum_{q_{t+1}} p(q_t, q_{t+1} | y_0, \dots, y_T) \quad (12.36)$$

$$= \sum_{q_{t+1}} p(q_t | q_{t+1}, y_0, \dots, y_T) p(q_{t+1} | y_0, \dots, y_T) \quad (12.37)$$

$$= \sum_{q_{t+1}} p(q_t | q_{t+1}, y_0, \dots, y_t) p(q_{t+1} | y_0, \dots, y_T) \quad (12.38)$$

$$= \sum_{q_{t+1}} \frac{p(q_t, q_{t+1}, y_0, \dots, y_t)}{\sum_{q_t} p(q_t, q_{t+1}, y_0, \dots, y_t)} p(q_{t+1} | y_0, \dots, y_T) \quad (12.39)$$

$$= \sum_{q_{t+1}} \frac{p(q_t, y_0, \dots, y_t) p(q_{t+1} | q_t)}{\sum_{q_t} p(q_t, y_0, \dots, y_t) p(q_{t+1} | q_t)} p(q_{t+1} | y_0, \dots, y_T) \quad (12.40)$$

$$= \sum_{q_{t+1}} \frac{\alpha(q_t) a_{q_t, q_{t+1}}}{\sum_{q_t} \alpha(q_t) a_{q_t, q_{t+1}}} \gamma(q_{t+1}) \quad (12.41)$$

We see that this recursion makes use of the alpha variables, which therefore must be computed before the gamma recursion begins. The gamma recursion is initialized with $\gamma(q_T) = \alpha(q_T)$.

Note that the data y_t are not referenced in the gamma recursion; the alpha recursion has absorbed all of the necessary data likelihoods.

12.6 The $\xi(q_t, q_{t+1})$ variables

The alpha-beta or the alpha-gamma algorithm provide us with the posterior probability of the hidden states of the HMM. These quantities are the direct analogs of the posterior probabilities h_i that we studied in the simpler mixture setting. Moreover, they play the same role in estimating the parameters of the output distribution—as we will see in Section 12.8 they are the expected sufficient statistics for these parameters. To estimate the transition probability matrix A , however, we need something more. It is clear intuitively, and justified in Section 12.8, where we write out the complete log likelihood, that what is required is the matrix of cooccurrence probabilities $p(q_t, q_{t+1}|y)$. In this section we show how to calculate these posterior probabilities.

Let us define

$$\xi(q_t, q_{t+1}) \triangleq p(q_t, q_{t+1}|y). \quad (12.42)$$

There are several ways to calculate this quantity. One way is to return to first principles and develop recursions for the $\xi(q_t, q_{t+1})$, following much the same procedure as we followed for the singleton probabilities $\gamma(q_t)$. This is indeed a rather useful exercise (which we ask the reader to carry out in Exercise ??), not only because it reinforces the Markovian calculations that we have engaged in, but because it provides a stepping-stone to the general “junction tree algorithm” that we discuss in Chapter 17. That algorithm provides a general framework from which to derive all of the recursions that we describe in this chapter. Moreover, if we have an algorithm for calculating $\xi(q_t, q_{t+1})$ in hand, we can also obtain the singleton probabilities via $\gamma(q_t) = \sum_{q_{t+1}} \xi(q_t, q_{t+1})$.

A second approach to calculating $\xi(q_t, q_{t+1})$ is to build on the recursions already developed for the alphas and betas:

$$\xi(q_t, q_{t+1}) = p(q_t, q_{t+1}|y) \quad (12.43)$$

$$\begin{aligned} &= \frac{p(y|q_t, q_{t+1})p(q_{t+1}|q_t)p(q_t)}{p(y)} \\ &= \frac{p(y_0, \dots, y_t|q_t)p(y_{t+1}|q_{t+1})p(y_{t+2}, \dots, y_T|q_{t+1})p(q_{t+1}|q_t)p(q_t)}{p(y)} \\ &= \frac{\alpha(q_t)p(y_{t+1}|q_{t+1})\beta(q_{t+1})a_{q_t, q_{t+1}}}{p(y)}. \end{aligned} \quad (12.44)$$

This result can also be expressed in terms of alphas and gammas:

$$\xi(q_t, q_{t+1}) = \frac{\alpha(q_t)p(y_{t+1}|q_{t+1})\gamma(q_{t+1})a_{q_t, q_{t+1}}}{\alpha(q_{t+1})}. \quad (12.45)$$

In either case we see that we can readily calculate the $\xi(q_t, q_{t+1})$ variables once we have finished the recursive calculation of the singleton probabilities $\gamma(q_t)$.

12.7 Numerical issues

To summarize, we have found that we can calculate all of the necessary posterior probabilities for the HMM recursively. Given an observed sequence y , we run the alpha recursion forward in time. If we require only the likelihood we simply sum the alphas at the final time step. If we also require the posterior probabilities of the states, we proceed to either the beta recursion or the gamma recursion.

Before these recursions are implemented on the computer, attention must be paid to numerical issues. In particular, the recursions involve repeated multiplications of small numbers and it is generally not long before the numbers underflow. To avoid underflow it suffices to normalize. We outline the basic ideas in Exercise ??, but in brief the procedure is as follows. The alpha variables, $p(y_0, \dots, y_t, q_t)$, can be viewed as unnormalized conditional probabilities. Indeed, normalizing means division by $p(y_0, \dots, y_t)$, which yields conditionals $p(q_t|y_0, \dots, y_t)$. Not only are these conditionals scaled in a numerically sensible manner, but they also have a sensible semantics—they are the filtered estimates of the states. In sum, one should always compute normalized alphas. In the backward direction, if one uses the gamma recursion one is already on safe ground—the gammas are conditional probabilities and hence sum to one. Moreover, it is easy to verify that normalized alphas can be used in Eq. 12.41 in place of the unnormalized alphas. Alternatively, if one uses the beta recursion, it turns out that a numerically sensible solution (although one that is devoid of probabilistic interpretation), is to use the normalization factors from the forward recursion to rescale the beta variables (these rescaled betas will not sum to one). It turns out that the rescaled variables are then used exactly as the original alphas and betas are used in the formulas for the posteriors $\gamma(q_t)$ and $\xi(q_t, q_{t+1})$ (i.e., the normalization factors cancel). See Exercise ?? for further discussion.

12.8 Parameter estimation

The parameters of an HMM are the transition matrix A , the initial probability distribution π and the parameters that are associated with the output probability distribution. In this section we discuss the problem of estimating these parameters from data.

Let $\theta = (\pi, A, \eta)$ represent all of the parameters of the HMM model, where $p(y_t|q_t, \eta)$ is the output distribution. The likelihood is given by $p(y|\theta)$, for a fixed observable sequence y . Taking the logarithm of Eq. 12.5 we have the following log likelihood:

$$p(y|\theta) = \log \sum_{q_0} \sum_{q_1} \cdots \sum_{q_T} \pi(q_0) \prod_{t=0}^{T-1} a_{q_t, q_{t+1}} \prod_{t=0}^T p(y_t|q_t, \eta). \quad (12.46)$$

Our goal is to maximize this expression with respect to θ .

As with our earlier models, this is in principle just another optimization problem that can be solved via standard numerical optimization methods. In practice, however, the EM algorithm is generally used to estimate HMM parameters.

12.8.1 EM algorithm

The EM algorithm for the HMM presents no new difficulties to surmount and we will make relatively short work of the derivation. For concreteness we derive the algorithm for the case in which the outputs y_t are multinomial variables; it should be obvious how to change the derivation to accommodate other output types (cf. Exercise ??).

In the multinomial case, y_t is an N -component vector such that y_t^j is equal to one for a particular component and zero for all other components. We use the symbol η_{ij} to denote the probability that the j th component of y_t is one, given that the i th component of q_t is one; i.e., $\eta_{ij} \triangleq p(y_t^j = 1 | q_t^i = 1, \eta)$. Using this notation we have:

$$p(y_t | q_t, \eta) = \prod_{i,j=1}^M [\eta_{ij}]^{q_t^i y_t^j} \quad (12.47)$$

as the general expression for the output distribution.

As usual we begin by writing down the complete log likelihood to discover the form of the M step estimates as well as the sufficient statistics that are needed for the E step. We have:

$$\log p(q, y) = \log \left\{ \pi_{q_0} \prod_{t=0}^{T-1} a_{q_t, q_{t+1}} \prod_{t=0}^T p(y_t | q_t, \eta) \right\} \quad (12.48)$$

$$= \log \left\{ \prod_{i=1}^M [\pi_i]^{q_0^i} \prod_{t=0}^{T-1} \prod_{i,j=1}^M [a_{ij}]^{q_t^i q_{t+1}^j} \prod_{t=0}^T \prod_{i,j=1}^M [\eta_{ij}]^{q_t^i y_t^j} \right\} \quad (12.49)$$

$$= \sum_{i=1}^M q_0^i \log \pi_i + \sum_{t=0}^{T-1} \sum_{i,j=1}^M q_t^i q_{t+1}^j \log a_{ij} + \sum_{t=0}^T \sum_{i,j=1}^M q_t^i y_t^j \log \eta_{ij}. \quad (12.50)$$

From this expression, we see that $m_{ij} \triangleq \sum_{t=0}^{T-1} q_t^i q_{t+1}^j$ is the sufficient statistic for a_{ij} , $n_{ij} \triangleq \sum_{t=0}^T q_t^i y_t^j$ is the sufficient statistic for η_{ij} , and q_0^i is the sufficient statistic for π_i . The maximum likelihood estimates for the case of complete data are therefore given by:

$$\hat{a}_{ij} = \frac{m_{ij}}{\sum_{k=1}^M m_{ik}} \quad (12.51)$$

$$\hat{\eta}_{ij} = \frac{n_{ij}}{\sum_{k=1}^N n_{ik}} \quad (12.52)$$

$$\hat{\pi}_i = q_0^i. \quad (12.53)$$

All of these estimates have natural interpretations. Note that m_{ij} is the count of the number of times that the process is in state i and transitions to state j . Dividing by $\sum_k m_{ik}$ yields the proportion of those transitions out of state i that go to state j —a natural estimate of a_{ij} . Similarly the estimate of η_{ij} is given by the proportion of times that the chain is in state i and produces the

j th output value. Finally, for the estimate of π_i , we obtain a singular distribution that puts all of the probability mass at the observed initial state.³

We turn to the E step of the EM algorithm. Consider first the expectation of the sufficient statistic $n_{ij} = \sum_{t=0}^T q_t^i y_t^j$. We have:

$$E(n_{ij}|y, \theta^{(p)}) = \sum_{t=0}^T E(q_t^i|y, \theta^{(p)}) y_t^j \quad (12.54)$$

$$= \sum_{t=0}^T p(q_t^i = 1|y, \theta^{(p)}) y_t^j \quad (12.55)$$

$$\triangleq \sum_{t=0}^T \gamma_t^i y_t^j, \quad (12.56)$$

where we introduce the notation γ_t^i in the last line. By definition γ_t^i is equal to $\gamma(q_t)$, evaluated at that value of q_t such that $q_t^i = 1$. Note finally that the dependence of γ_t^i on $\theta^{(p)}$ has been suppressed.

Similarly, for the sufficient statistic m_{ij} , we have:

$$E(m_{ij}|y, \theta^{(p)}) = \sum_{t=0}^{T-1} E(q_t^i q_{t+1}^j|y, \theta^{(p)}) \quad (12.57)$$

$$= \sum_{t=0}^{T-1} p(q_t^i q_{t+1}^j|y, \theta^{(p)}) \quad (12.58)$$

$$\triangleq \sum_{t=0}^{T-1} \xi_{t,t+1}^{ij}, \quad (12.59)$$

where we let $\xi_{t,t+1}^{ij}$ denote $\xi(q_t, q_{t+1})$ for (q_t, q_{t+1}) such that $q_t^i = 1$ and $q_{t+1}^j = 1$.

In summary, the sufficient statistics are calculated via the recursive forward-backward procedure from Section 12.4. We calculate the γ variables via either Eq. 12.13 or Eq. 12.41. The ξ variables are then calculated via Eq. 12.44 or Eq. 12.45.

With the estimated sufficient statistics in hand, we substitute into the maximum likelihood formulas (Eqs. 12.51, 12.52, and 12.53), to obtain the M step of the EM algorithm (also known, in the case of HMMs, as the ‘‘Baum-Welch updates’’). We obtain:

$$\hat{\eta}_{ij}^{(p+1)} = \frac{\sum_{t=0}^T \gamma_t^i y_t^j}{\sum_{k=1}^N \sum_{t=0}^T \gamma_t^i y_t^k} = \frac{\sum_{t=0}^T \gamma_t^i y_t^j}{\sum_{t=0}^T \gamma_t^i}. \quad (12.60)$$

³In a more general setting, in which we have multiple repeated observations from a single HMM (i.e., data that are IID at the level of entire sequences), the estimate of π_i becomes the proportion of times that the chain starts in state i , and indeed the estimates in Eq. 12.51 and Eq. 12.52 also become averages over the multiple repetitions (cf. Exercise ??).

where we use the fact that $\sum_{k=1}^N y_t^k = 1$,

$$\hat{a}_{ij}^{(p+1)} = \frac{\sum_{t=0}^{T-1} \xi_{t,t+1}^{i,j}}{\sum_{k=1}^M \sum_{t=0}^{T-1} \xi_{t,t+1}^{i,k}} = \frac{\sum_{t=0}^{T-1} \xi_{t,t+1}^{i,j}}{\sum_{t=0}^{T-1} \gamma_t^i}, \quad (12.61)$$

where by definition $\sum_{k=1}^M \xi_{t,t+1}^{i,k} = \gamma_t^i$, and:

$$\hat{\pi}_i^{(p+1)} = \gamma_0^i. \quad (12.62)$$

The EM algorithm iterates between performing these updates (the M step) and the forward-backward pass using the updated values (the E step).

12.9 Historical remarks and bibliography

Chapter 13

The Multivariate Gaussian

In this chapter we present some basic facts regarding the multivariate Gaussian distribution. We discuss the two major parameterizations of the multivariate Gaussian—the *moment parameterization* and the *canonical parameterization*, and we show how the basic operations of marginalization and conditioning are carried out in these two parameterizations. We also discuss maximum likelihood estimation for the multivariate Gaussian.

13.1 Parameterizations

The multivariate Gaussian distribution is commonly expressed in terms of the parameters μ and Σ , where μ is an $n \times 1$ vector and Σ is an $n \times n$, symmetric matrix. (We will assume for now that Σ is also positive definite, but later on we will have occasion to relax that constraint). We have the following form for the density function:

$$p(x|\mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp \left\{ -\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu) \right\}, \quad (13.1)$$

where x is a vector in \Re^n . The density can be integrated over volumes in \Re^n to assign probability mass to those volumes.

The geometry of the multivariate Gaussian is essentially that associated with the quadratic form $f(x) = \frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)$ in the exponent of the density. Recall our discussion in Chapter 6, where we showed that a quadratic form $f(x)$ is a paraboloid with level surfaces, i.e., surfaces of the form $f(x) = c$ for fixed c , being ellipsoids oriented along the eigenvectors of the matrix Σ . Now note that the exponential, $\exp(\cdot)$, is a scalar function that leaves the geometrical features of the quadratic form intact. That is, for any x lying on an ellipsoid $f(x) = c$, we obtain the value $\exp\{-c\}$. The maximum value of the exponential is 1, obtained at $x = \mu$ where $f(x) = 0$. The paraboloid $f(x)$ increases to infinity as we move away from $x = \mu$; thus we obtain a “bump” in $(n+1)$ -dimensional space centered at $x = \mu$. The level surfaces of the Gaussian bump are ellipsoids oriented along the eigenvectors of Σ .

The factor in front of the exponential in Eq. 13.1 is the normalization factor that ensures that the density integrates to one. To show that this factor is correct, we make use of the diagonalization

of Σ^{-1} . Diagonalization yields a product of n univariate Gaussians whose standard deviations are the eigenvalues of Σ . When we integrate, each of these univariate Gaussians contributes a factor $\sqrt{2\pi}\lambda_i$ to the normalization, where λ_i is the i th eigenvalue of Σ . Recall that the determinant of a matrix is the product of its eigenvalues to obtain the result. (We ask the reader to fill in the details of this derivation in Exercise ??).

As in the univariate case, the parameters μ and Σ have a probabilistic interpretation as the *moments* of the Gaussian distribution. In particular, we have the important result:

$$\mu = E(x) \tag{13.2}$$

$$\Sigma = E(x - \mu)(x - \mu)^T. \tag{13.3}$$

We will not bother to derive this standard result, but will provide a hint: diagonalize and appeal to the univariate case.

Although the moment parameterization of the Gaussian will play a principal role in our subsequent development, there is a second parameterization—the canonical parameterization—that will also be important. In particular, expanding the quadratic form in Eq. 13.1, and defining *canonical parameters*:

$$\Lambda = \Sigma^{-1} \tag{13.4}$$

$$\eta = \Sigma^{-1}\mu, \tag{13.5}$$

we obtain:

$$p(x|\eta, \Lambda) = \exp \left\{ a + \eta^T x - \frac{1}{2} x^T \Lambda x \right\}, \tag{13.6}$$

where $a = -1/2(n \log(2\pi) - \log |\Lambda| + \eta^T \Lambda \eta)$ is the normalizing constant in this representation. The canonical parameterization is also sometimes referred to as the *information parameterization*.

We can also convert from canonical parameters to moment parameters:

$$\mu = \Lambda^{-1}\eta \tag{13.7}$$

$$\Sigma = \Lambda^{-1}. \tag{13.8}$$

Moment parameters and canonical parameters are useful in different circumstances. As we will see, different kinds of transformations are more readily carried out in one representation or the other.

13.2 Joint distributions

Suppose that we partition the $n \times 1$ vector x into a $p \times 1$ subvector x_1 and a $q \times 1$ subvector x_2 , where $n = p + q$. Form corresponding partitions of the μ and Σ parameters:

$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \quad \Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}, \tag{13.9}$$

We can write a joint Gaussian distribution for x_1 and x_2 using these partitioned parameters:

$$p(x|\mu, \Sigma) = \frac{1}{(2\pi)^{(p+q)/2} |\Sigma|^{1/2}} \exp \left\{ -\frac{1}{2} \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{pmatrix}^T \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}^{-1} \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{pmatrix} \right\} \quad (13.10)$$

This partitioned form of the joint distribution raises a number of questions. In particular, we can equally well form partitioned versions of η and Λ and express the joint distribution in the canonical parameterization; is there any relationship between the partitioned form of these two representations? Also, what do the blocks in the partitioned forms have to do with the marginal and conditional probabilities of x_1 and x_2 ?

These questions all involve the manipulation of the quadratic forms in the exponents of the Gaussian densities; indeed, the underlying algebraic problem is that of “completing the square” of quadratic forms. In the next section, we discuss an algebra that provides a general solution to the problem of “completing the square.”

13.3 Partitioned matrices

Our first result in this section is to show how to block diagonalize a partitioned matrix. A number of useful results flow from this operation, including an explicit expression for the inverse of a partitioned matrix.

Consider a general partitioned matrix:

$$M = \begin{bmatrix} E & F \\ G & H \end{bmatrix}, \quad (13.11)$$

where we assume that both E and H are invertible. (Our results can be generalized beyond this setting). To invert this matrix, we follow a similar procedure to that of diagonalization. In particular, we wish to *block diagonalize* the matrix. We wish to put a block of zeros in place of G and a block of zeros in place of F .

To zero out the upper-right-hand corner of M , note that it suffices to *premultiply* the second block column of M by a “block row vector” having elements I and $-FH^{-1}$. Similarly, to zero out the lower-left-hand corner of M , it suffices to *postmultiply* the second row of M by a “block column vector” having elements I and $-H^{-1}G$. The magical fact is that these two operations do not interfere with each other; thus we can block diagonalize M by doing both operations. In particular, we have:

$$\begin{bmatrix} I & -FH^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} \begin{bmatrix} I & 0 \\ -H^{-1}G & I \end{bmatrix} = \begin{bmatrix} E - FH^{-1}G & 0 \\ 0 & H \end{bmatrix}. \quad (13.12)$$

The correctness of this decomposition can be verified directly.

We define the *Schur complement* of the matrix M with respect to H , denoted M/H , as the term $E - FH^{-1}G$ that appears in the block diagonal matrix. It is not difficult to show that M/H is invertible.

We now take the inverse of both sides of Eq. 13.12. Note that in a matrix expression of the form $XYZ = W$ inverting both sides yields $Y^{-1} = ZW^{-1}X$; this implies that we don't need to explicitly invert the block triangular matrices in Eq. 13.12 (although this is easily done). Note also that the inverse of a block diagonal matrix is the diagonal matrix of the inverse of its blocks. Thus we have:

$$\begin{bmatrix} E & F \\ G & H \end{bmatrix}^{-1} = \begin{bmatrix} I & 0 \\ -H^{-1}G & I \end{bmatrix} \begin{bmatrix} (M/H)^{-1} & 0 \\ 0 & H^{-1} \end{bmatrix} \begin{bmatrix} I & -FH^{-1} \\ 0 & I \end{bmatrix} \quad (13.13)$$

$$= \begin{bmatrix} (M/H)^{-1} & -(M/H)^{-1}FH^{-1} \\ -H^{-1}G(M/H)^{-1} & H^{-1} + H^{-1}G(M/H)^{-1}FH^{-1} \end{bmatrix}, \quad (13.14)$$

which expresses the inverse of a partitioned matrix in terms of its blocks.

We can also apply the determinant operator to both sides of Eq. 13.12. The block triangular matrices clearly have a determinant of one; thus we obtain another important result:

$$|M| = |M/H||H|. \quad (13.15)$$

(This result makes the choice of notation for the Schur complement seem quite natural!)

We are not yet finished. Note that we could alternatively have decomposed the matrix M in terms of E and M/E , yielding the following expression for the inverse:

$$\begin{bmatrix} E & F \\ G & H \end{bmatrix}^{-1} = \begin{bmatrix} E^{-1} + E^{-1}F(M/E)^{-1}GE^{-1} & -E^{-1}F(M/E)^{-1} \\ -(M/E)^{-1}GE^{-1} & (M/E)^{-1} \end{bmatrix}. \quad (13.16)$$

These two expressions for the inverse of M (Eq. 13.14 and Eq. 13.16) must be the same, thus we can set the corresponding blocks equal to each other. This yields:

$$(E - FH^{-1}G)^{-1} = E^{-1} + E^{-1}F(H - GE^{-1}F)^{-1}GE^{-1} \quad (13.17)$$

and

$$(E - FH^{-1}G)^{-1}FH^{-1} = E^{-1}F(H - GE^{-1}F)^{-1} \quad (13.18)$$

Both Eq. 13.17, which is generally referred to as the “matrix inversion lemma,” and Eq. 13.18 are quite useful in transformations involving Gaussian distributions. They allow expressions involving the inverse of E to be converted into expressions involving the inverse of H and vice versa.

13.4 Marginalization and conditioning

We now make use of our block diagonalization results to develop general formulas for the key operations of marginalization and conditioning in the multivariate Gaussian setting. We present results for both the moment parameterization and the canonical parameterization.

Our goal is to split the joint distribution Eq. 13.10 into a marginal probability for x_2 and a conditional probability for x_1 according to the factorization $p(x_1, x_2) = p(x_1|x_2)p(x_2)$. Focusing

first on the exponential factor, we make use of Eq. 13.12:

$$\begin{aligned}
& \exp \left\{ -\frac{1}{2} \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{pmatrix}^T \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}^{-1} \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{pmatrix} \right\} \\
&= \exp \left\{ -\frac{1}{2} \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{pmatrix}^T \begin{bmatrix} I & 0 \\ -\Sigma_{22}^{-1}\Sigma_{21} & I \end{bmatrix} \begin{bmatrix} (\Sigma/\Sigma_{22})^{-1} & 0 \\ 0 & \Sigma_{22}^{-1} \end{bmatrix} \begin{bmatrix} I & -\Sigma_{12}\Sigma_{22}^{-1} \\ 0 & I \end{bmatrix} \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{pmatrix} \right\} \\
&= \exp \left\{ -\frac{1}{2}(x_1 - \mu_1 - \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2))^T(\Sigma/\Sigma_{22})^{-1}(x_1 - \mu_1 - \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2)) \right\} \\
&\quad \cdot \exp \left\{ -\frac{1}{2}(x_2 - \mu_2)^T\Sigma_{22}^{-1}(x_2 - \mu_2) \right\}. \tag{13.19}
\end{aligned}$$

We next exploit Eq. 13.15 to split the normalization into two factors:

$$\frac{1}{(2\pi)^{(p+q)/2}|\Sigma|^{1/2}} = \frac{1}{(2\pi)^{(p+q)/2}(|\Sigma/\Sigma_{22}| |\Sigma_{22}|)^{1/2}} \tag{13.20}$$

$$= \left(\frac{1}{(2\pi)^{p/2}|\Sigma/\Sigma_{22}|^{1/2}} \right) \left(\frac{1}{(2\pi)^{q/2}|\Sigma_{22}|^{1/2}} \right) \tag{13.21}$$

To see that we have achieved our goal of factorizing the joint distribution into the product of a marginal distribution and a conditional distribution, note that if we group the first factor in Eq. 13.19 with the first factor in Eq. 13.21 we obtain a normalized Gaussian in the variable x_1 . Integrating with respect to x_1 , these factors disappear and the remaining factors must therefore represent the marginal distribution of x_2 :

$$p(x_2) = \frac{1}{(2\pi)^{q/2}|\Sigma/\Sigma_{22}|^{1/2}} \exp \left\{ -\frac{1}{2}(x_2 - \mu_2)^T\Sigma_{22}^{-1}(x_2 - \mu_2) \right\}. \tag{13.22}$$

Given this result, we are now licensed to interpret the factors that were integrated over as the conditional probability $p(x_1|x_2)$:

$$p(x_1|x_2) = \frac{1}{(2\pi)^{p/2}|\Sigma_{22}|^{1/2}} \exp \left\{ -\frac{1}{2}(x_1 - \mu_1 - \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2))^T(\Sigma/\Sigma_{22})^{-1}(x_1 - \mu_1 - \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2)) \right\} \tag{13.23}$$

To summarize our results, let (μ_2^m, Σ_2^m) denote the moment parameters of the marginal distribution of x_2 , and let $(\mu_{1|2}^c, \Sigma_{1|2}^c)$ denote the moment parameters of the conditional distribution of x_1 given x_2 . Eq. 13.22 and Eq. 13.23 yield the following expressions for these parameters:

Marginalization:

$$\mu_2^m = \mu_2 \tag{13.24}$$

$$\Sigma_2^m = \Sigma_{22} \tag{13.25}$$

Conditioning:

$$\mu_{1|2}^c = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2) \tag{13.26}$$

$$\Sigma_{1|2}^c = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} \tag{13.27}$$

We can also express the marginalization and conditioning operations in the canonical parameterization. The results can be obtained directly from the joint distribution, or indirectly by converting from the results for the moment parameterization. In any case, the results—which we ask the reader to derive in Exercise ??—are as follows:

Marginalization:

$$\eta_2^m = \eta_2 - \Lambda_{21}\Lambda_{11}^{-1}\eta_1 \quad (13.28)$$

$$\Lambda_2^m = \Lambda_{22} - \Lambda_{21}\Lambda_{11}^{-1}\Lambda_{12} \quad (13.29)$$

Conditioning:

$$\eta_{1|2}^c = \eta_1 - \Lambda_{12}x_2 \quad (13.30)$$

$$\Lambda_{1|2}^c = \Lambda_{11} \quad (13.31)$$

Note that the marginalization operation is simple in the moment parameterization and conditioning is complicated, whereas the opposite holds in the canonical parameterization.

13.5 Maximum likelihood estimation

Let us now suppose that we have an independently, identically distributed data set $\mathcal{D} = \{x_1, x_2, \dots, x_N\}$ sampled from a multivariate Gaussian distribution. We want to estimate the parameters of the distribution via maximum likelihood.

We form the log likelihood function by taking the logarithm of the product of N Gaussians:

$$l(\mu, \Sigma | \mathcal{D}) = -\frac{N}{2} \log |\Sigma| - \frac{1}{2} \sum_{i=1}^N (x_i - \mu)^T \Sigma^{-1} (x_i - \mu). \quad (13.32)$$

Taking the derivative with respect to μ is straightforward:

$$\frac{\partial l}{\partial \mu} = \sum_{i=1}^N (x_i - \mu)^T \Sigma^{-1}, \quad (13.33)$$

and setting to zero we obtain a pleasant result:

$$\hat{\mu} = \frac{1}{N} \sum_{i=1}^N x_i. \quad (13.34)$$

That is, the maximum likelihood estimate of the mean is the sample mean.

We now turn to the more challenging problem of computing the maximum likelihood estimate of the covariance matrix. Although the tools that we require to solve this problem are inevitably somewhat technical, they are important, and will reappear on several occasions later in the text.

Letting $l(\Sigma | \mathcal{D})$ denote those terms in the log likelihood that are a function of Σ , we obtain:

$$l(\Sigma | \mathcal{D}) = -\frac{N}{2} \log |\Sigma| - \frac{1}{2} \sum_n (x_n - \mu)^T \Sigma^{-1} (x_n - \mu). \quad (13.35)$$

We need to take the derivative with respect to Σ and set to zero. To calculate this derivative we require a short detour.

13.5.1 Traces and derivatives

The trace of a square matrix A is defined to be the sum of the diagonal elements a_{ii} of A :

$$\text{tr}[A] \triangleq \sum_i a_{ii} \quad (13.36)$$

An important property possessed by the trace is its invariance under cyclical permutations of matrix products:

$$\text{tr}[ABC] = \text{tr}[CAB] = \text{tr}[BCA], \quad (13.37)$$

where A , B and C are arbitrary matrices whose dimensions are conformal and are such that the product ABC (and therefore the other two products) is a square matrix. This result is easily established by writing out the matrix products explicitly.

Our interest in the trace is due to its usefulness in calculating derivatives of quadratic forms. In particular, we need to take the derivative of an expression of the form $x^T Ax$ with respect to the matrix A . There is a “trick” involving the trace that makes such calculations easy. We write:

$$x^T Ax = \text{tr}[x^T Ax] = \text{tr}[xx^T A] \quad (13.38)$$

where the first equality follows from the fact that $x^T Ax$ is a scalar. We see that we can calculate derivatives of quadratic forms by calculating derivatives of traces.

Let us calculate the derivative of $\text{tr}[BA]$ with respect to A . We write out the matrix product explicitly and calculate the partial with respect to a matrix element a_{ij} :

$$\frac{\partial}{\partial a_{ij}} \text{tr}[AB] = \frac{\partial}{\partial a_{ij}} \sum_k \sum_l a_{kl} b_{lk} \quad (13.39)$$

$$= b_{ji}, \quad (13.40)$$

which shows that

$$\frac{\partial}{\partial A} \text{tr}[BA] = B^T. \quad (13.41)$$

We can now use this result to calculate the derivative of the quadratic form $x^T Ax$:

$$\frac{\partial}{\partial A} x^T Ax = \frac{\partial}{\partial A} \text{tr}[xx^T A] = [xx^T]^T = xx^T, \quad (13.42)$$

which is the outer product of the vector x with itself.

13.5.2 Determinants and derivatives

From Eq. (13.35) we see that we also need to take the derivative of the logarithm of a determinant. In this section we establish the following result:

$$\frac{\partial}{\partial A} \log |A| = A^{-T}, \quad (13.43)$$

which together with the result on traces will allow us to solve our estimation problem.

To establish the result, note that:

$$\frac{\partial}{\partial a_{ij}} \log |A| = \frac{1}{|A|} \frac{\partial}{\partial a_{ij}} |A| \quad (13.44)$$

and recall the formula for the matrix inverse:

$$A^{-1} = \frac{1}{|A|} \tilde{A}, \quad (13.45)$$

where \tilde{A} is the matrix of cofactors. This shows that we need only establish the following result:

$$\frac{\partial}{\partial a_{ij}} |A| = \tilde{A}. \quad (13.46)$$

The determinant of a square matrix A can be expanded as follows (this formula is often taken as the definition of the determinant):

$$|A| = \sum_j (-1)^{i+j} a_{ij} M_{ij} \quad (13.47)$$

where M_{ij} is the minor associated with matrix element a_{ij} (the determinant of the matrix obtained by removing the i th row and j th column of A). Note that this formula holds for arbitrary i .

Given that a_{ij} does not appear in any of the minors in the sum, the derivative of $|A|$ with respect to a_{ij} is just $(-1)^{i+j} M_{ij}$. But the matrix of these values is simply the transpose of the matrix of cofactors.

13.5.3 Maximum likelihood estimate of Σ

With these two results in hand, we return to the problem of calculating the maximum likelihood estimate of the covariance matrix Σ .

We use the trace trick to cope with the quadratic form:

$$l(\Sigma | \mathcal{D}) = -\frac{N}{2} \log |\Sigma| - \frac{1}{2} \sum_n (x_n - \mu)^T \Sigma^{-1} (x_n - \mu) \quad (13.48)$$

$$= \frac{N}{2} \log |\Sigma^{-1}| - \frac{1}{2} \sum_n \text{tr}[(x_n - \mu)^T \Sigma^{-1} (x_n - \mu)] \quad (13.49)$$

$$= \frac{N}{2} \log |\Sigma^{-1}| - \frac{1}{2} \sum_n \text{tr}[(x_n - \mu)(x_n - \mu)^T \Sigma^{-1}], \quad (13.50)$$

where we have also used the fact that the determinant of the inverse of a matrix is the inverse of the determinant.

We now take the derivative with respect to the matrix Σ^{-1} :

$$\frac{\partial l}{\partial \Sigma^{-1}} = \frac{N}{2} \Sigma - \frac{1}{2} \sum_n (x_n - \mu)(x_n - \mu)^T. \quad (13.51)$$

Finally, setting to zero yields the maximum likelihood estimator:

$$\hat{\Sigma}_{ML} = \frac{1}{N} \sum_n (x_n - \hat{\mu}_{ML})(x_n - \hat{\mu}_{ML})^T, \quad (13.52)$$

which is the expected result.

This result also allows us to obtain maximum likelihood estimates for the canonical parameters:

$$\hat{\Lambda} = \hat{\Sigma}_{ML}^{-1} \quad (13.53)$$

$$\hat{\eta} = \hat{\Sigma}_{ML}^{-1} \hat{\mu}_{ML}; \quad (13.54)$$

where we have used the fact that maximum likelihood estimates are invariant to changes in parameterization (see Exercise ??).

There is much more to say about maximum likelihood estimation for the multivariate Gaussian. In particular, it is significantly more interesting to obtain estimates of Σ when we have constraints on Λ , for example a constraint that requires that certain elements of Λ are zero. Indeed, as we will see in Chapter ??, this is a rather natural constraint in the world of graphical models.

13.6 Historical remarks and bibliography

[section yet to be written].

Chapter 14

Factor Analysis

In this chapter we present a latent variable model in which the latent variable is a continuous random vector. The problem that we address is one of density estimation, although the ideas that we describe can be exploited in regression and classification settings well.

In many density estimation problems, the measured data vector may be high-dimensional, but we may have reason to believe that the data lie near a lower-dimensional manifold. In such a setting it may be useful to model the data generation process as a two-stage process, in which (1) a point in the manifold is generated according to a (simple) probability density, and (2) the observed data are generated conditionally from another (simple) density that is centered on the point. The coordinates of this point form the components of a latent random vector. Assuming that we wish to parameterize a continuous manifold, the latent variable is a continuous-valued random vector.

When we assume that the manifold is a linear subspace, we obtain a model known as *factor analysis*. Figure 14.3 shows the geometry underlying the factor analysis model. The observed data are assumed to lie near a p -dimensional subspace \mathcal{M} in \mathbb{R}^q , where $p < q$. Given a set of basis vectors $\{\lambda_j\}$, a point in \mathcal{M} can be represented as the product Λx , where x is a coordinate vector and Λ is the matrix whose columns are the basis vectors $\{\lambda_j\}$. We treat the coordinate vectors as values of a continuous random vector X , endowing X with a probability density. Specifically, in the case of factor analysis, we assume that X is a Gaussian random vector. Finally, given a point in \mathcal{M} , the observed data Y are assumed to be generated according to a Gaussian distribution centered around that point. We can view the resulting density as the convolution of the Gaussian density on the manifold with a Gaussian distribution extending into \mathbb{R}^q , resulting in a “thick subspace” lying in \mathbb{R}^q .

From the point of view of generating the “thick subspace,” the basis vectors for \mathcal{M} are not unique. Indeed, as we will see in Section 14.1, any orthogonal transformation of the basis vectors leaves the likelihood invariant. Historically, factor analysis arose in a setting in which the goal was often that of recovering unique basis vectors—these are the “factors” that give the model its name. The lack of identifiability of the basis vectors is a problem from this point of view, and factor analysis has often been seen as “controversial.” From the point of view of density estimation, however, the interest is in the subspace and not any particular basis for describing the subspace. In this setting, factor analysis should not be viewed as any more “controversial” than any of the

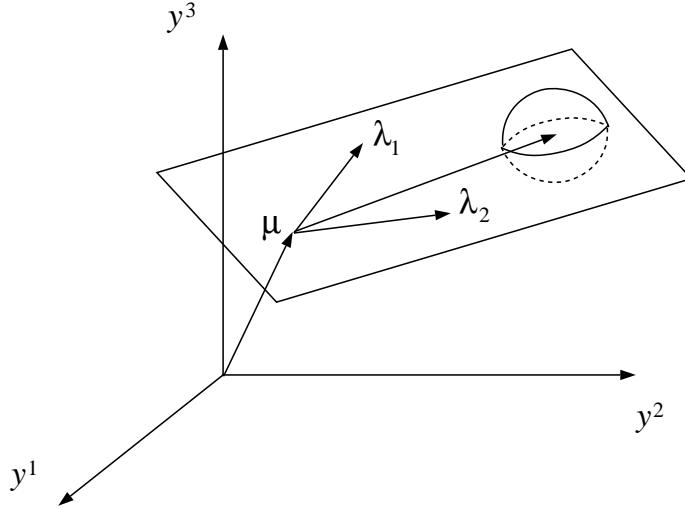


Figure 14.1: The geometry of the factor analysis model.

other models that we have described.

Factor analysis is closely related to *principal component analysis (PCA)*—another linear method for dimensionality reduction. Indeed, one can obtain PCA as a limiting case of factor analysis, much as one obtains the K-means algorithm as a limiting case of the Gaussian mixture model. We discuss PCA and some of the links between PCA and factor analysis in Section 14.4.

14.1 The model

The factor analysis model is shown as a graphical model in Figure 14.2. The model is comprised of a latent Gaussian variable X and an observable variable Y , where X is a p -dimensional random vector and Y a q -dimensional random vector, and where we assume $p < q$.

The model is parameterized as follows. Let X have a marginal Gaussian distribution:

$$X \sim \mathcal{N}(0, I), \quad (14.1)$$

with zero mean and an identity covariance matrix. Let the conditional distribution of Y be Gaussian, with mean $\mu + \Lambda x$:

$$Y \sim \mathcal{N}(\mu + \Lambda x, \Psi), \quad (14.2)$$

where Ψ is a diagonal covariance matrix.

The product of Gaussian distributions is Gaussian, and thus the joint distribution of X and Y is Gaussian, as are all marginals and conditionals computed under this joint. In particular, let us calculate the marginal of Y and the conditional distribution of X given Y .

We first calculate the marginal probability of Y , doing the calculation in two ways. The first approach is based on expressing Y as a sum:

$$Y = \mu + \Lambda x + W, \quad (14.3)$$

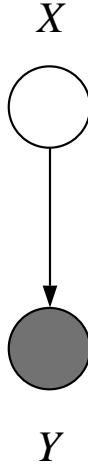


Figure 14.2: The factor analysis model as a graphical model.

where W is distributed as $\mathcal{N}(0, \Psi)$, and is independent of X . It is easy to verify that this representation yields the conditional distribution for Y in Eq. (14.2). We now make use of this representation to calculate the unconditional mean of Y :

$$E(Y) = E(\mu + \Lambda X + W) \quad (14.4)$$

$$= \mu + \Lambda EX + EW \quad (14.5)$$

$$= \mu, \quad (14.6)$$

and the unconditional covariance matrix of Y :

$$\text{Var}(Y) = E[(\mu + \Lambda X + W - \mu)(\mu + \Lambda X + W - \mu)^T] \quad (14.7)$$

$$= E[(\Lambda X + W)(\Lambda X + W)^T] \quad (14.8)$$

$$= \Lambda E(X X^T) \Lambda^T + E(W W^T) \quad (14.9)$$

$$= \Lambda \Lambda^T + \Psi. \quad (14.10)$$

Given that Y is Gaussian these two results determine the marginal distribution of Y .

There is another way to obtain these results that is based on the relationship between the conditional mean and covariance of a random vector Y and the unconditional mean and covariance. This approach will be particularly useful in Chapter 15. Recall that the conditional expectation, $E(Y | X)$, is a random variable, and thus it is meaningful to compute means and variances of $E(Y | X)$. In particular, in Appendix XXX we derive the following relationship:

$$E(Y) = E(E(Y | X)), \quad (14.11)$$

a result known as the *iterated expectation theorem*, and an analogous result for the variance:

$$\text{Var}(Y) = \text{Var}(E(Y | X)) + E(\text{Var}(Y | X)). \quad (14.12)$$

Let us now make use of these relationships to derive the unconditional distribution of Y for factor analysis. Using Eq. (14.11), we have:

$$E(Y) = E(\mu + \Lambda X) = \mu. \quad (14.13)$$

and from Eq. (14.12), we obtain:

$$\text{Var}(Y) = \text{Var}(\mu + \Lambda X) + E\Psi \quad (14.14)$$

$$= E[(\Lambda X)(\Lambda X)^T] + \Psi \quad (14.15)$$

$$= \Lambda\Lambda^T + \Psi; \quad (14.16)$$

the same results as before.

We also need to calculate the covariance of X and Y , which we can compute using Eq. (14.3):

$$\text{Cov}(X, Y) = E[X(\mu + \Lambda X + W - \mu)^T] \quad (14.17)$$

$$= E[X(\Lambda X + W)^T] \quad (14.18)$$

$$= \Lambda^T. \quad (14.19)$$

We can also obtain this expression using the following result from Appendix XXX:

$$\text{Cov}(X, Y) = \text{Cov}(X, E(Y | X)), \quad (14.20)$$

which yields:

$$\text{Cov}(X, Y) = \text{Cov}(X, \mu + \Lambda X) = \Lambda^T \quad (14.21)$$

as before.

Collecting together the results thus far, we have shown that the joint distribution of X and Y is a Gaussian with mean vector $(0, \mu^T)^T$ and covariance matrix:

$$\Sigma = \begin{bmatrix} I & \Lambda^T \\ \Lambda & \Lambda\Lambda^T + \Psi \end{bmatrix}. \quad (14.22)$$

We can now calculate the conditional distribution of X given Y , using the results from Chapter 13. We begin by calculating the conditional mean of X :

$$E(X | y) = \Lambda^T(\Lambda\Lambda^T + \Psi)^{-1}(y - \mu). \quad (14.23)$$

Note that the matrix that must be inverted in this calculation is a $(q \times q)$ -dimensional matrix. It is also possible to exploit Eq. (??) and invert a $(p \times p)$ -dimensional matrix instead:

$$E(X | y) = (I + \Lambda^T\Psi^{-1}\Lambda)^{-1}\Lambda^T\Psi^{-1}(y - \mu). \quad (14.24)$$

In the context of factor analysis, in which $p < q$, Eq. (14.24) is the preferred way to compute the conditional expectation.

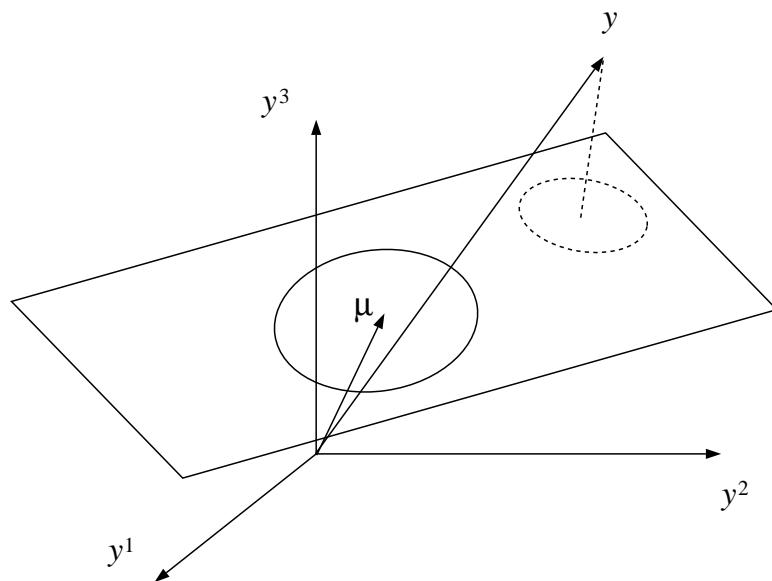


Figure 14.3: The solid ellipse corresponds to the Gaussian distribution of the latent variable X prior to the observation of Y . After $Y = y$ is observed, the distribution of X is depicted as a dotted ellipse. The mean of the updated distribution given by Eq. (14.24) and the covariance is given by Eq. (14.26).

We also compute the conditional variance of X :

$$\text{Var}(X | y) = I - \Lambda^T(\Lambda\Lambda^T + \Psi)^{-1}\Lambda \quad (14.25)$$

$$= (I + \Lambda^T\Psi^{-1}\Lambda)^{-1}, \quad (14.26)$$

where we have used Eq. (??) in the second step. Again, for dimensionality reasons, we usually prefer to implement Eq. (14.26).

Figure ?? summarizes our results from a geometric point of view. Before observing Y , the distribution of X is a Gaussian centered around the origin of the latent variable subspace. After an observation $Y = y$, we obtain an updated distribution for X , where Eq. (14.24) determines the mean and Eq. (14.26) determines the updated covariance matrix of the updated distribution. We in essence “project” y onto the latent subspace, obtaining not only a point projection, but an estimate of uncertainty as well.

In summary, we have obtained the probability distribution of the latent variable given the observed variable—the analog of the calculation of the posterior probability τ in Chapter ??.

14.2 Maximum likelihood estimation

We now turn to the problem of finding maximum likelihood estimates of the parameters of the factor analysis model.

14.2.1 The log likelihood

As in the case of the mixture model, the likelihood for the factor analysis model is a marginal probability. We have a seeming advantage in the case of the factor analysis, however, because the marginal probability can be calculated analytically. Indeed, as we have seen, the marginal probability of Y is a Gaussian with mean μ and covariance matrix $\Lambda\Lambda^T + \Psi$. The log likelihood is therefore a Gaussian log likelihood:

$$l(\theta | D) = -\frac{N}{2} \log |\Lambda\Lambda^T + \Psi| - \frac{1}{2} \left\{ \sum_n (y_n - \mu)^T (\Lambda\Lambda^T + \Psi)^{-1} (y_n - \mu) \right\}, \quad (14.27)$$

where as usual we assume an IID data set $\mathcal{D} = \{y_n : n = 1, \dots, N\}$.

We noted earlier that the likelihood remains invariant to orthogonal transformations of Λ . To show this, let R be an orthogonal matrix, and let $\tilde{\Lambda} = \Lambda R$. We have:

$$\tilde{\Lambda}\tilde{\Lambda}^T = \Lambda R(\Lambda R)^T = \Lambda R R^T \Lambda^T = \Lambda\Lambda^T. \quad (14.28)$$

Thus the likelihood, which depends on Λ only through the product $\Lambda\Lambda^T$ is not changed if Λ is postmultiplied by R .

One useful consequence of Eq. (14.27) is an analytical formula for estimating the mean μ . Indeed, from the point of view of estimating μ , Eq. (14.27) is simply a Gaussian log likelihood, and we obtain the usual maximum likelihood estimate:

$$\hat{\mu}_{ML} = \frac{1}{N} \sum_n y_n, \quad (14.29)$$

by differentiating with respect to μ and setting to zero.

In the remainder of this section, we will omit reference to the mean μ in order to simplify our notation. In practice we estimate μ according to Eq. (14.29) and then subtract the estimate from the data vectors. The resulting centered variables play the role of the data vectors y_n in the remainder of our discussion.

Unfortunately further progress in estimating the parameters is stymied—the parameters Λ and Ψ are coupled in Eq. (14.27), both by the determinant and the matrix inverse. There are no closed form expressions for the maxima of the log likelihood with respect to these parameters.

To decouple the parameters and obtain a simple algorithm for maximum likelihood estimation in factor analysis, we again make use of the EM algorithm.¹

14.2.2 An EM algorithm

To derive an EM algorithm, we follow the recipe discussed in Chapter 11. In particular, we write down the complete log likelihood, take the expectation, and maximize the resulting expected complete log likelihood with respect to the parameters.

Before immersing ourselves in the algebra, however, let us step back and consider the results that we expect to obtain. Suppose in particular that we have “complete data”—pairs of observations of X and Y . Clearly the estimation of the distribution of X would reduce to a Gaussian density estimation problem in this case, although given our assumption that X has zero mean and identity covariance matrix, we have no parameters to estimate for the distribution of X . From Eq. (14.3) we see that Y is a linear function of x , with additive white Gaussian noise W . Thus, if both X and Y were observed, we would have a linear regression problem.

This argument suggests that if we can “fill in” X in the E step, we should find that the M step reduces to estimating Λ and Ψ using linear regression. This is in fact correct, but we need to take care in defining what we mean by “fill in.” In particular, it is not correct to simply replace X with its conditional expectation in the linear regression formulas. To obtain the correct result, we need to calculate the expected complete log likelihood and identify the expected sufficient statistics for our problem.

14.2.3 The E step

Given complete data, $\mathcal{D}_c = \{(x_n, y_n) : n = 1, \dots, N\}$, the complete likelihood is simply a product of Gaussian distributions. Taking the logarithm, we obtain:

$$l_c(\theta | \mathcal{D}_c) = -\frac{N}{2} \log |\Psi| - \frac{1}{2} \sum_n x_n^T x_n - \frac{1}{2} \sum_n (y_n - \Lambda x_n)^T \Psi^{-1} (y_n - \Lambda x_n). \quad (14.30)$$

Although this expression may appear daunting, it has the important property that Λ and Ψ are decoupled, although this fact may not yet be clear. It is in fact a much simpler expression to maximize than the log likelihood.

¹An alternative approach is to use a nonlinear optimization algorithm such as conjugate gradients. See [?] for a presentation of this approach.

To make further progress we use the “trace trick.” Rewriting the quadratic forms, we have:

$$\begin{aligned} l_c(\theta | \mathcal{D}_c) &= -\frac{N}{2} \log |\Psi| - \frac{1}{2} \sum_n \text{tr}(x_n^T x_n) - \frac{1}{2} \sum_n \text{tr}[(y_n - \Lambda x_n)^T \Psi^{-1} (y_n - \Lambda x_n)] \\ &= -\frac{N}{2} \log |\Psi| - \frac{1}{2} \sum_n \text{tr}(x_n x_n^T) - \frac{1}{2} \sum_n \text{tr}[(y_n - \Lambda x_n)(y_n - \Lambda x_n)^T \Psi^{-1}] \\ &= -\frac{N}{2} \log |\Psi| - \frac{N}{2} \text{tr}(S \Psi^{-1}), \end{aligned} \quad (14.31)$$

where we have defined:

$$S \triangleq \frac{1}{N} \sum_n (y_n - \Lambda x_n)(y_n - \Lambda x_n)^T. \quad (14.32)$$

Note that this matrix has the form of a sample covariance matrix, although it depends on the unknown parameter Λ .

We now take the conditional expectation of the complete log likelihood, conditioning on the observed data y and the current parameter vector $\theta^{(t)}$. Using the operator notation $\langle \cdot \rangle$ to denote this conditional expectation, we obtain:

$$Q(\theta | \theta^{(t)}) = -\frac{N}{2} \log |\Psi| - \frac{N}{2} \text{tr}(\langle S \rangle \Psi^{-1}), \quad (14.33)$$

We now calculate the conditional expectation $\langle S \rangle$ —where we substitute the random variable X_n for x_n in the definition of S and treat S as a random quantity. We have:

$$\langle S \rangle = \frac{1}{N} \sum_n \langle y_n y_n^T - y_n X_n^T \Lambda^T - \Lambda X_n y_n^T + \Lambda X_n X_n^T \Lambda^T \rangle \quad (14.34)$$

$$= \frac{1}{N} \sum_n (y_n y_n^T - y_n \langle X_n^T \rangle \Lambda^T - \Lambda \langle X_n \rangle y_n^T + \Lambda \langle X_n X_n^T \rangle \Lambda^T), \quad (14.35)$$

where we see that we require the conditional expectations $\langle X_n \rangle$ and $\langle X_n X_n^T \rangle$.

In summary, the expected sufficient statistics that we require for the E step are the conditional expectations $\langle X_n \rangle$ and $\langle X_n X_n^T \rangle$. We have already obtained these expectations in Section 14.1. Thus:

$$\langle X_n \rangle = E(X_n | Y_n) \quad (14.36)$$

$$\langle X_n X_n^T \rangle = \text{Var}(X_n | y_n) + E(X_n | y_n)E(X_n | y_n)^T. \quad (14.37)$$

With these equations we “fill in” the conditional distribution of the latent variable X_n .

14.2.4 The M step

We now turn to the M step. To calculate the necessary derivatives, let us recall from Section 13.5.2 how to take derivatives of log determinants:

$$\frac{\partial}{\partial A} \log |A| = A^{-T}, \quad (14.38)$$

and derivatives of traces:

$$\frac{\partial}{\partial A} \text{tr}[BA] = B^T. \quad (14.39)$$

In Appendix A we also derive a slight extension of the latter result:

$$\frac{\partial}{\partial A} \text{tr}[BA^TCA] = 2CAB, \quad (14.40)$$

when B and C are symmetric.

We compute the derivative of Q with respect to Λ . The relevant terms are:

$$Q(\Lambda | \theta^{(t)}) = -\frac{1}{2} \sum_n \text{tr} \left\{ (y_n y_n^T - y_n \langle X_n^T \rangle \Lambda^T - \Lambda \langle X_n \rangle y_n^T + \Lambda \langle X_n X_n^T \rangle \Lambda^T) \Psi^{-1} \right\}. \quad (14.41)$$

Taking the derivative, we obtain:

$$\frac{\partial Q}{\partial \Lambda} = \sum_n \Psi^{-1} y_n \langle X_n^T \rangle - \sum_n \Psi^{-1} \Lambda \langle X_n X_n^T \rangle, \quad (14.42)$$

where we have used the fact that $\text{tr}[A] = \text{tr}[A^T]$ and the circulation property of the trace. Setting to zero, we obtain:

$$\Lambda^{(t+1)} = \left(\sum_n y_n \langle X_n^T \rangle \right) \left(\sum_n \langle X_n X_n^T \rangle \right)^{-1}. \quad (14.43)$$

This is the expected result—we have obtained the normal equations from linear regression.²

Finally we compute the derivative of Q with respect to Ψ . The terms in the expected complete log likelihood that depend on Ψ are:

$$Q(\Psi | \theta^{(t)}) = -\frac{N}{2} \log |\Psi| - \frac{N}{2} \text{tr}(\langle S \rangle \Psi^{-1}). \quad (14.44)$$

Recall that Ψ is a diagonal matrix. To calculate the derivative of Q with respect to Ψ , we take the usual matrix derivative but retain only the diagonal terms. Taking the derivative with respect to Ψ^{-1} , setting to zero and retaining only the diagonal terms yields:

$$\Psi^{(t+1)} = \text{diag}(\langle S \rangle). \quad (14.45)$$

Recall that S depends on the parameter Λ . Thus we must substitute $\Lambda^{(t+1)}$ from Eq. (14.43) into the expression for $\langle S \rangle$. Carrying out this substitution, we find that we in fact obtain a simplification:

$$\langle S \rangle = \frac{1}{N} \sum_n \left(y_n y_n^T - y_n \langle X_n^T \rangle \Lambda^{(t+1)T} - \Lambda^{(t+1)} \langle X_n \rangle y_n^T + \Lambda^{(t+1)} \langle X_n X_n^T \rangle \Lambda^{(t+1)T} \right) \quad (14.46)$$

$$= \frac{1}{N} \left(\sum_n y_n y_n^T - \Lambda^{(t+1)} \langle X_n \rangle y_n^T \right), \quad (14.47)$$

²Note that in univariate regression the parameter vector enters into the regression model as a *row* vector. Writing the regression model using the notation of this chapter, we have: $Y = \mu + \theta^T x + W$, where Y is a scalar. In factor analysis each row of the matrix Λ corresponds to a parameter vector in univariate regression: $Y = \mu + \Lambda x + W$, where Y is a vector. Thus we should expect to obtain the transpose of the usual normal equations, and this is indeed what we obtain in Eq. (14.43).

and thus we have:

$$\Psi^{(t+1)} = \frac{1}{N} \text{diag} \left\{ \sum_n y_n y_n^T - \Lambda^{(t+1)} \sum_n \langle X_n \rangle y_n^T \right\}, \quad (14.48)$$

as the M step for Ψ .

14.3 Mixtures of factor analyzers

[Section not yet written].

14.4 Principal components analysis and factor analysis

[Section not yet written].

14.5 Appendix A

In this section we show how to calculate the derivative of the expression $\text{tr}[BA^TCA]$ with respect to A . The calculation is based on the equation:

$$\frac{\partial}{\partial A} \text{tr}[BA] = B^T \quad (14.49)$$

that we established in Section 13.5.1.

We use the product rule, first holding A^T constant and then holding A constant. Placing a bar over a matrix to indicate that it is being treated as a constant, we have:

$$\frac{\partial}{\partial A} \text{tr}[BA^TCA] = \frac{\partial}{\partial A} \text{tr}[B\bar{A}^TCA] + \frac{\partial}{\partial A} \text{tr}[BA^TCA\bar{A}] \quad (14.50)$$

$$= (BA^T C)^T + \frac{\partial}{\partial A} \text{tr}[\bar{A}^T C^T AB^T] \quad (14.51)$$

$$= C^T AB^T + \frac{\partial}{\partial A} \text{tr}[B^T \bar{A}^T C^T A] \quad (14.52)$$

$$= C^T AB^T + (B^T A^T C^T)^T \quad (14.53)$$

$$= C^T AB^T + CAB. \quad (14.54)$$

A special case of this result is obtained when B and C are symmetric:

$$\frac{\partial}{\partial A} \text{tr}[BA^TCA] = 2CAB, \quad (14.55)$$

which is the result that we require in Section 14.2.4.

Chapter 15

Kalman filtering and smoothing

Thus far we have presented two major categories of latent variable models: *mixture models*, which are based on a discrete latent variable, and *factor analysis models*, which are based on a continuous latent variable. The graphs underlying these models are identical—two-node graphs in which a single latent variable is connected to a single observable variable.

Chapter 12 presented a dynamical generalization of mixture models—the hidden Markov model (HMM). Graphically, the HMM was obtained by copying the two-node mixture model as a spatial array, connecting successive state nodes in the array. It is natural to wonder if a similar generalization of factor analysis might be worth considering. In fact the dynamical generalization of factor analysis is well worth considering—it yields an interesting and important methodology for time series analysis known as the *Kalman filter*. In fact, in an attempt to develop a consistent terminology, we reserve the term “Kalman filter” for the recursive inference algorithm that is the analog of the “alpha” algorithm in the HMM setting. The underlying model, which we refer to as the “state space model (SSM),” is structurally identical to the HMM; only the type of the nodes (real-valued vectors) and the probability model (linear-Gaussian) changes. The model has exactly the same Markov properties as the HMM, and its states are hidden in exactly the same way as in the HMM.

Historically, the HMM and the Kalman filtering methodology were developed in separate research communities and their close relationship has not always been widely appreciated. This is partly due to the fact that the general framework of graphical models came later than the HMM and the Kalman filter. Without the graphical framework, the algorithms underlying the inference calculation in the two cases look rather different (as we will see). This is, however, simply a reflection of the differences between the multinomial distribution and the Gaussian distribution, and it is imperative that we not let these details—important as they may be in practice—obscure the fundamental similarity between the two models.

We will develop the inference procedures for the SSM in some detail in this chapter. This is not only to acknowledge the historical importance of the Kalman filter, but also to provide an additional concrete example of the solution of the inference problem for a reasonably complex graphical model. Once we develop a general perspective on graphical models in Chapter 15, we will return to the SSM and the HMM, not only to provide concrete examples to ground our general theory, but also

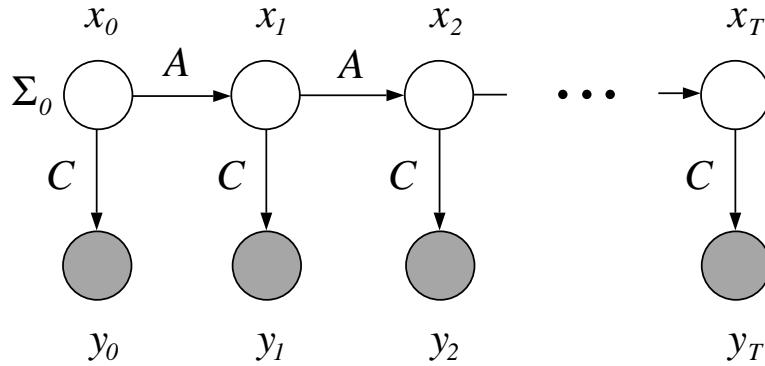


Figure 15.1: The SSM as a graphical model. Each vertical slice represents a time step. The top node in each slice represents the state variable x_t and the bottom node in each slice represents the observable output variable y_t .

to indicate that both are best viewed as jumping-off points for a much larger class of models.

15.1 The state space model

As we have already discussed, the model underlying Kalman filtering is a graphical model in the form of a chain (see Figure 15.1). We copy the two-node factor analysis model as an array and we link successive state nodes.

The independence relationships that characterize the SSM are identical to those that characterize the HMM. In particular, given the state at one moment in time, the states in the future are conditionally independent of those in the past. Moreover, the observation of the output nodes fails to separate any of the state nodes, and in general we expect for there to be a probabilistic relationship in the posterior distribution between all of the state nodes. As in the HMM, we hope that we can calculate these relationships recursively.

The state nodes in the factor analysis model are continuous, vector-valued nodes endowed with a Gaussian probability distribution. To develop a dynamical generalization of the factor analysis model we must represent the transition between the nodes at successive moments in time. Perhaps the simplest choice that we can make is to allow the mean of the state at time $t+1$ to be a linear function of the state at time t . Thus we write:

$$x_{t+1} = Ax_t + Gw_t, \quad (15.1)$$

where w_t is a “noise” term—a Gaussian random variable that is independent of w_s for $s < t$, and thus independent of x_t . We assume that w_t has zero mean and covariance matrix Q . Given that the sum of Gaussian variables is Gaussian, we have that x_{t+1} is indeed Gaussian. Conditional on x_t , its mean is Ax_t and its covariance is GQG^T .

In the factor analysis model, the output is endowed with a Gaussian distribution having a mean

that is a linear function of the state. We continue to use this model for the output of the SSM:

$$y_t = Cx_t + v_t, \quad (15.2)$$

where v_t is a Gaussian random variable with zero mean and covariance matrix R . Conditional on x_t , y_t is a Gaussian with mean Cx_t and covariance R .

Finally we endow the initial state, x_0 , with a Gaussian distribution having mean 0 and covariance Σ_0 . The assumption of zero mean is without loss of generality (a non-zero mean gives rise to a deterministic component that can be added to the probabilistic solution; see Exercise XXX for the details).

15.2 The unconditional distribution

Before beginning our investigation of the inference problem for the SSM, it is of interest to study the unconditional distribution of the states x_t .

The unconditional mean of x_t is clearly zero. This follows from the assumption that x_0 has zero mean, and via the dynamical equation (Eq. 15.1) each successive state has zero mean.

Turning to the unconditional covariance, which we denote Σ_t , we have:

$$\Sigma_{t+1} \triangleq E[x_{t+1}x_{t+1}^T] \quad (15.3)$$

$$= E[(Ax_t + Gw_t)(Ax_t + Gw_t)^T] \quad (15.4)$$

$$= AE[x_t x_t^T]A^T + GE[w_t w_t^T]G^T \quad (15.5)$$

$$= A\Sigma_t A^T + GQG^T, \quad (15.6)$$

where we have made use of our independence assumptions. This equation, a dynamical equation for the evolution of the unconditional covariance, is referred to as the *Lyapunov equation*.

It can also be verified that the unconditional covariance between neighboring states x_t and x_{t+1} is given by $\Sigma_t A^T$.

15.3 Inference

The inference problem for the SSM is the same as it was for the HMM—that of calculating the posterior probability of the states given an output sequence. Based on our experience with the HMM, we hope to be able to calculate such posterior probabilities recursively.

In the case of the HMM, we were able to decompose the inference problem into a “forward” problem and a “backward” problem. In the forward problem the evidence consisted of a partial sequence of outputs—all those outputs up to time t . The backward problem also utilized a partial sequence—all those outputs after time t . We will find that this same decomposition will yield recursive algorithms for the SSM.

As in the case of the HMM we distinguish between “filtering” and “smoothing”—two classes of problems that arise in this graphical model when we introduce evidence. We develop algorithms for solving both problems.

15.4 Filtering

The problem is to calculate an estimate of the state x_t based on a partial output sequence y_0, \dots, y_t . That is, we wish to calculate $P(x_t|y_0, \dots, y_t)$.¹

Sums of Gaussian variables are Gaussian, and thus, considering all of the variables in the SSM jointly, we have a (large) multivariate Gaussian distribution. Conditionals of Gaussians are Gaussian (see Chapter 13) and thus the probability distribution $P(x_t|y_0, \dots, y_t)$ must be Gaussian. This implies that we need only calculate a mean vector and a covariance matrix (or the corresponding canonical parameters). As we will see, inference in the SSM involves finding a recursion linking these conditional means and conditional covariances at neighboring moments in time.

We use a simplified notation for the conditional means and conditional covariances that emphasizes the particular output sequence being conditioned on. We write $\hat{x}_{t|t}$ to denote the mean of x_t conditioned on the partial sequence y_0, \dots, y_t . The covariance matrix of x_t conditioned on y_0, \dots, y_t is denoted $P_{t|t}$; thus:

$$\hat{x}_{t|t} \triangleq E[x_t|y_0, \dots, y_t] \quad (15.7)$$

$$P_{t|t} \triangleq E[(x_t - \hat{x}_{t|t})(x_t - \hat{x}_{t|t})^T|y_0, \dots, y_t]. \quad (15.8)$$

In our derivation of the algorithm, we will find that it is useful as an intermediate step to compute the probability distribution of x_t conditioned on y_0, \dots, y_{t-1} . In our new notation, this distribution has mean $\hat{x}_{t|t-1}$ and covariance matrix $P_{t|t-1}$.

To uncover the recursion behind the Kalman filter, let us refer to the graphical model fragments in Figure 15.2. In the fragment on the left, where we condition on the outputs y_0, \dots, y_t , we assume that we have already calculated $P(x_t|y_0, \dots, y_t)$; that is, we have calculated $\hat{x}_{t|t}$ and $P_{t|t}$. We wish to carry this distribution forward into the fragment on the right, where we condition on y_0, \dots, y_{t-1} . We decompose the transformation into two steps:

time update: $P(x_t|y_0, \dots, y_t) \rightarrow P(x_{t+1}|y_0, \dots, y_t)$

measurement update: $P(x_{t+1}|y_0, \dots, y_t) \rightarrow P(x_{t+1}|y_0, \dots, y_{t+1})$

Thus, in the *time update* step, we simply propagate the distribution forward one step in time, calculating the new mean and covariance based on the old mean and covariance, but based on no new measurements (i.e., no new outputs). In the *measurement update* step, we incorporate the new measurement y_{t+1} and update the probability distribution for x_{t+1} . The overall result is a transformation from $\hat{x}_{t|t}$ and $P_{t|t}$ to $\hat{x}_{t+1|t+1}$ and $P_{t+1|t+1}$.

Let us first consider the time update step. Recall the dynamic equation (Eq. 15.1):

$$x_{t+1} = Ax_t + Gw_t. \quad (15.9)$$

¹Note that this quantity is analogous to the normalized alpha variable from the HMM—the alpha variables themselves are *joint* probabilities: $P(x_t, y_0, \dots, y_t)$. The alphas and normalized alphas differ from each other, however, only by the normalization constant. In the Gaussian case we represent probability distributions by storing only the mean and covariance matrix (or the corresponding canonical parameters); the normalization factor is implicit. Thus there is no difference between “alphas” and “normalized alphas” in the SSM setting; all probabilities are implicitly normalized.

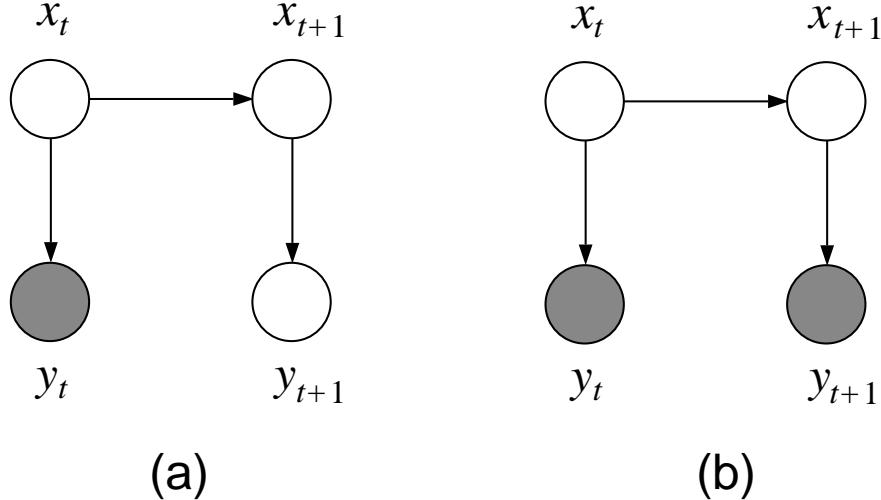


Figure 15.2: (a) A fragment of an SSM before a measurement update and (b) after a measurement update.

We take the conditional expectation on both sides of this equation. Given that w_t is independent of the conditioning variables y_0, \dots, y_t , the second term vanishes, and we have:

$$\hat{x}_{t+1|t} = A\hat{x}_{t|t}. \quad (15.10)$$

Similarly, taking the conditional covariance of both sides of the dynamic equation, we have:

$$P_{t+1|t} = E \left[(x_{t+1} - \hat{x}_{t+1|t})(x_{t+1} - \hat{x}_{t+1|t})^T \mid y_0, \dots, y_t \right] \quad (15.11)$$

$$= E[(Ax_t + Gw_t - A\hat{x}_{t|t})(Ax_t + Gw_t - A\hat{x}_{t|t})^T | y_0, \dots, y_t] \quad (15.12)$$

$$= AP_{t|t}A^T + GQG^T, \quad (15.13)$$

where we have used the facts that $\hat{x}_{t+1|t}$ is a constant in the conditional distribution, w_t has zero mean, and w_t and x_t are independent.

Now that we know the conditional distribution of x_{t+1} we proceed further in the graphical model fragment and calculate the conditional mean and covariance of y_{t+1} , as well as the conditional covariance of x_{t+1} and y_{t+1} . These calculations allow us to write down the joint conditional distribution of x_{t+1} and y_{t+1} , at which point the measurement update becomes a simple matter of “reversing the arrow”—calculating the conditional distribution of x_{t+1} given y_{t+1} .

The calculations are straightforward:

$$E[y_{t+1}|y_0, \dots, y_t] = E[Cx_{t+1} + v_{t+1}|y_0, \dots, y_t] \quad (15.14)$$

$$= C\hat{x}_{t+1|t} \quad (15.15)$$

$$E \left[(y_{t+1} - \hat{y}_{t+1|t}) (y_{t+1} - \hat{y}_{t+1|t})^T | y_0, \dots, y_t \right]$$

$$= E [(Cx_{t+1} + v_{t+1} - C\hat{x}_{t+1|t})(Cx_{t+1} + v_{t+1} - C\hat{x}_{t+1|t})^T | y_0, \dots, y_t] \quad (15.16)$$

$$= CP_{t+1|t}C^T + R \quad (15.17)$$

and

$$E [(y_{t+1} - \hat{y}_{t+1|t})(x_{t+1} - \hat{x}_{t+1|t})^T | y_0, \dots, y_t]$$

$$= E [(Cx_{t+1} + v_{t+1} - \hat{y}_{t+1|t})(x_{t+1} - \hat{x}_{t+1|t})^T | y_0, \dots, y_t] \quad (15.18)$$

$$= CP_{t+1|t}, \quad (15.19)$$

where we have made use of the various independence assumptions.

We summarize these results as follows. Conditioned on the past outputs y_0, \dots, y_t , the variables x_{t+1} and y_{t+1} have a joint Gaussian distribution, with mean and covariance matrix:

$$\begin{bmatrix} \hat{x}_{t+1|t} \\ C\hat{x}_{t+1|t} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} P_{t+1|t} & P_{t+1|t}C^T \\ CP_{t+1|t} & CP_{t+1|t}C^T + R \end{bmatrix} \quad (15.20)$$

This leaves us in a situation which is familiar to us from factor analysis. Making reference to Figure 15.2(b), we have a Gaussian graphical model fragment in which we wish to reverse the arrow; that is, we wish to compute the conditional distribution of x_{t+1} given y_{t+1} , where x_{t+1} and y_{t+1} have a joint Gaussian distribution. The only difference in the current situation is that the joint distribution is itself a conditional distribution, conditioned on the past outputs y_0, \dots, y_t .

Utilizing Eq. 13.26 and 13.27 from Chapter 13, we obtain:

$$\hat{x}_{t+1|t+1} = \hat{x}_{t+1|t} + P_{t+1|t}C^T(CP_{t+1|t}C^T + R)^{-1}(y_{t+1} - C\hat{x}_{t+1|t}) \quad (15.21)$$

$$P_{t+1|t+1} = P_{t+1|t} - P_{t+1|t}C^T(CP_{t+1|t}C^T + R)^{-1}CP_{t+1|t}. \quad (15.22)$$

We summarize the filtering equations that we have obtained. At time t we assume that we have available the mean estimate $\hat{x}_{t|t}$ and the covariance estimate $P_{t|t}$. Based on these estimates we calculate $\hat{x}_{t+1|t+1}$ and $P_{t+1|t+1}$ recursively as follows:

$$\hat{x}_{t+1|t} = A\hat{x}_{t|t} \quad (15.23)$$

$$P_{t+1|t} = AP_{t|t}A^T + GQG^T \quad (15.24)$$

$$\hat{x}_{t+1|t+1} = \hat{x}_{t+1|t} + P_{t+1|t}C^T(CP_{t+1|t}C^T + R)^{-1}(y_{t+1} - C\hat{x}_{t+1|t}) \quad (15.25)$$

$$P_{t+1|t+1} = P_{t+1|t} - P_{t+1|t}C^T(CP_{t+1|t}C^T + R)^{-1}CP_{t+1|t}. \quad (15.26)$$

These recursions constitute the Kalman filter. They are initialized with $\hat{x}_{0|-1} = 0$ and $P_{0|-1} = P_0$.

The update in Eq. 15.25 is often summarized in more a compact form by defining the *Kalman gain matrix*:

$$K_{t+1} \triangleq P_{t+1|t}C^T(CP_{t+1|t}C^T + R)^{-1}. \quad (15.27)$$

Using this notation we have:

$$\hat{x}_{t+1|t+1} = \hat{x}_{t+1|t} + K_{t+1}(y_{t+1} - C\hat{x}_{t+1|t}). \quad (15.28)$$

Moreover, we can use the matrix inversion formulas to write the gain matrix in an alternative form. In particular, using Eq. 13.17 and Eq. 13.18, we obtain:

$$K_{t+1} = P_{t+1|t} C^T (C P_{t+1|t} C^T + R)^{-1} \quad (15.29)$$

$$= (P_{t+1|t}^{-1} + C^T R C)^{-1} C^T R^{-1} \quad (15.30)$$

$$= (P_{t+1|t} + P_{t+1|t} C^T (C P_{t+1|t} C^T + R)^{-1} C P_{t+1|t}) C^T R^{-1} \quad (15.31)$$

$$= P_{t+1|t+1} C^T R^{-1}, \quad (15.32)$$

which expresses the gain matrix in terms of the updated matrix $P_{t+1|t+1}$.

15.5 Interpretation and relationship to LMS

The Kalman filtering equations have an appealing interpretation as an error-correcting algorithm. Let us write a single equation for the update of the mean by combining Eq. 15.23 and Eq. 15.25:

$$\hat{x}_{t+1|t+1} = A\hat{x}_{t|t} + K_{t+1}(y_{t+1} - CA\hat{x}_{t|t}). \quad (15.33)$$

Eq. 15.33 describes an error-correcting algorithm for estimating the state x_{t+1} . In particular, at time t , our best estimate of the state x_t is $\hat{x}_{t|t}$. Imitating the dynamical equation we produce an estimate $A\hat{x}_{t|t}$ of the state at time $t+1$. This estimate is then corrected based on the observation y_{t+1} ; in particular, we adjust our estimate by a term $(y_{t+1} - CA\hat{x}_{t|t})$ that is proportional to the error between the observed output and our prediction of the output.

This error-correction procedure is reminiscent of the LMS algorithm. To clarify the relationship, consider a simplified situation in which the matrix A is the identity matrix and the noise term w_t is zero. In this case, the “dynamical equation” $x_{t+1} = x_t + Gw_t$ reduces to the statement that the “state” is a constant. Let θ denote this constant. Furthermore, let the matrix C in Eq. 15.2 be replaced by the (time-varying) vector x_t^T (as in Section XXX). In this case, Eq. 15.2 reduces to:

$$y_t = x_t^T \theta + v_t. \quad (15.34)$$

We are back in the world of linear regression, in which the outputs y_t are a sequence of iid observations that provide information about the parameter vector θ . In this case the Kalman filtering equation becomes:

$$\hat{\theta}_{t+1} = \hat{\theta}_t + P_{t+1} R^{-1} (y_{t+1} - x_t^T \hat{\theta}) x_t, \quad (15.35)$$

where we have used the fact that R^{-1} is a scalar and have dropped the unnecessary second time subscript on the P matrix.

We have derived an equation which, when combined with the update for P_{t+1} , is referred to as the *recursive least squares (RLS) algorithm*. RLS is a special case of the Kalman filter and, as such, provides the optimal least-squares estimate of θ based on data y_t up to and including time t .

If we proceed further and approximate the matrix $P_{t+1} R^{-1}$ with a scalar multiplier μ , Eq. 15.35 reduces to the LMS algorithm (Eq. 6.6). Thus LMS can be viewed as an approximation to the Kalman filter. We have gained in simplicity—no longer needing to carry forward a covariance

matrix—but we have lost in accuracy. The LMS algorithm requires multiple passes through a data set to converge to the least-squares estimate of the parameter; the Kalman filter converges in a single pass.

Although this connection between the Kalman filter and the LMS algorithm is a interesting and useful relationship to be aware of, the approximation of $P_{t+1}R^{-1}$ by a scalar multiplier receives no particular justification within the theory of Kalman filtering. Rather it requires a different theoretical framework (that of stochastic approximation) for its justification.

15.6 Information filter

Recall that the multivariate Gaussian distribution can be described using either the moment parameterization or the canonical parameterization. Our derivation of the Kalman filter used the moment parameterization of the Gaussian, but it is also of interest to define a filtering algorithm in terms of the canonical parameterization. The result is an algorithm known as an *information filter*.

We can derive the information filter either from first principles or by transforming the equations that we have already obtained. We pursue the former approach in Chapter 18, where we reconsider the SSM from the perspective of the junction tree framework. In the current section we pursue the latter approach. This is essentially an exercise in the use of the matrix inversion lemmas (Eq. 13.17 and Eq. 13.18).

Recall from Chapter 13 that the canonical parameters of a Gaussian distribution can be obtained from the moment parameters by the following transformation (cf. Eq. 13.5): $\Lambda = \Sigma^{-1}$ and $\xi = \Sigma^{-1}\mu$. Define $\hat{\xi}_{t|t-1}$ and $S_{t|t-1}$ to be the canonical parameters of the distribution of x_t conditioned on $y_{1,\dots,t-1}$ and let $\hat{\xi}_{t|t}$ and $S_{t|t}$ to be the canonical parameters of the distribution of x_t conditioned on $y_{1,\dots,t}$. We obtain a set of recursions for these quantities by substituting from Eqs. 15.23 to 15.26.

Let us begin with the inverse covariance matrices. Defining $H \triangleq GQG^T$ to simplify the notation, we have:

$$S_{t+1|t} = P_{t+1|t}^{-1} \quad (15.36)$$

$$= (AP_{t|t}A^T + H)^{-1} \quad (15.37)$$

$$= H^{-1} - H^{-1}A(P_{t|t}^{-1} + A^TH^{-1}A)^{-1}A^TH^{-1} \quad (15.38)$$

$$= H^{-1} - H^{-1}A(S_{t|t} + A^TH^{-1}A)^{-1}A^TH^{-1}. \quad (15.39)$$

A further application of the matrix inversion lemma yields:

$$S_{t+1|t+1} = P_{t+1|t+1}^{-1} \quad (15.40)$$

$$= (P_{t+1|t} - P_{t+1|t}C^T(CP_{t+1|t}C^T + R)^{-1}CP_{t+1|t})^{-1} \quad (15.41)$$

$$= P_{t+1|t}^{-1} + C^TR^{-1}C \quad (15.42)$$

$$= S_{t+1|t} + C^TR^{-1}C. \quad (15.43)$$

Turning now to the ξ parameters, we have:

$$\hat{\xi}_{t+1|t} = P_{t+1|t}^{-1} \hat{x}_{t+1|t} \quad (15.44)$$

$$= P_{t+1|t}^{-1} A \hat{x}_{t|t} \quad (15.45)$$

$$= P_{t+1|t}^{-1} A P_{t|t} \hat{\xi}_{t|t} \quad (15.46)$$

$$= (A P_{t|t} A^T + H)^{-1} A P_{t|t} \hat{\xi}_{t|t} \quad (15.47)$$

$$= H^{-1} A (P_{t|t}^{-1} + A^T H^{-1} A)^{-1} \hat{\xi}_{t|t} \quad (15.48)$$

$$= H^{-1} A (S_{t|t} + A^T H^{-1} A)^{-1} \hat{\xi}_{t|t}, \quad (15.49)$$

and

$$\hat{\xi}_{t+1|t+1} = P_{t+1|t+1}^{-1} \hat{x}_{t+1|t+1} \quad (15.50)$$

$$= P_{t+1|t+1}^{-1} (\hat{x}_{t+1|t} + P_{t+1|t+1} C^T R^{-1} (y_{t+1} - C \hat{x}_{t+1|t})) \quad (15.51)$$

$$= (P_{t+1|t+1}^{-1} - C^T R^{-1} C) P_{t+1|t}^{-1} \hat{\xi}_{t+1|t} + C^T R^{-1} y_{t+1} \quad (15.52)$$

$$= (P_{t+1|t}^{-1} + C^T R^{-1} C - C^T R^{-1} C) P_{t+1|t}^{-1} \hat{\xi}_{t+1|t} + C^T R^{-1} y_{t+1} \quad (15.53)$$

$$= \hat{\xi}_{t+1|t} + C^T R^{-1} y_{t+1}. \quad (15.54)$$

We summarize the information filter equations. At time t we assume that we have available $\hat{\xi}_{t|t}$ and $S_{t|t}$. Based on these estimates we calculate $\hat{\xi}_{t+1|t+1}$ and $S_{t+1|t+1}$ recursively as follows:

$$\hat{\xi}_{t+1|t} = H^{-1} A (S_{t|t} + A^T H A)^{-1} \hat{\xi}_{t|t} \quad (15.55)$$

$$\hat{\xi}_{t+1|t+1} = \hat{\xi}_{t+1|t} + C^T R^{-1} y_{t+1} \quad (15.56)$$

$$S_{t+1|t} = H^{-1} - H^{-1} A (S_{t|t} + A^T H^{-1} A)^{-1} A^T H^{-1} \quad (15.57)$$

$$S_{t+1|t+1} = S_{t+1|t} + C^T R^{-1} C. \quad (15.58)$$

These recursions are initialized with $\hat{\xi}_{0|-1} = \bar{\xi}_0$ and $S_{0|-1} = S_0$.

The Kalman filter and the information filter are mathematically equivalent; the major practical difference between them is essentially numerical. Recall that the condition number of a matrix is the reciprocal of the condition number of its inverse; this implies that poor conditioning for one set of recursions generally implies good conditioning for the other set. A related issue concerns the initial conditions. If we are quite certain about the initial state, then we would set P_0 to zero, in which case S_0 is undefined and we would be forced to use the Kalman filter. On the other hand, if we are quite uncertain about the initial state, we would set S_0 , in which case P_0 is undefined and we would be forced to use the information filter.

15.7 Smoothing

We now turn to the issue of obtaining estimates of the state at time t based on data up to and including a later time T . As in the case of the HMM, the calculation of this state estimate requires

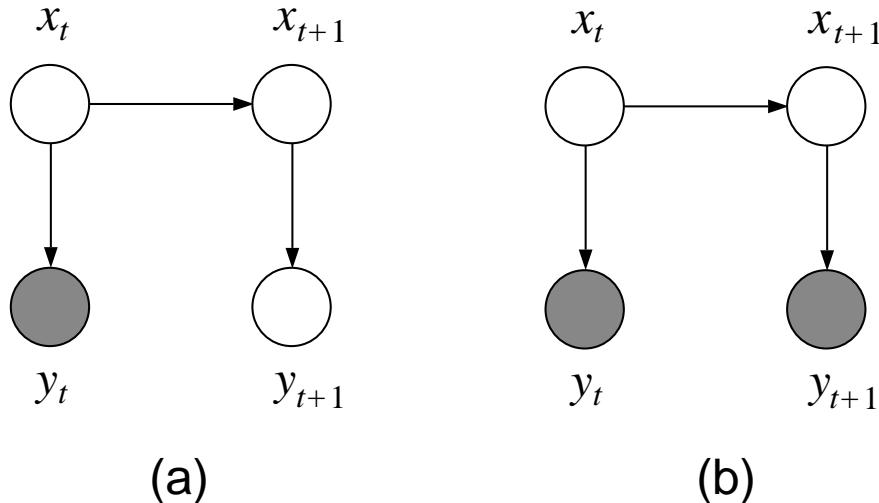


Figure 15.3: (a) A fragment of an SSM in which the observations up to and including y_t are available, and (b) the same fragment in which observations y_{t+1} to y_T are available.

us to combine a forward recursion with a backward recursion. Furthermore, we once again have the choice between an algorithm that computes backward-filtered estimates and combines them with the forward-filtered estimates (an “alpha-beta algorithm”), or an algorithm that recurses directly on the filtered-and-smoothed estimates (an “alpha-gamma algorithm”). Both kinds of algorithm are available in the literature on state-space models, but the latter approach appears to dominate (as opposed to the HMM literature, where the former approach dominates). In this section we begin with the “alpha-gamma” approach, deriving the the “Rauch-Tung-Striebel (RTS) smoothing algorithm,” and then turn to an alternative “alpha-beta” approach.

15.7.1 The Rauch-Tung-Striebel (RTS) smoother

Our approach to deriving the RTS smoothing algorithm will once again be based on the graphical model fragment shown in Figure 15.2, which we reproduce in Figure 15.3. We begin by writing down the joint distribution of x_t and x_{t+1} , conditional on y_0, \dots, y_t . Recall that $\hat{x}_{t+1|t} = A\hat{x}_{t|t}$, which implies:

$$E[(x_t - \hat{x}_{t|t})(x_{t+1} - \hat{x}_{t+1|t})^T | y_0, \dots, y_t] = P_{t|t} A^T. \quad (15.59)$$

Thus our distribution has the following mean and covariance matrix:

$$\begin{bmatrix} \hat{x}_{t|t} \\ \hat{x}_{t+1|t} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} P_{t|t} & P_{t|t}A^T \\ AP_{t|t} & P_{t+1|t} \end{bmatrix}, \quad (15.60)$$

where all of the quantities are available to us after a forward Kalman filtering pass.

We now introduce a “backwards” computation. In particular, we condition on x_{t+1} and calculate the probability of x_t , still conditioning on y_0, \dots, y_t . Using the by-now familiar Gaussian

conditioning rule (Eq. 13.26), we obtain:

$$E[x_t|x_{t+1}, y_0, \dots, y_t] = \hat{x}_{t|t} + P_{t|t} A^T P_{t+1|t}^{-1} (x_{t+1} - \hat{x}_{t+1|t}) \quad (15.61)$$

$$= \hat{x}_{t|t} + L_t (x_{t+1} - \hat{x}_{t+1|t}), \quad (15.62)$$

where we have introduced the notation $L_t \triangleq P_{t|t} A^T P_{t+1|t}^{-1}$, and

$$\text{Var}[x_t|x_{t+1}, y_0, \dots, y_t] = P_{t|t} - P_{t|t} A^T P_{t+1|t}^{-1} A P_{t|t} \quad (15.63)$$

$$= P_{t|t} - L_t P_{t+1|t} L_t^T. \quad (15.64)$$

The purpose of conditioning on x_{t+1} is to render x_t independent of the future observations y_{t+1}, \dots, y_T . That is, we can use conditional independence to write:

$$E[x_t|x_{t+1}, y_0, \dots, y_T] = E[x_t|x_{t+1}, y_0, \dots, y_t] \quad (15.65)$$

$$= \hat{x}_{t|t} + L_t (x_{t+1} - \hat{x}_{t+1|t}) \quad (15.66)$$

and

$$\text{Var}[x_t|x_{t+1}, y_0, \dots, y_T] = \text{Var}[x_t|x_{t+1}, y_0, \dots, y_t] \quad (15.67)$$

$$= P_{t|t} - L_t P_{t+1|t} L_t^T. \quad (15.68)$$

The quantities on the left-hand side of these equations are almost what we want; indeed, if we could drop x_{t+1} we would have the desired filtered-and-smoothed quantities.

The remainder of the derivation is an exercise in conditional expectation. Recall from Appendix XXX the following fundamental facts about conditional expectations:

$$E[X|Z] = E[E[X|Y, Z]|Z] \quad (15.69)$$

and

$$\text{Var}[X|Z] = \text{Var}[E[X|Y, Z]|Z] + E[\text{Var}[X|Y, Z]|Z] \quad (15.70)$$

which show us how to compute unconditional expectations using conditional expectations. We will substitute x_t for X , x_{t+1} for Y , and y_0, \dots, y_T for Z in these equations.

Beginning with Eq. 15.66, we take the conditional expectation on both sides, conditioning with respect to y_0, \dots, y_T :

$$\hat{x}_{t|T} \triangleq E[x_t|y_0, \dots, y_T] \quad (15.71)$$

$$= E[E[x_t|x_{t+1}, y_0, \dots, y_T]|y_0, \dots, y_T] \quad (15.72)$$

$$= E[\hat{x}_{t|t} + L_t (x_{t+1} - \hat{x}_{t+1|t})|y_0, \dots, y_T] \quad (15.73)$$

$$= \hat{x}_{t|t} + L_t (x_{t+1|T} - \hat{x}_{t+1|t}), \quad (15.74)$$

where we have used the fact that all of the quantities in Eq. 15.74 other than x_{t+1} are constants when we condition on y_0, \dots, y_T .

Eq. 15.74 is the basic update equation in the RTS smoothing algorithm. We see that a estimate of x_t based on all of the data can be obtained by correcting the filtered estimate $\hat{x}_{t|t}$ by an error term composed of a smoothed estimate of x_{t+1} and the filtered estimate $\hat{x}_{t+1|t}$. The gain matrix L_t is a quantity that depends only on matrices computed during the forward pass.

We now work on the conditional variance equation (Eq. 15.68). Using Eq. 15.70, we have:

$$P_{t|T} \triangleq \text{Var}[x_t|y_0, \dots, y_T] \quad (15.75)$$

$$= \text{Var}[E[x_t|x_{t+1}, y_0, \dots, y_T]|y_0, \dots, y_T] + E[\text{Var}[x_t|x_{t+1}, y_0, \dots, y_T]|y_0, \dots, y_T] \quad (15.76)$$

$$= \text{Var}[\hat{x}_{t|t} + L_t(x_{t+1} - \hat{x}_{t+1|t})|y_0, \dots, y_T] + E[P_{t|t} - L_t P_{t+1|t} L_t^T|y_0, \dots, y_T] \quad (15.77)$$

$$= L_t \text{Var}[(x_{t+1} - \hat{x}_{t+1|t})|y_0, \dots, y_T] L_t^T + P_{t|t} - L_t P_{t+1|t} L_t^T \quad (15.78)$$

$$= L_t \text{Var}[x_{t+1}|y_0, \dots, y_T] L_t^T + P_{t|t} - L_t P_{t+1|t} L_t^T \quad (15.79)$$

$$= L_t P_{t+1|T} L_t^T + P_{t|t} - L_t P_{t+1|t} L_t^T \quad (15.80)$$

$$= P_{t|t} + L_t(P_{t+1|T} - P_{t+1|t})L_t^T, \quad (15.81)$$

where at several junctures we have used the fact that expectations taken with respect to y_0, \dots, y_T are constant when conditioning with respect to the larger conditioning set y_0, \dots, y_T .

We summarize the RTS smoothing algorithm. Based on the quantities $\hat{x}_{t+1|t}$, $P_{t|t}$ and $P_{t+1|t}^{-1}$ from the filtering algorithm, we compute:

$$\hat{x}_{t|T} = \hat{x}_{t|t} + L_t(x_{t+1|T} - \hat{x}_{t+1|t}), \quad (15.82)$$

$$P_{t|T} = P_{t|t} + L_t(P_{t+1|T} - P_{t+1|t})J_t^T, \quad (15.83)$$

where $L_t \triangleq P_{t|t} A^T P_{t+1|t}^{-1}$. The algorithm is initialized by using $\hat{x}_{T|T}$ and $P_{T|T}$ from the filtering pass.

15.7.2 The two-filter smoother

In this section we describe an alternative approach to smoothing in the SSM which is the analog of the alpha-beta algorithm for the HMM. In this approach, known as the “two-filter algorithm,” the idea is to combine the “forward” conditional probability $P(x_t|y_0, \dots, y_t)$ with the “backward” conditional probability $P(x_t|y_{t+1}, \dots, y_T)$. Note that the latter quantity, like the former quantity, is a “filtered estimate”; that is, a conditional probability of the state given a (partial) output sequence. This differs from the traditional beta variable in the HMM, which is the conditional probability of the output sequence given the state. Clearly we can move from one to the other, however, by multiplying or dividing by the unconditional probability of the state, $P(x_t)$, which is available via the Lyapunov equation. Thus, the difference is minor and it is appropriate to think of the two-filter algorithm as the analog of the alpha-beta algorithm.

Given that we want filtered estimates in the backward direction, a simple approach to deriving the backward algorithm is to “invert the dynamics” and apply a forward filtering algorithm to the inverted dynamics. In graphical model terms, we invert the arrows in the graph. This is in itself a useful exercise.

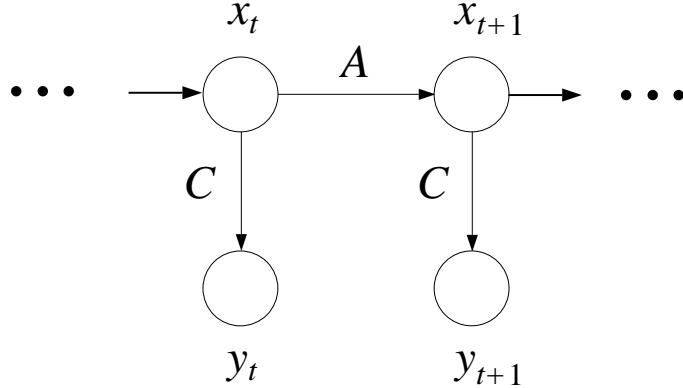


Figure 15.4: (a) A fragment of an SSM with no observations.

In this section we assume that the matrix A is invertible and make use of A^{-1} in our derivation of the algorithm. In fact this assumption is not necessary, and a lookahead at the algorithm that we derive shows that A^{-1} does not appear. (In Chapter 18 we present an alternative derivation of the algorithm from the point of view of the junction tree algorithm, and in that derivation we do not make use of A^{-1}).

The naive approach to inverting the dynamical equation is to simply write:

$$x_t = A^{-1}x_{t+1} - A^{-1}Gw_t, \quad (15.84)$$

and let t run backwards in time. This approach is not viable, however, because w_t is not independent of the “past” values of the state; i.e., x_{t+1}, \dots, x_T . Indeed, these states are all a function of w_t . Thus one of the assumptions that we used in deriving the Kalman filter is not valid and we cannot simply apply the Kalman filter to Eq. 15.84.

To obtain a more useful inverse of the dynamics, consider the graphical model fragment shown in Figure 15.4. The forward dynamics yields a joint probability distribution on (x_t, x_{t+1}) characterized by the Lyapunov equation $\Sigma_{t+1} = A\Sigma_t A^T + GQG^T$. Indeed the covariance matrix of (x_t, x_{t+1}) is given by:

$$\begin{bmatrix} \Sigma_t & \Sigma_t A^T \\ A\Sigma_t & A\Sigma_t A^T + GQG^T \end{bmatrix} \quad (15.85)$$

We can invert the relationship between x_t and x_{t+1} by solving for Σ_t in terms of Σ_{t+1} and rewriting the covariance matrix in terms of Σ_{t+1} . Thus:

$$\Sigma_t = A^{-1}\Sigma_{t+1}A^{-T} - A^{-1}GQG^TA^{-T}, \quad (15.86)$$

where we assume that A is invertible.² This equation also implies:

$$A\Sigma_t = \Sigma_{t+1}A^{-T} - GQG^TA^{-T}, \quad (15.87)$$

²In fact this assumption is not necessary, see Exercise XXX.

and we can rewrite the covariance matrix as follows:

$$\begin{bmatrix} A^{-1}\Sigma_{t+1}A^{-T} - A^{-1}GQG^TA^{-T} & A^{-1}\Sigma_{t+1} - A^{-1}GQG^T \\ \Sigma_{t+1}A^{-T} - GQG^TA^{-T} & \Sigma_{t+1} \end{bmatrix} \quad (15.88)$$

Noting that the upper-right-hand corner of this matrix can be written as $A^{-1}(I - A^{-1}GQG^T\Sigma_{t+1}^{-1})\Sigma_{t+1}$, we see that if we define:

$$\tilde{A} = A^{-1}(I - A^{-1}GQG^T\Sigma_{t+1}^{-1}) \quad (15.89)$$

then we obtain $\tilde{A}\Sigma_{t+1}$ and $\Sigma_{t+1}\tilde{A}^T$ in the corners of the matrix, and the matrix begins to take the form of a forward covariance matrix. This suggests that we define the inverse dynamics via:

$$x_t = \tilde{A}x_{t+1} + \tilde{G}\tilde{w}_{t+1}, \quad (15.90)$$

with \tilde{G} and \tilde{w}_t chosen appropriately so as to match the forward dynamics (Eq. 15.1). Indeed, choosing

$$\tilde{G} = -A^{-1}G \quad (15.91)$$

$$\tilde{w}_{t+1} = w_t - QG^T\Sigma_{t+1}^{-1}x_{t+1} \quad (15.92)$$

Eq. 15.90 matches Eq. 15.1. Moreover, we have:

$$\tilde{Q} \triangleq E[\tilde{w}_{t+1}\tilde{w}_{t+1}^T] = Q - QG^T\Sigma_{t+1}^{-1}GQ, \quad (15.93)$$

and substituting Eqs. 15.89, 15.92 and 15.93 in the backward Lyapunov equation:

$$\Sigma_t = \tilde{A}\Sigma_{t+1}\tilde{A}^T + \tilde{G}\tilde{Q}\tilde{G}^T \quad (15.94)$$

we recover the forward Lyapunov equation.

Finally, it can also be verified (see Exercise XXX) that \tilde{w}_{t+1} is independent of the “past” values of the state x_{t+1}, \dots, x_T .

We have therefore succeeded in obtaining a version of the inverse dynamics to which standard filtering algorithms can be applied. If we use the canonical parameterization (i.e., the information filter in Eqs. 15.39, 15.43, 15.49, and 15.54), utilizing the inverse dynamical equation and noting that the output equation $y_t = Cx_t + v_t$ has not changed, we obtain:

$$S_{t|t+1} = A^T H A + \Sigma_t^{-1} - A^T H^{-1} (S_{t+1|t+1} + H^{-1} - \Sigma_{t+1}^{-1})^{-1} H^{-1} A \quad (15.95)$$

$$S_{t|t} = S_{t|t+1} + C^T R^{-1} C \quad (15.96)$$

$$\hat{\xi}_{t|t+1} = A^T H^{-1} (S_{t+1|t+1} + H^{-1} - \Sigma_{t+1}^{-1})^{-1} \hat{\xi}_{t+1|t+1} \quad (15.97)$$

$$\hat{\xi}_{t|t} = \hat{\xi}_{t|t+1} + C^T R^{-1} y_t, \quad (15.98)$$

where t and $t + 1$ have been interchanged to reflect the fact that we are filtering backward in time. This filter calculates the canonical representation of $P(x_t|y_{t+1}, \dots, y_T)$. Thus, converting to the moment representation, we have $\hat{x}_{t|t+1} = S_{t|t+1}^{-1} \hat{\xi}_{t|t+1}$ and $P_{t|t+1} = S_{t|t+1}^{-1}$.

The final issue that we must address involves the fusing of the probability distributions $P(x_t|y_0, \dots, y_t)$ and $P(x_t|y_{t+1}, \dots, y_T)$ to obtain the posterior probability $P(x_t|y_0, \dots, y_T)$. This problem is not unique to the filtering and smoothing domain, but arises in many other settings as well. It is therefore worth posing and solving the problem in full generality; this we do in the following section. Anticipating the result, we have the following fusion rule for $\hat{x}_{t|T}$, the estimate of x_t based on all of the data:

$$\hat{x}_{t|T} = P_{t|T}^{-1} (\hat{x}_{t|t} + P_{t|t+1}^{-1} \hat{x}_{t|t+1}), \quad (15.99)$$

where the covariance matrix $P_{t|T}$ is computed as follows:

$$P_{t|T} = \left(P_{t|t}^{-1} + P_{t|t+1}^{-1} - \Sigma_t^{-1} \right)^{-1}. \quad (15.100)$$

The appearance of Σ_t^{-1} in the latter equation should not be a surprise. The filtering process and the smoothing process both make use of the prior statistics on x_t ; in the latter case this is because we have inverted the dynamics. When the covariance matrices of these two processes are combined we have included the prior covariance twice. To avoid double-counting Σ_t^{-1} must be subtracted in the combination rule.

15.7.3 Fusion of Gaussian posterior probabilities

Let us consider three sets of random variables: x , z_1 and z_2 . Suppose that these variables are characterized by a multivariate Gaussian distribution and suppose moreover that z_1 and z_2 are conditionally independent given x . We wish to fuse the posteriors $P(x|z_1)$ and $P(x|z_2)$ into an overall posterior $P(x|z_1, z_2)$.

Let us assume, without loss of generality, that x , z_1 , and z_2 have zero means. Non-zero means can be subtracted away and added back at the end of the analysis.

Under the conditional independence assumption, there are three ways to represent the distribution of x , z_1 , and z_2 as a directed graphical model. The representation given in Figure 15.5(a) is particularly useful for our purposes. To parameterize the graph, we require the marginal $P(x)$, and conditionals $P(z_1|x)$ and $P(z_2|x)$. For the marginal, we endow x with a zero mean and covariance Σ . For the conditionals, recall that Gaussian conditionals are linear functions of the conditioning variable (cf. Eq. ref{eq:Gaussian-conditional-mean}). Thus we can write:

$$z_1 = M_1 x + v_1 \quad (15.101)$$

$$z_2 = M_2 x + v_2, \quad (15.102)$$

for appropriately chosen matrices M_1 and M_2 and zero-mean Gaussian variables v_1 and v_2 having covariance matrices R_1 and R_2 . Note moreover that v_1 and v_2 are independent of x and are conditionally independent of each other given x .

Let us now consider a generic linear equation $z = Mx + v$, where v is independent of x and has covariance R . To calculate the conditional expectation of x given z we first obtain the covariance matrix of the pair (x, z) :

$$\begin{bmatrix} \Sigma & \Sigma M^T \\ M\Sigma & M\Sigma M^T + R \end{bmatrix}. \quad (15.103)$$

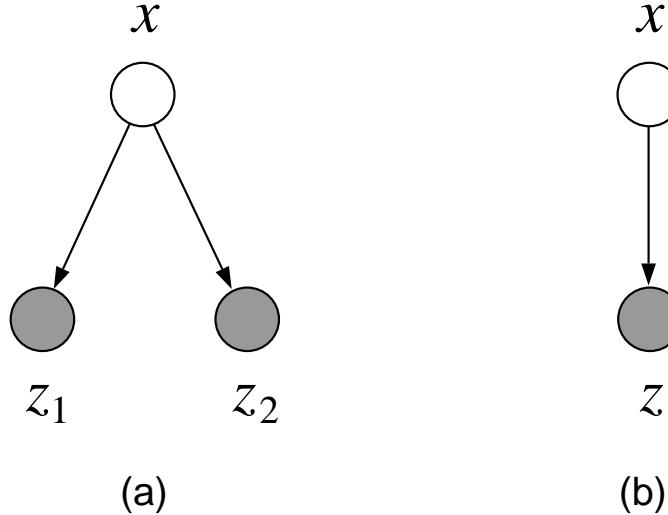


Figure 15.5: A graphical model representation of the fusion problem. (a) The observables z_1 and z_2 are assumed conditionally independent given x . The conditional probabilities of z_i are parameterized as linear functions of x with additive, independent noise terms. (b) Conjoining z_1 and z_2 into a single observable vector z .

We then apply the usual Gaussian conditioning formulas (Eqs. 13.26 and 13.27) to obtain the conditional distribution of x given z . Denoting the mean of this conditional distribution as \hat{x} and the covariance as P , we have:

$$\hat{x} = \Sigma M^T (M \Sigma M^T + R)^{-1} z \quad (15.104)$$

$$= (M^T R^{-1} M + \Sigma^{-1})^{-1} M^T R^{-1} z, \quad (15.105)$$

where we have used a matrix inversion identity (Eq. 13.18) in the second step, and:

$$P = \Sigma - \Sigma M^T (M \Sigma M^T + R)^{-1} M \Sigma \quad (15.106)$$

$$= (\Sigma^{-1} + M^T R^{-1} M)^{-1}, \quad (15.107)$$

where again we use a matrix inversion identity (Eq. 13.17) to simplify the result.

The individual conditionals of x given z_1 and z_2 are special cases of the foregoing equations. Defining $\hat{x}_i \triangleq E(x|z_i)$ and letting P_i denote the corresponding conditional covariance, we have:

$$\hat{x}_1 = (M_1^T R_1^{-1} M_1 + \Sigma^{-1})^{-1} M_1^T R_1^{-1} z_1 \quad (15.108)$$

$$\hat{x}_2 = (M_2^T R_2^{-1} M_2 + \Sigma^{-1})^{-1} M_2^T R_2^{-1} z_2, \quad (15.109)$$

and

$$P_1 = (M_1^T R_1^{-1} M_1 + \Sigma^{-1})^{-1} \quad (15.110)$$

$$P_2 = (M_2^T R_2^{-1} M_2 + \Sigma^{-1})^{-1}. \quad (15.111)$$

Now let us consider the overall posterior of x given both z_1 and z_2 . Grouping z_1 and z_2 into a single variable z (cf. Figure 15.5(b)), we can apply Eqs. 15.105 and 15.107 where:

$$M \triangleq \begin{bmatrix} M_1 \\ M_2 \end{bmatrix} \quad \text{and} \quad R \triangleq \begin{bmatrix} R_1 & 0 \\ 0 & R_2 \end{bmatrix}. \quad (15.112)$$

From these definitions we obtain:

$$\hat{x} = \left(\begin{bmatrix} M_1^T & M_2^T \end{bmatrix} \begin{bmatrix} R_1^{-1} & 0 \\ 0 & R_2^{-1} \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \end{bmatrix} + \Sigma^{-1} \right)^{-1} \begin{bmatrix} M_1^T & M_2^T \end{bmatrix} \begin{bmatrix} R_1^{-1} & 0 \\ 0 & R_2^{-1} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

$$= (M_1^T R_1^{-1} M_1 + M_2^T R_2^{-1} M_2 + \Sigma^{-1})^{-1} (M_1^T R_1^{-1} z_1 + M_2^T R_2^{-1} z_2) \quad (15.113)$$

$$= (P_1^{-1} + P_2^{-1} - \Sigma^{-1})^{-1} (P_1^{-1} \hat{x}_1 + P_2^{-1} \hat{x}_2) \quad (15.114)$$

We can similarly expand Eq. 15.107 to obtain the overall conditional covariance P :

$$P = (P_1^{-1} + P_2^{-1} - \Sigma^{-1})^{-1}, \quad (15.115)$$

thus allowing us to rewrite Eq. 15.114 as:

$$\hat{x} = P(P_1^{-1} \hat{x}_1 + P_2^{-1} \hat{x}_2). \quad (15.116)$$

Eqs. 15.116 and 15.115 are our general solution to the Gaussian fusion problem.

Let us relate these results back to the two-filter smoothing problem. We collect the observations up to and including time t into a single “past” vector $z_1 \triangleq (y_0, \dots, y_t)$, and collect the “future” observations into a single vector $z_2 \triangleq (y_{t+1}, \dots, y_T)$. Let $x \triangleq x_t$. These definitions fit the problem specification of the current section; in particular (x, z_1, z_2) are characterized by a multivariate Gaussian distribution (a marginal of the larger Gaussian distribution that includes the other state variables), and moreover z_1 and z_2 are independent given x . The estimate $\hat{x}_{t|t}$ is the conditional expectation of x given z_1 , and must therefore have the form in Eq. 15.109, for matrices M_1 and R_1 that we do not bother to calculate. Similarly $\hat{x}_{t|t+1}$ must be of the form in Eq. 15.109, and the conditional covariances $P_{t|t}$ and $P_{t|t+1}$ must have the form of Eqs. 15.111 and 15.111. Substituting into Eqs. 15.116 and 15.115 we obtain the fusion rules at the end of the previous section (Eqs. 15.99 and 15.100).

15.8 Parameter estimation

We follow the by now familiar recipe for developing an EM algorithm for parameter estimation for the SSM. We write out the expected complete log likelihood, identify the expected sufficient statistics, solve for maximum likelihood estimates in terms of these expected sufficient statistics. This latter problem is simply linear regression.

[Section not yet finished].

15.9 Historical remarks and bibliography

Chapter 17

The Junction Tree Algorithm

In earlier chapters we have presented a number of examples of inferential calculations in graphical models. The general problem has been to calculate the conditional probability of a node or a set of nodes, given the observed values of another set of nodes. In the case of mixture models and factor analysis models the problem was to calculate the conditional probabilities of the latent variables given the observed data, and the solution was a rather straightforward application of Bayes rule. In the case of the HMM and the state-space model we saw a somewhat more complex inference problem involving dependencies between nodes arranged in a sequence. The solution was again an application of Bayes rule, but it was necessary to find recursions that allowed the inference problem to be solved efficiently. The Markov properties of the underlying graphical model provided the formal machinery to justify these recursions.

In the current chapter we present a general approach to inference that makes systematic use of the Markov properties of graphical models. All of the examples that we have treated until now emerge as special cases; moreover, the recursions that we worked out rather painstakingly in each individual case can now be derived more systematically. The general idea is to use the Markov properties of graphical models to find ways to decompose a general probabilistic calculation into a linked set of local computations. The key to this approach is an appropriate definition of “local.”

Chapter 3 presented a simple elimination algorithm (`ELIMINATION`) for inference on directed or undirected graphs. As `ELIMINATION` runs it creates dependencies between nodes, in effect redefining the “locality” relationships in the graph. To develop a deeper understanding of probabilistic inference, it proves helpful to abstract away from the specific process of elimination and to focus on this general notion of locality. In effect we shift our focus from the *process* of inference to the *data structures* that underly inference. We find that a particular data structure—the *junction tree*—emerges from these considerations. The junction tree makes explicit the important (and beautiful) relationship between graph-theoretic locality and efficient probabilistic inference.

Although we present specific algorithms for probabilistic inference in this chapter, it is important to emphasize at the outset that our goal is less that of providing specific recipes as it is of providing an understanding of the key general concepts that underly inference. Thus, while we will describe concrete algorithms (the “Hugin algorithm,” the “Shafer-Shenoy algorithm,” and the “Lauritzen-Spiegelhalter algorithm”), we view all of these algorithms as instances of a general algorithmic

framework that we will refer to generically as the *junction tree algorithm*. Understanding the general framework makes it easy to see how various specific algorithms arise and how they interrelate. Moreover, an important bonus of developing the general junction tree framework is the realization that probabilistic inference is itself an instance of a more general class of problems, all of which involve factorized potentials on graphs, and all of which can be solved using suitable variations on the junction tree theme. We discuss some instances of this more general class at the end of the chapter.

We begin by returning to the elimination algorithm from Chapter 3, stripping away some of its inessential details, and aiming to overcome some of its deficiencies.

17.1 From elimination to the junction tree

In Figure 17.1(a) we show the graph that served as a running example in Chapter 3. The factored form of the joint probability distribution for this graph is as follows:

$$p(x_1, x_2, \dots, x_6) = p(x_1)p(x_2 | x_1)p(x_3 | x_1)p(x_4 | x_2)p(x_5 | x_3)p(x_6 | x_2, x_5). \quad (17.1)$$

As in Chapter 3 we will use the elimination ordering $(X_6, X_5, X_4, X_3, X_2, X_1)$ in our examples.

Each factor in Eq. (17.1) expresses a dependency among one or more variables. Forming summands during a run of the elimination algorithm creates additional dependencies—for example, summing over x_6 creates an intermediate factor that is a function of x_2 and x_5 . The *elimination cliques* associated with an elimination ordering can be viewed as an explicit record of these dependencies. Recall that we can abstract away from probabilistic inference and view these elimination cliques as being formed by a purely graph-theoretic procedure (called `UNDIRECTED-GRAPHELIMINATE` in Chapter 3) in which we link all of the neighbors of a given node (thus forming a clique), and remove the node from the graph. In particular, for the elimination ordering $(X_6, X_5, X_4, X_3, X_2, X_1)$, the elimination cliques are as shown in Figure ??(b). While the elimination algorithm `ELIMINATION` does not explicitly form these cliques, the graph-theoretic operation of forming elimination cliques parallels the algebraic operation of marginalizing over a node, and neatly summarizes the graphical consequences of marginalization.

The elimination algorithm is “query-oriented.” That is, the algorithm yields the marginal or conditional probability of a given query node—the last node in the elimination ordering. Intermediate factors that are created along the way are discarded. While in some cases this is what we want, in many cases it is not. Consider in particular the chain-structured graphical model associated with the HMM or the state-space model. To calculate the posterior probability of any particular node we can eliminate forward and backward until we arrive at the node. In doing so we create a number of intermediate factors. Many of these same intermediate factors can be used in calculating the posterior probability of other nodes. Clearly we wish to avoid recomputing such factors, as we would do in a naive application of elimination. We also need to know which intermediate factors are needed for which posterior probabilities and how to combine factors—in essence we need a calculus for the intermediate factors. The elimination algorithm provides us with little help in this regard.

As a first step in moving beyond the elimination algorithm we need to allocate data structures—“permanent storage”—to the intermediate factors. Each such factor is associated with one of the

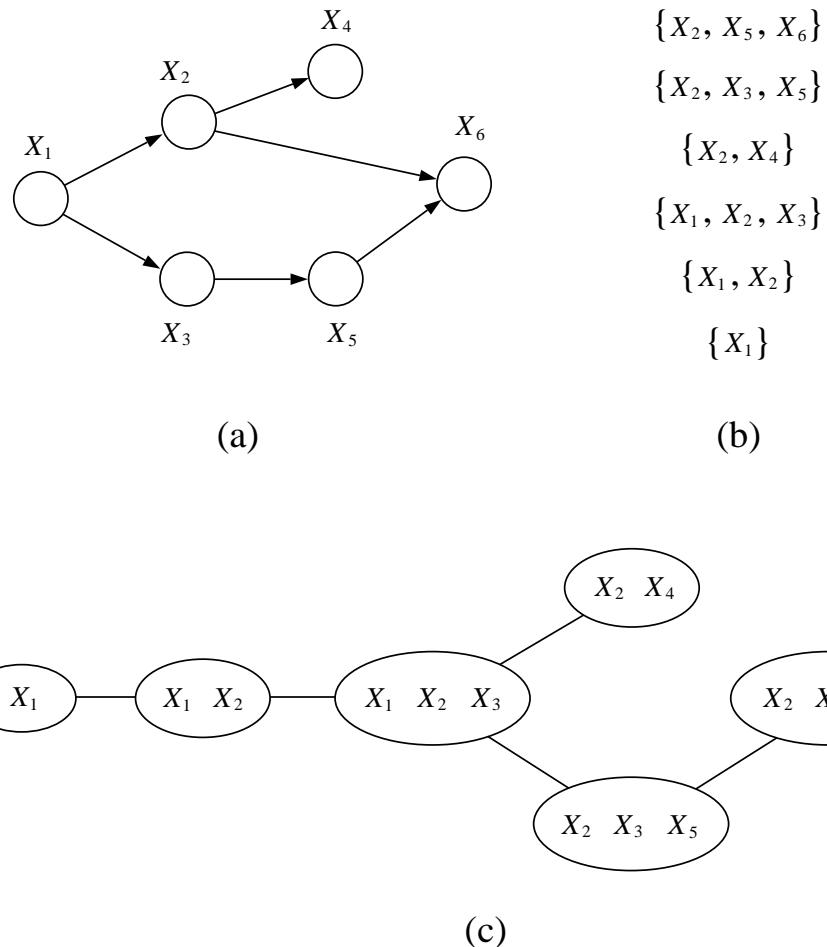


Figure 17.1: (a) The six-node example from Chapter 3. (b) The elimination clique created from a run of the elimination algorithm using the ordering $(X_6, X_5, X_4, X_3, X_2, X_1)$. (c) The elimination cliques arranged into a clique tree.

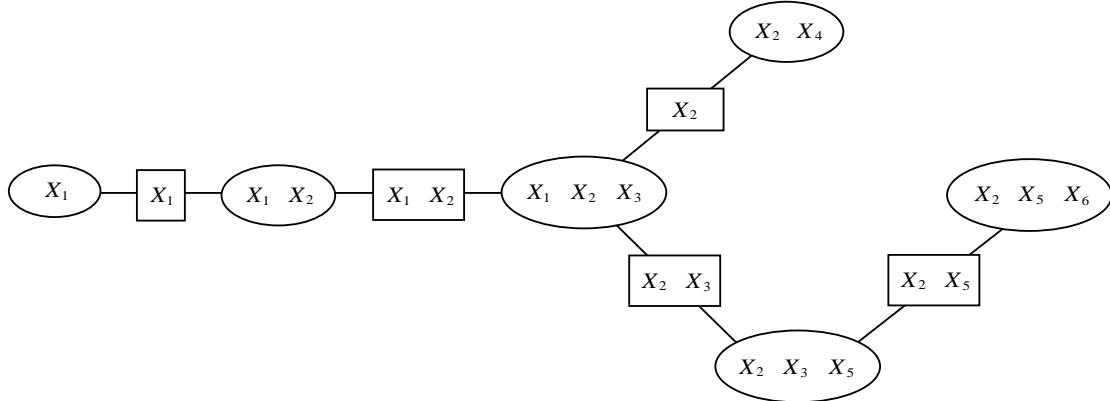


Figure 17.2: A clique tree annotated with separator sets.

elimination cliques in Figure 17.1(b). We can therefore view the nodes in this figure as representations of the storage that we need if we are to record the intermediate factors created during a run of the elimination algorithm.

While a list of the elimination cliques reveals some of the structure associated with the elimination algorithm, there is additional structure that is worth noting. In particular, as we have seen, summing over a variable produces an intermediate factor that subsequently appears in the summand associated with a later variable. For example, summing over x_5 creates an intermediate factor that refers to x_3 and thus appears in the summand when we subsequently sum over x_3 . If we view the nodes in Figure 17.1(b) as storage sites, and if we view the operation of summing as operating on the data stored at these sites, then it is natural to try to represent the transfer of information between these sites. For example, the sum over x_3 requires the factor created at the x_5 site, and we therefore need to transfer this factor between the site corresponding to the elimination of x_5 —the elimination clique $\{X_2, X_3, X_5\}$ —and the site corresponding to the elimination of x_3 —the elimination clique $\{X_1, X_2, X_3\}$. As shown in Figure 17.1(c), we can capture this flow of information by drawing an edge between these elimination cliques.

The graphical object in Figure 2.1(c) is a *clique tree*—a singly-connected graph in which the nodes are the cliques of an underlying graph. Every run of the elimination algorithm can be viewed as implicitly creating a clique tree—the clique tree can be viewed in essence as an “execution trace” of the algorithm. What we are groping towards, however, is an algorithm that goes beyond the elimination framework by explicitly representing a clique tree as a data structure. The nodes in such a clique tree will store intermediate factors, allowing these factors to be reused in multiple queries. Information will flow around the clique tree in multiple directions.

In Figure 17.2 we annotate the clique tree with some additional structure that will prove to be useful. Between each linked pair of cliques we introduce a *separator set*—the intersection of the corresponding cliques. The separator sets are themselves cliques, being the intersection of cliques. These sets provide an explicit representation of the variables referred to by the intermediate factors that pass between cliques. Consider, for example, the intermediate factor created at the clique $\{X_2, X_3, X_5\}$. Summing over x_5 creates a factor that is a function of x_2 and x_3 , and this factor

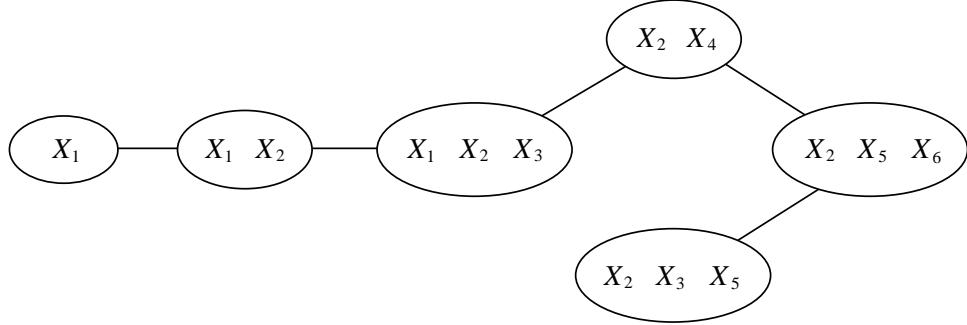


Figure 17.3: A clique tree that does not possess the junction tree property. Note in particular that the cliques containing the node X_3 do not form a connected subtree.

is sent to the clique $\{X_1, X_2, X_3\}$, where we subsequently sum over x_3 . The separator set on the link between these cliques contains the nodes $\{X_2, X_3\}$, and thus explicitly represents the domain of the intermediate factor transferred between the cliques.

Not all clique trees are created equal. In particular, the clique tree in Figure 2.1(c) has some special properties. Note that the index “2” appears in five different nodes in the figure, and that these five nodes are connected—they form a *connected subtree*. Moreover, this is true of all of the other node indices. This interesting and important property is known as the *junction tree property*. Not all clique trees possess the junction tree property; for example, the tree in Figure 17.3 does not possess the junction tree property. As we will see in the remainder of the chapter, understanding the junction tree property is the key to a general understanding of probabilistic inference.

17.2 Potentials

With the discussion in the previous section as background, we embark on a general discussion of the junction tree algorithm. We will be focusing on a particular variant of the general junction tree algorithm known as the “Hugin algorithm,” and will discuss other variations in later sections and in the exercises.

Let $G = (V, E)$ denote a directed or undirected graph with vertices V and edges E . Let \mathcal{C} denote a set of *cliques* of G ; i.e., \mathcal{C} is a set of completely connected subsets of V . We generally require these subsets to be maximal, so that no member of \mathcal{C} is a subset of another member of \mathcal{C} . However, at the cost of a bit of redundancy it is at times convenient to allow such proper subsets to appear in \mathcal{C} .

Let X be a random vector indexed by the vertices V . Recall that we allow subsets of the vertex set V to be used as indices; thus, corresponding to each clique $C \in \mathcal{C}$, we have a set of random variables X_C , with realizations x_C . The number of such realizations is the product of the number of realizations of each individual random variable X_u , for $u \in C$.

Associated with each $C \in \mathcal{C}$ we define a *potential* $\psi_C(x_C)$, a nonnegative function on the realizations x_C . In general there are no constraints on the potential functions other than nonnegativity. Note in particular that the sets C can overlap, and we make no “consistency” requirements on the

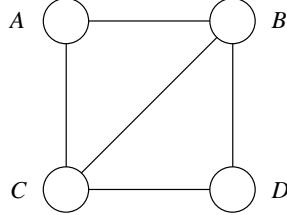


Figure 17.4: A four-node model which we assume is parameterized with pairwise potentials: ψ_{AB} , ψ_{AC} , ψ_{BC} , ψ_{BD} , and ψ_{CD} .

overlap.

We now define a joint probability distribution on X as the normalized product of potential functions:

$$p(x) \triangleq \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_C). \quad (17.2)$$

This is of course the same definition as that used for undirected graphs. Note, however, a subtle but important change in focus—in the current section we view the set of subsets \mathcal{C} as an explicit data structure, with the underlying graph in the background. Technically, our data structure is a *hypergraph*—a set of subsets—with Eq. (17.2) defining the joint probability distribution associated with the hypergraph.

There are problems in which it is natural to pose the problem directly in terms of factored potentials on sets of subsets, without focusing on an underlying graph. Most commonly, however, the potentials on the hypergraph are *initialized* from those of an underlying graph. Let us consider how this initialization process works for both undirected and directed graphs.

Undirected graphs come endowed with potential functions on cliques, and if these cliques are the same as the set of subsets \mathcal{C} , then the initialization problem is vacuous; we simply define $\psi_C(x_C)$ to be the corresponding potential from the underlying graph. In general, however, these sets are not the same. In particular, we generally include only the maximal cliques in the set \mathcal{C} . If the parameterization of the underlying undirected graph is restricted to cliques that are proper subsets of the maximal cliques of the graph, as is often the case, then we have a many-to-one mapping from parameterized cliques to \mathcal{C} . Consider, for example, the undirected graphical model in Figure 17.4, where we assume that the model is parameterized via pairwise potentials. The maximal cliques of the graph are, however, triplets of nodes. In such a situation, the potentials on maximal cliques in Eq. (17.2) are formed as the product of potentials from the underlying graph. Thus, in our example, we define ψ_{ABC} to be the product $\psi_{AB}\psi_{AC}$, while we define ψ_{BCD} to be the product $\psi_{BC}\psi_{BD}\psi_{CD}$. Note that ψ_{BC} can be associated with either triple; we have arbitrarily assigned it to ψ_{BCD} . In general each potential ψ_D on the underlying graph is assigned to one and only one ψ_C on the hypergraph, where $D \subset C$. If we assume that \mathcal{C} includes the maximal cliques, then this can always be done.

Having assigned each underlying potential to one and only one ψ_C , the product in Eq. (17.2) is a faithful representation of the joint probability from the underlying graph.

Similar issues arise when we initialize a set of clique potentials from an underlying directed

$\text{MORALIZE}(G)$

```

for each node  $X_i$  in  $I$ 
    connect all of the parents of  $X_i$ 
end drop the orientation of all edges
return  $G$ 

```

Figure 17.5: An algorithm to moralize a directed graph.

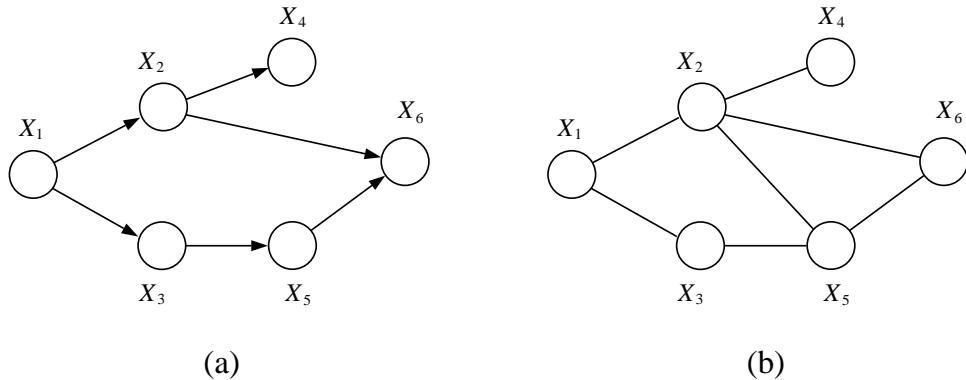


Figure 17.6: (a) A directed graph. Note that the conditional probability $p(x_6 | x_2, x_5)$ has as arguments a subset of nodes that are not contained in any clique in the graph. In the moral graph in (b), an edge has been added between X_2 and X_5 , and now the arguments in the potential $p(x_6 | x_2, x_5)$ are contained with the clique $\{X_2, X_5, X_6\}$.

graph, with the additional complication that the original potentials—the local conditional probabilities from the directed graph—need not be defined on cliques. In particular, if the parents of node X_i are not linked, then $p(x_i | x_{\pi_i})$ is not a function on a clique. To handle this situation, and thereby allow a uniform treatment of directed and undirected graphs, we *moralize* the directed graph. Recall from Chapter 3 that the moral graph G^m corresponding to a directed graph G is obtained by linking the parents of each node and dropping the directionality of the edges. We define the moralization procedure more formally in Figure 17.2. On a moral graph, the local conditional probabilities are potential functions on cliques. We associate each such probability with one and only one potential $\psi_C(x_C)$, again assuming that C includes the maximal cliques. Taking the product over these potentials is then equivalent to taking the product $\prod_i p(x_i | x_{\pi_i})$, and faithfully represents the joint probability from the underlying directed graph.

Note that for directed graphs the potentials are already normalized; in other words, the normalization factor Z is automatically one.

Figure 17.2 shows an example for a directed graph.

Note that the moralization procedure adds edges to a directed graph. How does this procedure

square with the semantic distinctions between directed graphs and undirected graphs presented in the previous chapter? Recall that a given graph—directed or undirected—is associated with a family of probability distributions. This family can be specified by writing down the list of conditional independence statements associated with the graph. Any distribution that respects all of the conditional independence statements in the list belongs to the family. Clearly, if we make *fewer* statements we make the family *larger*. Now note that a moral graph necessarily makes fewer conditional independence statements than its corresponding directed graph. In particular, a directed graph asserts all of the conditional independencies that characterize the moral graph, as well as additional independencies between the parents of a given node in the marginal distribution in which the node is eliminated. Thus the set of probability distributions associated with the directed graph is a subset of the set of probability distributions associated with the moral graph. If we solve the inference problem for the family of probability distributions associated with the undirected moral graph, we solve it for the family of probability distributions associated with the directed graph as well.

Moralization is not merely a convenience, but is also a necessary component of any inference algorithm. Marginalization or conditioning couples the parents of a node, creating an intermediate factor that is in general a non-trivial function of the parents.¹ Intuitively, moralization is necessary to capture dependencies such as “explaining-away” that arise whenever a node is an evidence node or has descendants that are evidence nodes.

To summarize, our procedure will be to identify the maximal cliques of an undirected or (moralized) directed graph.² We initialize the potential functions associated with these cliques from the potentials and local conditional probabilities on the underlying graph.

17.3 Introducing evidence

We now consider the problem of conditioning, or “introducing evidence.” We suppose that the nodes are partitioned into subsets H and E , and that the random vector X_E is observed to take on a specific value. The problem that we discuss in this section is that of representing the conditional probability $p(x_H | x_E)$. Once we have decided on such a representation, the inferential problem of computing marginals under this probability—the conditional probabilities of subsets of the nodes X_H —will be no different in principle from the calculation of marginal probabilities under the overall joint $p(x)$.

Our general approach will be to represent conditionals via taking “slices” of the potentials defining the joint probability. Suppose in particular that we have represented the joint probability as a product over cliques as in Eq. (17.2). For each clique C , consider the intersection $C \cap E$. The nodes in this intersection have been fixed to specific values, and the potential in effect now ranges

¹If a node is not an evidence node or has no descendants that are evidence nodes, summing over the values of the node yields the trivial value of one.

²Some readers may wonder how we can achieve this—finding maximal cliques is an NP-hard problem! In fact, we will not be finding the maximal cliques of arbitrary graphs, but only of a special class—the *triangulated graphs*. Maximal cliques of triangulated graphs can be found easily. Let us postpone our discussion of triangulation, however, at the cost of a bit of naiveté with regards to identifying maximal cliques.

over the complement (in C) of this set of nodes, i.e., $C \cap H$. where $C = (C \cap H) \cup (C \cap E)$ by the assumption that H and E partition V . Thus, for a particular fixed configuration \bar{x}_E , we have:

$$p(x_H, \bar{x}_E) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_{C \cap H}, \bar{x}_{C \cap E}). \quad (17.3)$$

This is a product of “slices” of potential functions.

A slice of a potential function is itself a potential function. Thus we can also view Eq. (17.3) as a product of potential functions on subsets $\{X_{C \cap H}$ of the nodes X_H , suppressing reference to the nodes X_E . That is, writing $\tilde{\psi}_{C \cap H}(x_{C \cap H}) \triangleq \psi_C(x_{C \cap H}, \bar{x}_{C \cap E})$ to suppress the explicit reference to the fixed configuration \bar{x}_E , we have:

$$p(x_H, \bar{x}_E) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \tilde{\psi}_{C \cap H}(x_{C \cap H}) \quad (17.4)$$

as a product of potential functions over X_H .

There is an oddity to Eq. (17.4), however, in that the normalization factor Z is obtained by summing over both X_H and X_E , whereas the product is defined only over X_H . It should be no surprise that Z is not in fact the normalization factor for the product of potentials $\tilde{\psi}_{C \cap H}$; indeed, this product is not normalized. Let us compute the normalization factor. Summing over H , and denoting the sum as \tilde{Z} , we compute:

$$\tilde{Z} \triangleq \sum_H p(x_H, \bar{x}_E) \quad (17.5)$$

$$= \sum_H \frac{1}{Z} \prod_{C \in \mathcal{C}} \tilde{\psi}_{C \cap H}(x_{C \cap H}). \quad (17.6)$$

We also know, however, that $\sum_H p(x_H, \bar{x}_E) = p(\bar{x}_E)$, by definition. Putting these facts together, we have:

$$\frac{p(x_H, \bar{x}_E)}{p(\bar{x}_E)} = \frac{\prod_{C \in \mathcal{C}} \tilde{\psi}_{C \cap H}(x_{C \cap H})}{\sum_H \prod_{C \in \mathcal{C}} \tilde{\psi}_{C \cap H}(x_{C \cap H})}. \quad (17.7)$$

That is, the slices $\tilde{\psi}_{C \cap H}(x_{C \cap H})$ provide a potential function representation of the *conditional* probability $p(x_H | \bar{x}_E)$. The normalization factor for this representation is the marginal probability $\tilde{Z} = p(\bar{x}_E)$. Note that the original normalization constant, Z , cancels when we form the ratio on the right-hand-side of Eq. (17.7). Thus, for the purpose of calculating conditional probabilities, we have no need of knowing the normalization constant associated with the original set of potentials; it suffices to compute the normalization constant of the sliced potentials.

Let us see how this works for a particularly simple case. In Figure 17.7. we show a directed graph and the corresponding moralized graph for two binary nodes X and Y . Given the three probabilities $p(X = 1) = .8$, $p(Y = 1 | X = 1) = .7$ and $p(Y = 1 | X = 0) = .4$, we can construct a joint probability distribution. Converting to a set of cliques, we have a single clique $\{X, Y\}$, with clique potential given by the product $p(x)p(y | x)$:

$$\psi_{\{X, Y\}} = \begin{bmatrix} .12 & .08 \\ .24 & .56 \end{bmatrix} \quad (17.8)$$



Figure 17.7: A two-node graphical model with its moralized graph.

Given that this potential arises from a directed graph, it is no surprise that the clique potential is normalized. Suppose that we now observe evidence $Y = 1$. We obtain the slice:

$$\tilde{\psi}_{\{X\}} = \begin{bmatrix} .08 \\ .56 \end{bmatrix}, \quad (17.9)$$

which is a function only of X . Note that this new clique potential is unnormalized. Normalizing yields the number $\tilde{Z} = .64$, which we recognize as the probability $p(Y = 1)$. Moreover, the normalized potential is given by dividing $\tilde{\psi}_{\{X\}}$ by $\tilde{Z} = .64$:

$$\frac{1}{\tilde{Z}} \tilde{\psi}_{\{X\}} = \begin{bmatrix} .125 \\ .875 \end{bmatrix}, \quad (17.10)$$

which is the conditional distribution $p(x | Y = 1)$.

To summarize, our general representation of a probability distribution is a (possibly) unnormalized set of potentials on a set of cliques. Conditioning is handled by restricting attention to subsets of the original set of cliques, and by defining potentials on these subsets that are slices of the original potentials. In general we make no fundamental representational distinction between conditional and joint distributions.

This perspective also helps to reveal more of the unity in undirected and directed representations of probabilities. In the directed case, the set of potentials is normalized at the outset: $Z = 1$. But as soon as we observe evidence, the resulting set of slices is no longer normalized, and the conditional distribution is represented as an unnormalized product of potential functions, as in the undirected case.

An equivalent approach to representing conditional probability distributions involves introducing “evidence potentials.” An evidence potential is a delta function, $\delta(x_E, \bar{x}_E)$, i.e., a function which is equal to one if its arguments are equal and zero otherwise. We used evidence potentials in our presentation of the elimination algorithm in Chapter 3. Multiplying the original product of potentials by the evidence potential yields an unnormalized product on the set (X_H, X_E) . Summing over x_E has the effect of setting $p(x_H, x_E)$ equal to $p(x_H, \bar{x}_E)$. Thus we obtain the same representation as that considered in this section, once we “marginalize” and restrict attention to X_H . The approach based on evidence potentials is elegant because it treats slices as formally equivalent to marginalization; indeed that was the reason that we introduced it in Chapter 3. In practice, however, using evidence potentials involves introducing zeros and then summing over those zeros. As an algorithmic matter it is more efficient to simply take slices.

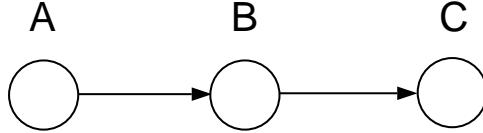


Figure 17.8: A three-node Markov chain.

17.4 Clique trees

We now begin to address the crux of the problem, which is that of computing *marginal probabilities*. Thus, we wish to compute the marginal $p(x_F | x_E)$, where (F, G) is a partition of H and where F ranges over a set of subsets of interest. In particular, we may wish to compute all probabilities $p(x_F | x_E)$, where F ranges over all singleton nodes. More generally, we will address the problem of computing $p(x_F | x_E)$, where F ranges over all cliques in \mathcal{C} , and over all subsets of these cliques.

A still more general problem is that of computing $p(x_F | x_E)$ for arbitrary F , and while we will address this problem in Section ??, it is worth noting that in most applications it suffices to compute marginal probabilities for the cliques. In particular, the cliques are sufficient statistics for distributions that factor according to Eq. (17.2); thus, for computing expected sufficient statistics in the context of an EM algorithm it suffices to obtain clique marginals.

We define a *clique tree* as a singly-connected graph whose nodes represent members of the clique set \mathcal{C} . Edges in this graph will allow us to define information flows between cliques. The junction tree algorithm can be understood as an algorithm that uses these information flows to manipulate the clique potentials so as to yield marginal probabilities. In particular, after the algorithm runs, the potential ψ_C will be equal to the marginal probability $p(x_C, \bar{x}_E)$. This probability is an un-normalized version of the conditional $p(x_C | \bar{x}_E)$, where the normalization constant is obtained by summing or integrating ψ_C over x_C . Thus, we can obtain the desired marginal probabilities via a local operation. The goal of the remainder of the chapter is to explain how this is achieved.

In the previous two sections, we showed how to initialize the clique potentials so as obtain a representation of the joint or conditional probability. This is a global representation; the individual potentials do not necessarily correspond to local probabilities. Consider in particular the Markov chain shown in Figure 17.8. The cliques of this graph are $\{A, B\}$ and $\{B, C\}$. The joint probability is $p(x_A, x_B, x_C) = p(x_A)p(x_B | x_A)p(x_C | x_B)$, and while $p(x_A)$ and $p(x_B | x_A)$ can be grouped to initialize the potential ψ_{AB} to the marginal $p(x_A, x_B)$, the remaining factor $\psi_{BC} = p(x_C | x_B)$ is not a marginal. To convert this potential into a marginal, we marginalize ψ_{BC} to obtain $p(x_B)$, and multiply ψ_{BC} by this factor. The transfer of the probability $p(x_B)$ is an instance of the information flow that we referred to above.

After adjusting ψ_{BC} we have achieved the goal of obtaining marginal probabilities for both of the cliques, but we have also lost something. In particular, the joint probability on (x_A, x_B, x_C) is not equal to the product of marginals $p(A, B)$ and $p(B, C)$, and thus the product of the clique potentials is no longer a representation of the joint probability.

The junction tree approach in essence allows us to have our cake and eat it too, retaining a representation of the joint probability while also manipulating the clique potentials so as to convert

them into marginal probabilities. This is done by utilizing an extended representation of joint probabilities that makes use of the separator sets discussed in Section ???. The remainder of this section introduces this important generalized representation.

On each edge of a clique tree we associate a *separator set* which contains the intersection of the cliques that it links. For example, in Figure 17.16, the separator is the singleton X_B . For a general clique tree on N nodes, we have $N - 1$ separators.

We now augment our potential-based representation of joint probabilities to include potential functions on the separators as well as the cliques. Thus, letting \mathcal{S} denote the set of all separators, we introduce a potential function $\phi_S(x_S)$ for each $S \in \mathcal{S}$. Given a clique tree with cliques \mathcal{C} and separators \mathcal{S} we define the joint probability as follows:

$$p(x) = \frac{\prod_C \psi_C(x_C)}{\prod_S \phi_S(x_S)}. \quad (17.11)$$

Note that we have omitted explicit reference to a normalizing constant Z . We adopt a convention of including the empty set as one of the separators and letting the “potential” on this empty set be the normalizing constant Z .

We have several questions to answer regarding this extended representation, but let us first return to our example and show what the representation achieves for us.

Expanding the joint probability associated with Figure 17.8, we have:

$$p(x_A, x_B, x_C) = p(x_A, x_B)p(x_C | x_B) \quad (17.12)$$

$$= \frac{p(x_A, x_B)p(x_B, x_C)}{p(x_B)}. \quad (17.13)$$

This has the form of the extended representation shown in Eq. (17.11), where we define $\psi_{AB} = p(x_A, x_B)$, $\psi_{BC} = p(x_B, x_C)$, and $\phi_B = p(x_B)$. Thus, making use of the flexibility offered by the separator potentials, we are able to achieve a representation that is a product of marginals, and yet is also a representation of the joint probability. It turns out that we can always find this kind of representation for a given probability distribution. The proof of this fact will emerge during our development of the junction tree algorithm.

In our discussion of the Hammersley-Clifford theorem in Chapter 16, we showed that the representation of joint probability in Eq. (17.2) is general, in the sense that it allows us to capture all of the joint probability distributions that respect the conditional independence statements asserted by a graph. Clearly the extended representation includes all such joint probability distributions (set the separator potentials to unity). Does it include any others? The answer is no. This is seen by noting that the separators are (by definition) subsets of one or more cliques. Associating each separator with one such clique, and dividing that clique potential by the separator potential, we obtain a new set of clique potentials that represent the same joint, but without the separators. Thus the separator potentials do not enlarge the set of joint probability distributions that we can represent. They are essentially a convenience—they allow us to represent the set of joint probability distributions associated with a graphical model in a more flexible way.

An additional issue that we need to consider is the possibility of division by zero. We allow division by zero but only in a constrained set of circumstances. In particular, we define a separator

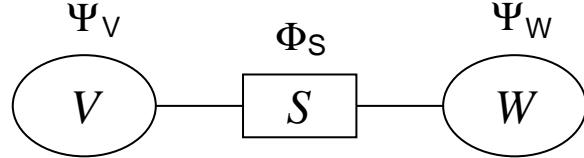


Figure 17.9: The basic data structures underlying the flow of information between cliques V and W .

potential to be *supportive* if whenever a configuration yields a value of zero for the separator potential, the clique potentials at both ends of the edge containing that separator also evaluate to zero. Thus we can never divide by zero in Eq. (17.11) unless the numerator is also zero. In this case we define the ratio to be zero. This makes sense—if a clique potential is zero for a configuration then the probability of that configuration should also be zero.

Each step of the junction tree algorithm is guaranteed to maintain supportiveness (see Exercise ??). Thus, if we have supportive separator potentials at the outset then we maintain supportiveness as the algorithm runs.

We initialize the separator potentials to unity. Thus, at the outset, once we have introduced evidence, the set of clique potentials and separator potentials are (as before) a global representation of the joint conditional probability $p(x_H | x_E)$. The new capability that the extended representation has provided is the ability (in principle) to obtain a local representation of marginal probabilities, while maintaining an overall representation of the joint. We now show how this is achieved in practice.

17.5 Local consistency

Note that cliques can overlap, so the same node can appear in multiple cliques. Clearly, if the potentials are to represent marginal probabilities, it is necessary that they be consistent with each other; that is, they must give the same marginals for nodes that they have in common. This seemingly innocuous observation is the germ of the junction tree algorithm. We will find that consistency is not only a necessary condition, but it is also a sufficient condition for a probabilistic inference algorithm. Moreover, it turns out not to be necessary to compare all pairs of cliques that intersect; it will suffice to arrange the cliques into a special clique tree—a “junction tree”—and require only that cliques that are neighbors in the junction tree agree on the nodes that they have in common.

Let us postpone the general junction tree construction, and instead focus on the elemental problem of achieving consistency between a pair of cliques. Suppose that we have two cliques V and W and suppose that V and W have a non-empty intersection S (see Figure 17.9). The cliques V and W have potentials ψ_V and ψ_W , and we also endow S with a potential ϕ_S that we initialize to unity. The basic operation of the junction tree algorithm is an exchange of information between V and W , with S serving as a conduit for the flow of information. We first update W based on V ,

where the asterisk means “updated value of”:

$$\phi_S^* = \sum_{V \setminus S} \psi_V \quad (17.14)$$

$$\psi_W^* = \frac{\phi_S^*}{\phi_S} \psi_W. \quad (17.15)$$

The first equation *marginalizes* the potential ψ_V with respect to S , storing the result in the separator potential. The second equation *rescales* the potential on W by multiplying by an “update factor” that is the ratio of the new separator potential to its old value.

This update has an important invariant: the joint distribution $p(x_H, \bar{x}_E)$. Note that ψ_V is unchanged during the update. Defining $\psi_V^* = \psi_V$, we have:

$$\frac{\psi_V^* \psi_W^*}{\phi_S^*} = \frac{\psi_V \psi_W \phi_S^*}{\phi_S \phi_S^*} \quad (17.16)$$

$$= \frac{\psi_V \psi_W}{\phi_S}, \quad (17.17)$$

and thus the joint distribution as defined in Eq. ?? is unchanged. Whether or not we have achieved anything useful with the update is as yet unclear; but at least the joint probability has not been altered.

We now pass information from W back to V , using the same update rule. In particular:

$$\phi_S^{**} = \sum_{W \setminus S} \psi_W^* \quad (17.18)$$

$$\psi_V^{**} = \frac{\phi_S^{**}}{\phi_S^*} \psi_V^*. \quad (17.19)$$

(Noting that ψ_W^* is unchanged during this update, we define $\psi_W^{**} = \psi_W^*$).

Note that once again the joint probability $p(x_H, \bar{x}_E)$ remains unaltered by the update.

There is another important property that characterizes the pair of updates. In particular, the potentials ψ_V^{**} and ψ_W^{**} are consistent with respect to their intersection S ; that is, they have the same marginals. This is easily verified:

$$\sum_{V \setminus S} \psi_V^{**} = \sum_{V \setminus S} \frac{\phi_S^{**}}{\phi_S^*} \psi_V^* \quad (17.20)$$

$$= \frac{\phi_S^{**}}{\phi_S^*} \sum_{V \setminus S} \psi_V^* \quad (17.21)$$

$$= \frac{\phi_S^{**}}{\phi_S^*} \phi_S^* \quad (17.22)$$

$$= \phi_S^{**} \quad (17.23)$$

$$= \sum_{W \setminus S} \psi_W^{**}. \quad (17.24)$$

Inspecting this derivation, we see that the key steps for achieving consistency are Eqs. 17.14 and 17.19. In the forward pass, from V to W , the algorithm stores the marginal of the V potential in the separator potential. In the backward pass, from W to V , the algorithm divides the V potential by its stored marginal and multiplies the result by the new marginal ϕ_S^{**} . This latter marginal is the marginal of the W potential. The rescaling equation essentially substitutes one marginal for another, thus making the two clique potentials consistent. This is achieved in the context of a symmetric algorithm that passes information in both directions, and leaves the joint probability distribution invariant.

Consider for example the Markov chain in Figure 17.8. Initially, the clique potential on $\{X, Y\}$ is $p(x, y)$, and the clique potential on $\{Y, Z\}$ is $p(z | y)$. The first pair of update equations results in the following update:

$$\phi_Y^* = \sum_x p(x, y) = p(y) \quad (17.25)$$

$$\psi_{YZ}^* = \frac{p(y)}{1} p(z | y) = p(y, z), \quad (17.26)$$

and we see that the clique potentials have become marginal probabilities. The backward phase in this case is vacuous; marginalizing over $p(y, z)$ yields $p(y)$ again for the separator marginal and the update factor is unity.

Now consider the chain in the case in which evidence is observed. Suppose for simplicity that all nodes are binary, and the evidence is $X = 1$. Incorporating the evidence means taking the slice of the potential on $\{X, Y\}$ in which $X = 1$; i.e., taking the second row of the potential table. The marginalization operation is now a vacuous operation, and we have:

$$\phi_Y^* = p(X = 1, y). \quad (17.27)$$

Performing the update of the $\{Y, Z\}$ potential yields:

$$\psi_{YZ}^* = p(X = 1, y)p(z | y) = p(X = 1, y, z). \quad (17.28)$$

Thus our potentials are as follows:

$$\psi_{XY}^* = p(X = 1, y) \quad (17.29)$$

$$\phi_Y^* = p(X = 1, y) \quad (17.30)$$

$$\psi_{YZ}^* = p(X = 1, y, z), \quad (17.31)$$

and we see that we have obtained marginals as before, but these are unnormalized marginals. Normalizing (a local operation), we can readily read off the conditionals $p(y | X = 1)$, $p(y | X = 1)$, and $p(y, z | X = 1)$. Note that once again the backward pass is vacuous.

The reader may wish to try the cases in which evidence $Z = 1$ is available and when both $X = 1$ and $Z = 1$ are observed.

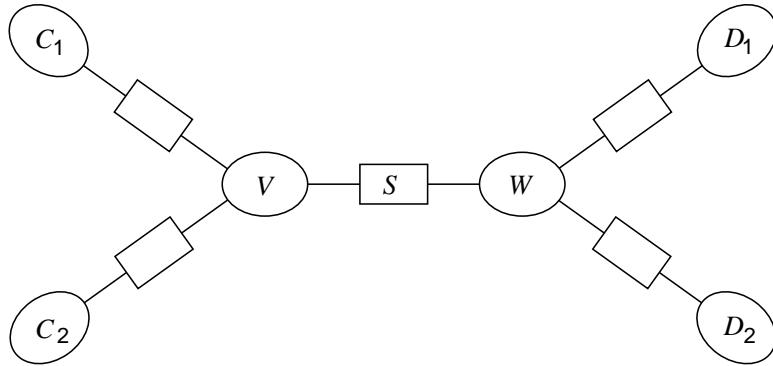


Figure 17.10: A clique tree with explicit representation of the separators. The separators are the intersection of the pair of cliques at the ends of the edge. Thus, for example, $S = V \cap W$.

17.6 Propagation in a clique tree

We now turn to the issue of how to perform local updates when we have multiple overlapping cliques.

In Figure 17.10 we show a clique tree. Each edge in this tree is associated with a separator. Cliques that are neighbors in this tree are subject to the updating procedure described in the previous section.

There are two issues that we must address—how to construct an appropriate clique tree and how to perform the updates so that local consistency obtained between a clique and its neighbor is not ruined by subsequent updates between the clique and other neighbors. In this section we focus on the second issue, returning to the problem of constructing the tree in Section 17.10.

How do we maintain local consistency in a clique tree? Consider again the clique tree shown in Figure 17.10. Suppose that we were to achieve local consistency between V and W using the pair of updates discussed in the previous section, and subsequently we update W based on its other neighbors. The latter updates would generally ruin the consistency that has been achieved between V and W . To ensure that this does not happen, we develop a protocol that constrains the order in which updates are performed.³

Let us refer to the update of one clique based on another as a “message-passing” operation. That is, we “pass a message” from V to W by evaluating Eqs. 17.14 and 17.15. In general, as we saw in the previous section, we require a message in both directions in order to render a pair of cliques consistent with each other.

Our problem is to decide when a given clique is allowed to pass a message to one of its neighbors. This problem is solved by the following protocol:

Message-Passing Protocol. *A clique can send a message to a neighboring clique only when it has received messages from all of its other neighbors.*

³In fact the protocol is not needed if we are willing to perform redundant steps. If each node is updated repeatedly (for example in parallel), consistency-ruining steps will eventually be corrected (see Exercise ??).

For example, in Figure 17.10, we can send a message from W to V only when W has received messages from its other neighbors D_1 and D_2 .

An easy argument establishes the correctness of the protocol. Consider the moment in time at which W has received all of the messages from its other neighbors, and is sending a message to V . There are two cases to consider: either V has not yet sent its message to W , or V has already sent its message to W . In the latter case, we know that V has already received messages from all of its other neighbors. The message from W to V renders the cliques consistent. Neither clique receives any additional messages, thus consistency is maintained. In the former case, W sends a message to V , storing its marginal on S , and waits. At some later time, V will have received all of the messages from its other neighbors and will send a message to W . This message will utilize the stored marginal and render W consistent with V . Neither clique will undergo any additional updates and consistency is maintained.

Although our protocol is correct, is it realizable? Are there message-passing algorithms that realize the protocol and ensure that a message is passed in both directions between every pair of cliques?

There are in fact many message-passing algorithms that realize the protocol; their existence is a simple consequence of the recursive definition of a tree. One way to obtain such algorithms is based on designating one clique in the tree as the root. Once a root of the clique tree is designated, the tree becomes an oriented tree with each leaf having a unique path to the root. Clearly each leaf can send a message inward at any time. Interior nodes send a message toward the root once they have received messages from all of their children. Once all messages have arrived at the root, we propagate messages outward to the leaves.

More formally, we define the following pair of recursive procedures:

```

COLLECTEVIDENCE( node )
begin
    for each child of node
        begin
            UPDATE( node, COLLECTEVIDENCE( child ) )
        end
    return( node )
end

DISTRIBUTEVIDENCE( node )
begin
    for each child of node
        begin
            UPDATE( child, node )
            DISTRIBUTEVIDENCE( child )
        end
    end
end

```

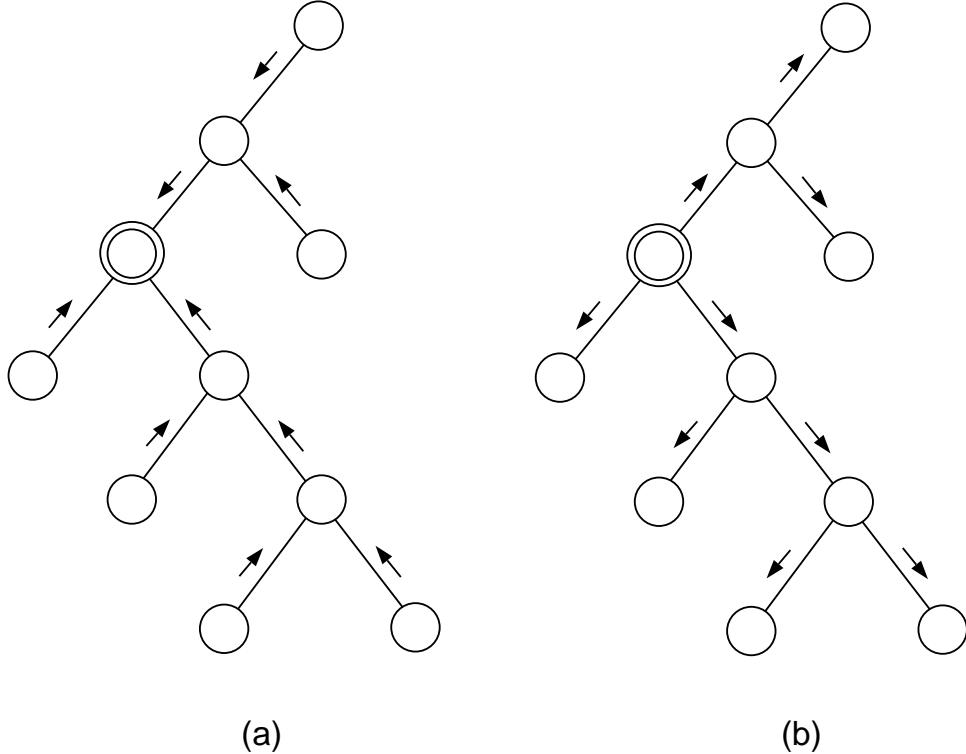


Figure 17.11: (a) The message-passing resulting from a call of `COLLECTEVIDENCE` at the root node (the doubly-circled node). (b) The message-passing resulting from a call of `DISTRIBUTEEVIDENCE` at the root node.

where $\text{UPDATE}(V, W)$ is a routine that invokes the pair of equations Eq. 17.14 and 17.15. Calling `COLLECTEVIDENCE(root)` followed by `DISTRIBUTEEVIDENCE(root)` causes messages to propagate inward to the root and outward to the leaves.

Theorem 1 *The `COLLECTEVIDENCE` and `DISTRIBUTEEVIDENCE` recursions respect the Message-Passing Protocol.*

Proof. When `COLLECTEVIDENCE` is called at a node, the node calls all of its other neighbors and waits on return messages from those nodes before returning a message back to its caller. Thus `COLLECTEVIDENCE` obeys the protocol.

After `COLLECTEVIDENCE` has run, each node has received a message from all of its neighbors except its parent. Once it receives a message from its parent it is free to send messages to any other node. `DISTRIBUTEEVIDENCE` sends a message from a parent to its child before calling itself on that child. Thus `DISTRIBUTEEVIDENCE` respects the protocol. \square

Consider the example shown in Figure 17.11, where the doubly-circled node is designated as

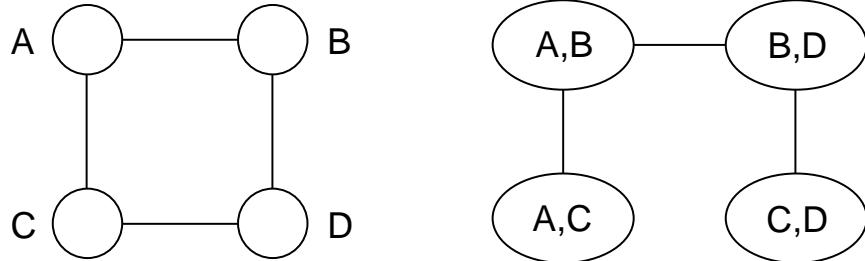


Figure 17.12: An undirected graphical model and a corresponding clique tree.

the root node. A call of `COLLECTEVIDENCE` results in messages proceeding inward as shown in Figure 17.11, and a call of `DISTRIBUTEVIDENCE` results in the outward-going messages shown in Figure 17.11. Note that it is clear that one and only one message is passed in both directions between every pair of cliques.

17.7 The junction tree property

At this point we have developed most of the machinery associated with the junction tree algorithm, and we are in the position to describe recursive inference algorithms for some non-trivial graphical models. In fact the machinery discussed thus far is sufficient to handle all of the models that we considered in Part I. In particular, an impatient reader could jump to Chapter 18 to see how the algorithm specializes to the case of the HMM and the state-space model. Both of those cases involve a rather obvious choice for the tree of cliques, and given a particular choice of root node, the recursive algorithms that we developed in earlier chapters fall out naturally from `COLLECTEVIDENCE` and `DISTRIBUTEVIDENCE`.

Despite this heady success, we have as yet no theoretical guarantee that the algorithm is correct for general graphical models. In fact it turns out that the algorithm as developed thus far is *not* correct for general graphical models. In this section we identify the (last) problem that must be addressed. We should emphasize at the outset that the problem is essentially a data structure problem involving the construction of the clique tree. There is in fact no problem with our marginalizing and rescaling equations, nor with our Message-Passing Protocol. It suffices to get the data structure right.

To see that our labor is not yet finished, consider the undirected graphical model shown in Figure 17.12. There are four cliques in this graph. A particular choice of clique tree is shown in Figure 17.12. Note that this clique tree has a problematic feature. In particular, the node C appears in two different cliques in the tree and these cliques are not neighbors. Given that our algorithm only enforces local consistency, there is no guarantee that the two cliques containing C will be consistent. Indeed, if the leftmost clique that contains C is changed (e.g., by the introduction of evidence), there is no mechanism to insure that this information will flow to the rightmost clique that contains C . In general, local consistency does not imply global consistency.

Note that the lack of global consistency does not imply that we have an incorrect representation

of the joint probability distribution. Indeed, as we saw earlier, the junction tree algorithm does not alter the joint probability, and thus we maintain a correct representation of the joint throughout. What we fail to achieve in Figure 17.12 is locality—the clique potentials correctly represent the joint probability, but they are not local marginal probabilities.

The reader can verify that there is no alternative clique tree that avoids the problem. All clique trees have a pair of nodes that lie in non-neighboring cliques.

A clue to understanding the problem comes from observing that the elimination algorithm would unavoidably create new links in the graph in Figure 17.12; e.g., eliminating C would connect A and B . Another way to put the problem is that there is no way to choose an elimination ordering such that the elimination cliques are contained within the cliques of the original graph.

While this argument based on elimination provides insight, we prefer to restate the problem directly in terms of properties of clique trees. To do so, we articulate a property that rules out the problematic configurations of the kind that we saw in Figure 17.12. The relevant property is known as the *junction tree property*:

The junction tree property. A clique tree possesses the *junction tree property* if for every pair of cliques V and W , all cliques on the (unique) path between V and W contain $V \cap W$.

A clique tree that possesses the junction tree property is referred to as a *junction tree*.

The consequences of the junction tree property for inference are as follows. If a node A appears in two cliques in a junction tree, then A is contained in every clique along the path between these two cliques. If the cliques along the path are *pairwise* consistent with respect to A then they will be *jointly* consistent with respect to A . *In a junction tree, local consistency implies global consistency.*

This argument implies that if we are fortunate enough to have a clique tree that is a junction tree, and if we run the message-passing procedure as described in the previous section, we achieve not only local consistency but also global consistency. We can get the same answer for any node A by consulting any potential that contains A .

Recall however that our goal is to obtain a set of potentials that are not only consistent, but are also marginals—that is, each potential represents the marginal probability of the nodes in its clique. It is conceivable that the junction tree could be consistent, but the potentials would not be marginals. In fact, somewhat surprisingly, this cannot be the case. In a junction tree, the junction tree algorithm not only achieves global consistency, but it yields the sought-after clique marginals as well. To prove this important result we require the following lemma.

Lemma 1 *Let C be a leaf in a junction tree for a graph with vertex set V . Let S be the associated separator (see Figure 17.13). Let $R = C \setminus S$ be the set of nodes in C but not in the separator, and let $U = V \setminus C$ be the set of nodes in V but not in C . Then:*

$$R \perp\!\!\!\perp U \mid S \tag{17.32}$$

Proof. Suppose, by way of contradiction, that $A \in R$ has a neighbor $N \in U$. Consider the maximal complete subset containing both A and N . This clique is not C because $N \notin C$. However, A cannot be contained in any clique other than C because A would have to belong to S as well, by

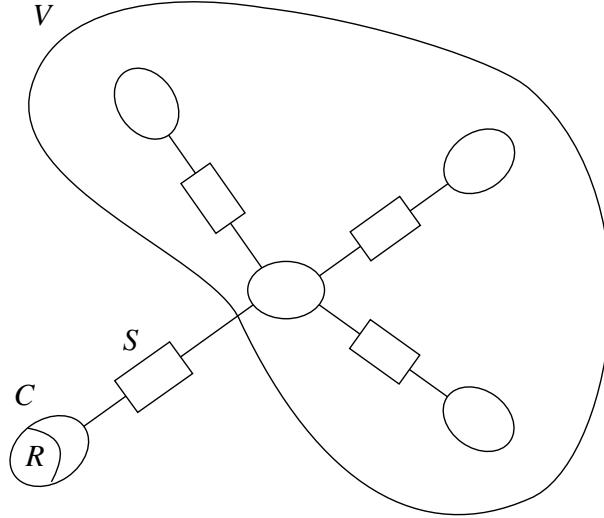


Figure 17.13: The “residual” set $R = C \setminus S$ is the set of nodes in C that are not in S , and, by the junction tree property, also not in U .

the junction tree property, and nodes in R are not in S by definition. Thus no such N exists and S must therefore separate A from U . Since A is arbitrary, S separates R from T . \square

We now state and prove our main result.

Theorem 2 *Let the probability $p(x_H, \bar{x}_E)$ be represented by the clique potentials ψ_C and separator potentials ϕ_S of a junction tree. When the junction tree algorithm terminates, the clique potentials and separator potentials are proportional to local marginal probabilities. In particular:*

$$\psi_C = p(x_C, \bar{x}_E) \quad (17.33)$$

$$\phi_S = p(x_S, \bar{x}_E) \quad (17.34)$$

Proof. The separators are subsets of the cliques. That the separator potentials are proportional to marginals therefore follows from the fact that they are consistent with the clique potentials. Thus we need only prove the result for the clique potentials.

The proof is a proof by induction. The result holds for the base case of a single clique by definition. Let us suppose that the result holds for junction trees of N or fewer cliques, and consider a junction tree with $N + 1$ cliques.

We choose a clique \tilde{C} that is a leaf in the junction tree. Let \tilde{S} be the corresponding separator, let $\tilde{R} = \tilde{C} \setminus \tilde{S}$ and let $\tilde{T} = V \setminus \tilde{C}$. We also define analogous quantities in which the evidence variables are omitted. In particular, let $C = \tilde{C} \setminus E$, $R = \tilde{R} \setminus E$ and $T = \tilde{T} \setminus E$. By Lemma 1 we have:

$$p(x_H, \bar{x}_E) = p(x_R, x_S, x_T, \bar{x}_E) = p(x_R | x_S, \bar{x}_E)p(x_S, x_T, \bar{x}_E). \quad (17.35)$$

Summing both sides over R , we obtain:

$$p(x_S, x_T, \bar{x}_E) = \sum_R p(H, \bar{x}_E) \quad (17.36)$$

$$= \sum_R \frac{\prod_C \psi_C(x_C)}{\prod_S \phi_S(x_S)} \quad (17.37)$$

$$= \sum_R \frac{\psi_C}{\phi_S} \frac{\prod_{C' \neq C} \psi_{C'}(C')}{\prod_{S \neq S'} \phi_{S'}(x'_S)} \quad (17.38)$$

$$= \frac{\sum_R \psi_C}{\phi_S} \frac{\prod_{C' \neq C} \psi_{C'}(C')}{\prod_{S \neq S'} \phi_{S'}(x'_S)} \quad (17.39)$$

$$= \frac{\prod_{C' \neq C} \psi_{C'}(C')}{\prod_{S \neq S'} \phi_{S'}(x'_S)} \quad (17.40)$$

where Eq. 17.40 follows from the fact that C and S are consistent and thus $\sum_R \psi_C = \phi_S$.

Eq. 17.40 shows that $p(x_S, x_T, \bar{x}_E)$ is represented by the clique potentials and separator potentials on the junction tree over $S \cup T$. By the induction hypothesis, after a full round of message passing the clique potentials on this junction tree are equal to marginals.

It remains to show that the clique potential on C is a marginal. Let D be the neighbor of C in the junction tree. By consistency we have $\phi_S(x_S) = \sum_{D \setminus S} \psi_D(x_D)$. We have $\psi_D = p(x_D, \bar{x}_E)$ and thus $\psi_S(x_S) = p(x_S, \bar{x}_E)$. Thus:

$$p(x_R | x_S, \bar{x}_E) = \frac{\psi_C(x_C)}{\phi_S(x_S)} \quad (17.41)$$

$$= \frac{\psi_C(x_C)}{p(x_S, \bar{x}_E)} \quad (17.42)$$

which implies $\psi_C(x_C) = p(x_C, \bar{x}_E)$. \square

17.8 Triangulated graph \Rightarrow Junction tree

The junction tree property provides a sufficient condition for the correctness of the junction tree algorithm. What class of graphs have a junction tree? How do we handle graphs that do not have a junction tree?

In this section we present a sufficient condition for a graph to have a junction tree—the condition is that the graph must be *triangulated*. It turns out that triangulation is also a necessary condition for a graph to have a junction tree. In the current section, however, we restrict ourselves to the proof of sufficiency, proving necessity in Appendix A. The Appendix also demonstrates that triangulation is equivalent to *decomposability*; a characterization of graphs that we discussed in Section ??.

We begin by defining a triangulated graph and then proceed to the proof of sufficiency. The reader willing to accept the proof on faith can read the definition of triangulation in the next paragraph and then skip to the following section without loss of continuity.

Consider a cycle in an undirected graph. A cycle is *chordless* if there are no edges between nodes that are not successors in the cycle. For example, the cycle $A - B - D - C - A$ in Figure 17.12 is chordless because there is no edge between A and C or between B and D . A graph is said to be *triangulated* if there are no chordless cycles in the graph.

Our first stop in the proof of sufficiency is a simple lemma that shows that triangulated graphs can be decomposed into three subsets with special properties.

Lemma 2 *Let $\mathcal{G} = (V, E)$ be a noncomplete triangulated graph with at least three nodes. Then there exists a decomposition of V into disjoint sets A , B and S such that S separates A and B and S is complete.*

Proof. Choose a pair of nonadjacent nodes α and β . Let S be the minimal set of nodes such that any path from α to β passes through S . Let A be the set of nodes reachable from α when S is removed and similarly let B be the set of nodes reachable from β when S is removed. Clearly these two sets are separated by S . We need only establish that S is complete.

Let C and D be nodes in S . Since S is minimal, there is a path from α to C and from α to D ; thus there is a path from C to D in $A \cup S$. Take the shortest such path. Similarly take the shortest path joining C to D in $B \cup S$. Link these paths to obtain a cycle. This cycle must have a chord. This chord must be an edge between C and D , by our choice of shortest paths. Thus C and D are neighbors. \square

We also require the notion of a simplicial node. A node is *simplicial* if all of its neighbors are connected. The following lemma guarantees the existence of simplicial nodes in triangulated graphs.

Lemma 3 *Every triangulated graph that contains at least two nodes has at least two simplicial nodes. If the graph is not complete, then these nodes can be chosen to be nonadjacent.*

Proof. We again use induction and again the base case is trivial. Consider a triangulated graph \mathcal{G} with $N + 1$ nodes. If the graph is complete then all nodes are simplicial. Otherwise we use Lemma 2 to decompose the graph into disjoint sets A , B and S . The subgraphs $A \cup S$ and $B \cup S$ cannot contain any chordless cycles (because any such cycles would also be chordless in \mathcal{G}), and thus they are both triangulated. The induction hypothesis implies the existence of two simplicial nodes in $A \cup S$. If $A \cup S$ is not complete these can be taken to be nonadjacent, and, given that S is complete, one of the two nodes can be taken to be in A . Otherwise, pick any node in A . Similarly, the induction hypothesis implies the existence of two simplicial nodes in $B \cup S$, and one of these can be taken in B . Given that A and B are separated by S , the two nodes that we have selected are simplicial in \mathcal{G} and they are also nonadjacent. \square

We now demonstrate that triangulation implies the existence of a clique tree with the junction tree property.

Theorem 3 *All triangulated graphs have a junction tree.*

Proof. We once again use induction and once again the base case is trivial. Consider a graph \mathcal{G} with $N + 1$ nodes. By Lemma 3, the graph has at least one simplicial node α .

Removing a simplicial node from a triangulated graph yields a triangulated graph, because no chordless cycles can be created. Thus by the induction hypothesis, the graph with α removed has a junction tree T . We construct a junction tree for \mathcal{G} from T .

Let C denote the clique formed by α and its neighbors. If $C \setminus \alpha$ is a clique in T , then simply add α to that clique; T with the augmented clique is a junction tree for \mathcal{G} .

If $C \setminus \alpha$ is not a clique D in T , then it is a subset of a clique D in T . Add C as a new leaf node for T , with a link to D and a separator set $S = C \setminus \alpha$. The result is a junction tree. This is established by noting that (1) α is contained only in C and therefore cannot violate the junction tree property; and (2) all other nodes in C are in S and in D and therefore cannot violate the junction tree property. \square

17.9 Elimination \Rightarrow Triangulation

In this section we show that `UNDIRECTEDGRAPHELIMINATE` can be viewed as a procedure for creating a triangulated graph. This result will show us how to deal with nontriangulated graphs within the junction tree framework. It also allows us to demonstrate that the elimination algorithm is a special case of the junction tree algorithm.

Recall that `UNDIRECTEDGRAPHELIMINATE` is a simple iterative algorithm that successively eliminates the nodes in a graph by (1) connecting the (remaining) neighbors of the node and (2) removing the node and its edges from the graph. The input to the algorithm is a graph and an elimination ordering.

Theorem 4 `UNDIRECTEDGRAPHELIMINATE` yields a triangulated graph.

Proof. We prove the theorem by induction. The base case is a graph with a single node, which is obviously triangulated. Suppose now that the hypothesis holds for graphs with N or fewer nodes and consider a graph with $N + 1$ nodes. Eliminating a node results in a graph with N nodes, which cannot contain a chordless cycle by the induction hypothesis. Moreover, it is not possible to form a chordless cycle involving the eliminated node, because the elimination step connects all of the neighbors of the node. \square

Thus the edges added by the `UNDIRECTEDGRAPHELIMINATE` algorithm are exactly those that turn a nontriangulated graph into a triangulated graph.

This result suggests the following general approach to dealing with nontriangulated graphs. Given an initial undirected graph (possibly obtained by moralizing a directed graph), we first triangulate the graph using `UNDIRECTEDGRAPHELIMINATE`. We are not constrained in our choice of elimination ordering and can use any of a variety of heuristics to choose a “good” elimination ordering; e.g., one that introduces as few extra edges as possible (see Appendix A). Given a triangulation, we construct a junction tree from the triangulated graph and run the message-passing procedure. The algorithm calculates marginal probabilities for all of the cliques of the triangulated graph. Marginals for subsets of these cliques (e.g., individual nodes) can be obtained by further marginalization and normalization of individual potentials.

The correctness of this approach follows from an argument similar to that used to justify moralization. Adding edges to a graph can only decrease the set of conditional independencies

associated with the graph and thus expand the set of probability distributions associated with the graph. This implies that the set of probability distributions associated with the triangulation of a graph includes the set of probability distributions associated with the original graph. Solving the inference problem for the triangulated graph solves it for the original graph as well.

Our argument also suggests (correctly) that the elimination algorithm is a special case of the junction tree algorithm. As we ask the reader to show in Exercise ??, applying the junction tree algorithm to the cliques of the triangulated graph resulting from a given elimination ordering we recover exactly the probabilistic calculations of the elimination algorithm.

It is possible to prove a converse to Theorem 4 showing that for any triangulated graph there exists an ordering such that elimination using that ordering introduces no new edges.⁴ Thus, elimination and triangulation are essentially equivalent notions. This does not imply, however, that practical algorithms for triangulation are necessarily best viewed as elimination algorithms. Rather, treating triangulation as a combinatorial optimization problem provides a broader perspective on the problem. In Appendix A, we return to these issues and describe practical algorithms for graph triangulation.

If our goal is to obtain the marginal probabilities of all of the non-evidence nodes in the graph, then the naive elimination algorithm would require us to choose different elimination orderings in which the target node is the final node in the ordering. These different elimination orderings would in general produce incommensurate elimination cliques, and make it difficult, if not impossible, to share the intermediate potentials. The junction tree framework, on the other hand, calculates a single triangulation, in effect using a single elimination ordering. While this ordering may not be optimal for calculating any given individual marginal, the choice of a single ordering makes it possible to share intermediate potentials, and thus supports the efficient calculation of marginals for all cliques in the graph.

17.10 Constructing the junction tree

The results of Section 17.8 show that every triangulated graph has a junction tree. This proof—an existence proof—leaves us just short of our goal. How do we construct a junction tree from a triangulated graph?

It is certainly not the case that every clique tree obtained from a triangulated graph is a junction tree. Consider the triangulated graph shown in Figure 17.14(a). The clique tree in Figure 17.14(b) is not a junction tree (consider node B). A junction tree for this graph is shown in Figure 17.14(c).

The separators in Figure 17.14(b) are $\{C, D\}$ and $\{D\}$, whereas in Figure 17.14(c) the separators are $\{B, D\}$ and $\{C, D\}$. The total cardinality of the separator sets is larger in the latter figure. Intuitively this fact would seem to have something to do with the fact that Figure 17.14(c) possesses the junction tree property while Figure 17.14(b) does not.

To each clique tree T associated with a triangulated graph we can assign a *weight* $w(T)$ given by the sum of the cardinalities of the separator sets in the tree. We show in this section that a clique tree is a junction tree if and only if it has maximal weight, ranging over all possible trees of cliques. There may be several such trees.

⁴See, e.g., Jensen, (1996).

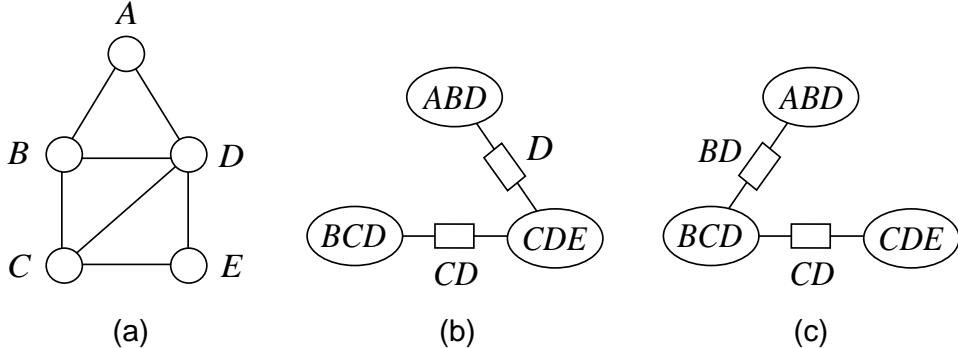


Figure 17.14: (a) A triangulated graph. (b) A clique tree based on (a) that does not have the junction tree property. (c) A clique tree based on (a) that does have the junction tree property.

Our problem is an instance of the classical “maximal spanning tree problem.” The problem is readily solved via one of a number of simple greedy algorithms. One solution is given by Kruskal’s algorithm: Begin with no edges between the cliques. At each step add an edge that has maximal separator cardinality, ensuring that the resulting graph has no cycles. Once the graph is fully connected (there is a path between any pair of cliques), we have a maximal spanning tree.⁵

Consider a node X_k and a clique tree T with cliques C_i and separators S_j . Consider further the count of the number of times that X_k appears as an element in one the cliques C_i , as well as the count of the number of times that X_k appears as an element in one of the S_j . Clearly these counts are related, and in particular the fact that T is a tree implies that the latter count is no more than the former count less one:

$$\sum_{j=1}^{M-1} \mathbb{1}(X_k \in S_j) \leq \sum_{i=1}^M \mathbb{1}(X_k \in C_i) - 1, \quad (17.43)$$

where $\mathbb{1}(\cdot)$ is the indicator function and where M is the number of cliques. Moreover, this inequality becomes an equality when the subgraph of T induced by X_k is a tree.

As we have noted earlier, the statement that the subgraph of T induced by a node X_k is a tree is nothing more than a restatement of the junction tree property. Thus we have in Eq. 17.43 an inequality which is indicative of the junction tree property, at least with respect to a single node X_k .

We are now ready to state the theorem linking junction trees and maximal spanning trees.

Theorem 5 *A clique tree T is a junction tree if and only if it is a maximal spanning tree.*

Proof. The total weight of a clique tree is equal to the sum of the cardinalities of its separators.

⁵See Cormen, Leiserson, and Rivest (1990) for a proof of this result. Another approach is given by Prim’s algorithm, which maintains a partial tree at each step and iteratively adds nodes to this tree.

Thus we have:

$$w(T) = \sum_{j=1}^{M-1} |S_j| \quad (17.44)$$

$$= \sum_{j=1}^{M-1} \sum_{k=1}^N 1(X_k \in S_j) \quad (17.45)$$

$$= \sum_{k=1}^N \sum_{j=1}^{M-1} 1(X_k \in S_j) \quad (17.46)$$

$$\leq \sum_{k=1}^N \left[\sum_{i=1}^M 1(X_k \in C_i) - 1 \right] \quad (17.47)$$

$$= \sum_{i=1}^M \sum_{k=1}^N 1(X_k \in C_i) - M \quad (17.48)$$

$$= \sum_{i=1}^M |C_i| - M. \quad (17.49)$$

Noting that the right-hand side is independent of T , and that the inequality in Eq. 17.47 is an equality if and only if T is a junction tree, we obtain the result. \square

17.11 The Hugin algorithm

The algorithm that we have developed in previous sections is known as the “Hugin algorithm,” an instance of the general junction tree framework. We summarize the algorithm here. There are five principal steps to the algorithm, the first of which applies only to directed graphs.

- **Moralization.** The moralization step converts a directed graph into an undirected graph. Nodes that have a common child are linked, and directed edges are converted to undirected edges. The local conditional probability of each node is multiplied onto the potential of a clique that contains the node and its parents.
- **Introduction of evidence.** Evidence is introduced by taking slices of the potentials.
- **Triangulation.** The graph is triangulated, using one of several possible algorithms. The potential of each clique of the original graph is multiplied onto the potential of a clique that contains the clique.
- **Construction of junction tree.** A junction tree is constructed by forming a maximal spanning tree from the cliques of the triangulated graph. Separators are introduced and their potentials are initialized to unity.

- **Propagation of probabilities.** Computation proceeds in the junction tree via the following update equations:

$$\phi_S^* = \sum_{V \setminus S} \psi_V \quad (17.50)$$

$$\psi_W^* = \frac{\phi_S^*}{\phi_S} \psi_W. \quad (17.51)$$

The updates must respect the Message-Passing Protocol. This can be achieved by designating a root node and calling `COLLECTEVIDENCE` and `DISTRIBUTEEVIDENCE` from the root. Once the algorithm terminates, the clique potentials and separator potentials are proportional to marginal probabilities. Further marginalization can be performed to obtain the probabilities of singleton nodes or other subsets.

17.12 The Shafer-Shenoy algorithm

There are a number of variations on the junction tree theme. All of these variations have at their core the notion of a triangulated graph and the junction tree property, but the way that propagation proceeds on the junction tree can be different. Some of these variations can provide additional insights into exact inference and provide different pathways for generalizations to approximate inference. Moreover, different variations on junction tree propagation can have different numerical properties or time/space properties. In this section we discuss one such variation—the Shafer-Shenoy algorithm.

The Shafer-Shenoy algorithm can be viewed as a variation on the junction tree framework in which no use is made of separator potentials. While the separator potentials have been useful in providing a simple mechanism for achieving consistency between neighboring cliques, and while we will encounter architectural examples in which separator potentials are particularly useful (cf. Section 18.2.4), there is a sense in which separator potentials are redundant (they are simply marginals of the clique potentials) and perhaps they can be disposed with.

Rather than focusing on separator potentials, let us instead focus on the ratios of separator potentials; the quantities that we referred to as “update factors” in our earlier presentation. Recall that in the second step of the message-passing calculation (Eq. 17.15), the clique potential is multiplied by the update factor. What we will show is that a propagation procedure can be based solely on the update factors.

Consider the pair of cliques C_i and C_j in Figure 17.15, with separator $S_{ij} = C_i \cap C_j$. We wish to exchange messages between these cliques so as to implement a junction tree algorithm, and we wish to do so without making use of a potential on the separator S_{ij} . To do so, define $\mu_{ij}(S_{ij})$ as the message sent from C_i to C_j .⁶ The Shafer-Shenoy algorithm tells us how to calculate $\mu_{ij}(S_{ij})$ based on the messages arriving at clique C_i from all cliques other than clique C_j :

$$\mu_{ij}(S_{ij}) = \sum_{C_i \setminus S_{ij}} \psi_{C_i} \prod_{k \neq i} \mu_{ki}(S_{ki}) \quad (17.52)$$

⁶Note that we are using the term “message” in a slightly more specific manner than before; for the Shafer-Shenoy algorithm, we equate “message” with the values $\mu_{ij}(S_{ij})$.

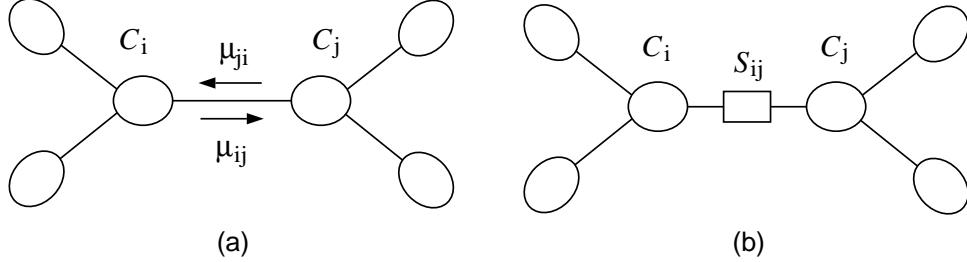


Figure 17.15: (a) A junction tree showing the messages μ_{ij} and μ_{ji} that are passed between cliques C_i and C_j . Note that both messages are functions of the separator S_{ij} . (b) A junction tree showing the separator explicitly.

Once clique C_i has received messages from all of its neighbors, we compute the marginal probability for C_i as follows:

$$p(C_i) \propto \psi_{C_i} \prod_k \mu_{ki}(S_{ki}). \quad (17.53)$$

Equations 17.52 and 17.53 constitute the Shafer-Shenoy algorithm. We now derive this algorithm from the point of view of our earlier junction tree algorithm, thereby proving the correctness of the implicit assertion in Eq. 17.53—that we do in fact obtain the marginal probabilities via this algorithm.

Consider now the pair of cliques C_i and C_j in Figure 17.15(b) with the explicit separator S_{ij} . The connection between the new algorithm and the earlier algorithm is made as follows. Define $\mu_{ij}(S_{ij})$ to be the update factor associated with the update of the link in the direction from C_i to C_j . That is, if the first update of this link proceeds in the i -to- j direction, let:

$$\mu_{ij}(S_{ij}) \triangleq \frac{\phi_{S_{ij}}^*}{\phi_{S_{ij}}^{**}}; \quad (17.54)$$

otherwise, let:

$$\mu_{ij}(S_{ij}) \triangleq \frac{\phi_{S_{ij}}^{**}}{\phi_{S_{ij}}^*}. \quad (17.55)$$

In either case, $\mu_{ij}(S_{ij})$ is the update factor arriving at clique C_j from clique C_i . Now note that the final potential at a given clique is the product of its initial potential and all of the update factors arriving from its neighbors. This immediately shows that Eq. 17.53 has the correct form. We have reduced our problem to that of establishing the correctness of Eq. 17.52.

We consider two cases. Suppose first that the initial update of the link between C_i and C_j occurs in the i -to- j direction. For this update to occur it must be the case that C_i has already received updates from all of its other neighbors (the Message-Passing Protocol). Thus at the moment when the update occurs, the value of the potential on C_i must be the product of its initial potential and the update factors from its neighbors C_k , for $k \neq j$. Let us assume (as an inductive hypothesis) that these update factors are correctly given by $\mu_{ki}(S_{ki})$, and consider the update factor that C_i

sends to C_j . From Eq. 17.14 we have:

$$\phi_{S_{ij}}^* = \sum_{C_i \setminus S_{ij}} \psi_{C_i} \prod_{k \neq i} \mu_{ki}(S_{ki}). \quad (17.56)$$

Comparing this with Eq. 17.52, we see that $\mu_{ij}(S_{ij}) = \phi_{S_{ij}}^*$ and, recalling that the initial value of the separator potential, $\phi_{S_{ij}}$, is unity, we have $\mu_{ij}(S_{ij}) = \phi_{S_{ij}}^* \phi_{S_{ij}}$ as required.

Now consider the case in which an earlier update has already occurred in the j -to- i direction. In this case, at the moment of the update from C_i to C_j , the potential on C_i must be the product of its initial potential and the update factors from *all* of its neighbors, including C_j . Thus, from Eq. 17.14 we have:

$$\phi_{S_{ij}}^{**} = \sum_{C_i \setminus S_{ij}} \psi_{C_i} \prod_k \mu_{ki}(S_{ki}) \quad (17.57)$$

$$= \sum_{C_i \setminus S_{ij}} \psi_{C_i} \mu_{ji}(S_{ji}) \prod_{k \neq j} \mu_{ki}(S_{ki}) \quad (17.58)$$

$$= \sum_{C_i \setminus S_{ij}} \psi_{C_i} \frac{\phi_{S_{ij}}^*}{\phi_{S_{ij}}} \prod_{k \neq j} \mu_{ki}(S_{ki}) \quad (17.59)$$

$$(17.60)$$

and this yields:

$$\frac{\phi_{S_{ij}}^{**}}{\phi_{S_{ij}}^*} = \sum_{C_i \setminus S_{ij}} \prod_{k \neq j} \mu_{ki}(S_{ki}), \quad (17.61)$$

where we again use the fact that $\phi_{S_{ij}} \equiv 1$. Comparing this result with Eq. 17.52, we see that $\mu_{ij}(S_{ij}) = \phi_{S_{ij}}^{**}/\phi_{S_{ij}}^*$ as required.

17.13 Computational complexity

In this section we discuss the computational complexity of the junction tree algorithm. For concreteness we focus on the Hugin algorithm and consider the computational complexity of the Shafer-Shenoy algorithm in the exercises.

It is important to distinguish between two phases of the junction tree algorithm. The first phase, which we will refer to as the *compilation phase*, involves moralization, triangulation and the maximal spanning tree algorithm. The second phase, the *propagation phase*, involves the introduction of evidence and message-passing on the junction tree.

The compilation phase is an “off-line” phase, occurring once for a given graphical model. The algorithms in the propagation phase are “on-line,” running each time a new set of conditional probabilities is desired.

Moralization is clearly a computationally tractable procedure. Letting N denote the number of nodes in the graph, and M the number of edges, moralization runs in time $O(N + M)$.

Moreover, the maximal spanning tree problem is computationally tractable. This is a well-studied problem and the computational complexity results are classical. In particular, the run time of Kruskal's algorithm is $O(N^2)$ and the run time of Prim's algorithm is $O(N^2)$.⁷

Let us turn to the triangulation problem. If we are not concerned with optimality (e.g., finding a junction tree with the smallest maximal clique, or the smallest number of edges), then finding a triangulation is computationally tractable. In particular, the run time of `UNDIRECTEDGRAPHELIMINATE` is easily seen to be $O(XXX)$. The problem of finding an optimal junction tree, however, is an NP-hard problem, under any of a number of definitions of optimality. We discuss this intractability result in more detail in Appendix A.

The fact that triangulation is an off-line phase of the junction tree algorithm tempers some of the concern that accompanies the NP-hardness result. Moreover, as we discuss in Appendix A, there are heuristic algorithms available for triangulation that perform reasonably well in empirical experiments. One may be willing to pay the cost of allowing one of these algorithms to run for a substantial time to obtain a good triangulation. Finally, it is important to be aware that for many graphical models the initial graph is sufficiently dense that even the optimal triangulation, if it could be found, would have a large number of edges or a large maximal clique size. It is the size of these cliques, which impacts the second phase of the junction tree algorithm, which is generally the key practical limitation in using the algorithm.

The second phase of the algorithm involves conditioning and message-passing. Conditioning is a straightforward procedure that simply annotates each clique with the indices that are to be held fixed in the slice corresponding to the conditioning variables. We therefore turn to the message-passing procedure.

Each step of the message-passing procedure involves the marginalization and rescaling of clique potentials. Let us suppose that these potentials are represented nonparametrically, as tables. This is a worst-case assumption, and specific parametric representations of the clique potentials may give more favorable complexity results. Marginalizing a table requires us to access each entry in the table, and thus the number of operations scales as the number of entries in the table. The number of such entries is exponential in the number of variables in the corresponding clique. This exponentiality is the key determinant of the computational complexity of the junction tree algorithm.

Rescaling a potential again involves accessing each entry in the affected clique potential, and thus is again exponential in the number of variables in the clique.

The number of cliques in a junction tree is no more than N , the number of nodes in the underlying graph (assuming that we use maximal cliques). Thus the number of separators is bounded above by $N - 1$, and we have at most $2N - 1$ messages flowing in a run of the Hugin algorithm. Each message involves two operations on clique potentials—a marginalization operation and a rescaling operation. In summary, a complete run of the Hugin algorithm involves at most $4N - 2$ such operations. Given that the size of a clique can be as large as the number of nodes N , the exponentiality of an individual marginalization or rescaling operation dominates the computational complexity.

It is of interest to compare the number of operations needed to obtain the marginal probabilities

⁷See, e.g., Cormen Leisherson, and Rivest (1990).

Figure 17.16: XXX

of all of the nodes in the graph—obtained via the junction tree algorithm—to the number of operations needed to obtain the marginal probabilities of a single node in the graph—obtained via the elimination algorithm. The latter algorithm is special case; just run `CollectEvidence`. Touches each potential once.

17.14 Generalized marginalization

One of the virtues of the junction tree framework is its clear distinction between the graph-theoretic and the algebraic machinery involved in probabilistic inference. The algebraic machinery that we utilized in deriving the algorithm was elementary—our proofs reposed on the associative, commutative and distributive laws of arithmetic. As we discuss in this section, if we replace the specific algebraic operators that we used with other operators that obey these same laws, we find that the junction tree framework extends readily to a wide class of other problems involving factorized algebraic expressions, of which probabilistic inference is a special case.

17.14.1 Maximum probability configurations

In Section ?? we discussed the Viterbi algorithm for hidden Markov models. Given an observation sequence, this algorithm returns a single configuration of the hidden states that has maximal probability. In this section we describe a “generalized Viterbi algorithm” that computes most probable configurations for arbitrary graphical models.

That we need to do essentially no additional work to derive such an algorithm is suggested by returning to the example in Figure 17.16. Let us find a set of values of the nodes—a *configuration*—that maximizes the joint probability $p(x_1, x_2, \dots, x_6)$.

The first few steps of the calculation are as follows:

$$\begin{aligned} \max_x p(x) &= \max_{x_1} \max_{x_2} \max_{x_3} \max_{x_4} \max_{x_5} \max_{x_6} p(x_1)p(x_2 | x_1)p(x_3 | x_1)p(x_4 | x_2)p(x_5 | x_3)p(x_6 | x_2, x_5) \\ &= \max_{x_1} p(x_1) \max_{x_2} p(x_2 | x_1) \max_{x_3} p(x_3 | x_1) \max_{x_4} p(x_4 | x_2) \max_{x_5} p(x_5 | x_3) \max_{x_6} p(x_6 | x_2, x_5). \end{aligned}$$

Computing the maximum of $p(x_6 | x_2, x_5)$ with respect to x_6 yields an intermediate factor that is a function of x_2 and x_5 . This factor is then retained until needed in a subsequent maximization, in this case the maximization over x_5 .

The sequence of steps continue in an identical manner to those that we carried out in our development of the elimination algorithm in Chapter 3. Clearly, from a symbolic point of view, the computation is the same. In particular, the graphical consequences of the maximization operator are identical to those of our earlier calculations with the summation operator.

In the light of this example, let us consider replacing “sum” with “max” in the junction tree algorithm. The only step in the algorithm that specifically refers to summation is the marginalization

step in Eq. (17.14). Changing this step to maximization, we have:

$$\phi_S^* = \max_{V \setminus S} \psi_V \quad (17.62)$$

$$\psi_W^* = \frac{\phi_S^*}{\phi_S} \psi_W, \quad (17.63)$$

where the rescaling step is unchanged.

In essence we now obtain an inference algorithm based on a generalized notion of marginalization. All of the steps that we took in deriving the junction tree algorithm go through as before, given that the maximization operator has the same commutativity and associativity properties as summation, and given that maximization distributes over multiplication just as summation distributes over multiplication.

What do we obtain from this algorithm? Recall that our key result is that contained in Theorem 2, where we showed that at the end of the junction tree procedure, each clique potential is equal to its marginal probability. Here “marginal” means that the random variables not contained in the clique have been “summed out.” If we replace summation by maximization, we obtain the same result, but now “marginal” means that the random variables not contained in the clique have been “maximized out.” Thus, we must have

$$\psi_C(x_C) = \max_{V \setminus C} p(x). \quad (17.64)$$

We interpret the resulting entries in the clique potential as containing the values of the maximal probability attainable for each possible configuration of the random variables X_C . Maximizing over these values, we obtain the actual configuration

We could also take one or more of the variables to be evidence variables and maximize the conditional probability distribution of the remaining variables; this would simply involve holding the evidence variables fixed.

17.14.2 Appendix A. Decomposable \equiv Triangulated \equiv Junction tree

In Section 17.8 we showed that all triangulated graphs possess a junction tree. For the purpose of devising an inference algorithm, this result suffices, focusing our attention on the problem of finding a triangulation of a graph. It is of interest to know, however, that in a certain sense triangulation is not merely a means to an end, but rather triangulation is forced on us if we wish to avail ourselves of the junction tree property. In particular, in this Appendix we strengthen our earlier result and show that a graph has a junction tree if and only if the graph is triangulated.

We also show that these two properties are equivalent to a third property—*decomposability*. Recall from Section ?? that a graph is *decomposable* if it can be recursively subdivided into sets A , B and S , where S separates A and B , and where S is complete. The equivalence of decomposability and the junction tree property provides an appealing interpretation of the junction tree algorithm as a divide-and-conquer algorithm.

Theorem 6 *All decomposable graphs are triangulated.*

Proof. We prove the result by induction. The base case of a single node is trivial. We assume that the result holds for N or fewer nodes and consider a graph with $N + 1$ nodes.

If the graph is complete then it is obviously triangulated. Otherwise, the definition of decomposability implies a decomposition of the graph into sets A , B , and S such that S is complete and S separates A and B . Also, both $A \cup S$ and $B \cup S$ are decomposable. By the induction hypothesis there are no chordless cycles in either $A \cup S$ or $B \cup S$. The only possible chordless cycles must therefore include one or more nodes in both A and B . But such cycles must pass twice through S , and the completeness of S implies that they have a chord. \square

Theorem 6 and Theorem 3 together show that all decomposable graphs have a junction tree. We have proved the correctness of the junction tree algorithm for the class of decomposable graphs.

We now show a stronger result, namely that decomposability, triangulation and the junction tree property are equivalent. This implies that the junction tree algorithm is correct *only* for the class of decomposable graphs.

Theorem 7 *The following are equivalent characterizations of an undirected graph \mathcal{G} :*

- (D) \mathcal{G} is decomposable.
- (T) \mathcal{G} is triangulated.
- (J) \mathcal{G} has a junction tree.

Proof. We have already shown that (D) implies (T) implies (J). Thus we can prove the theorem by showing that (J) implies (D).

The proof is a proof by induction. In the base case \mathcal{G} has a single clique and is decomposable by definition. Suppose that the theorem holds for junction trees with N or fewer cliques and consider a junction tree T for $\mathcal{G} = (X, E)$ with $N + 1$ cliques.

Let C be a leaf node in T with separator S . Define $R = C \setminus S$ (recall Figure 17.13). Consider the disjoint sets R , $X \setminus C$ and S . We show that these sets are a decomposition of \mathcal{G} .

Lemma 1 implies that S separates R and $X \setminus C$. That S is complete follows from the fact that S is the intersection of a pair of cliques.

To show that \mathcal{G} is decomposable it remains to show that $C = R \cup S$ and $X \setminus R = (X \setminus C) \cup S$ are decomposable. We show that both subsets have junction trees and conclude by the induction hypothesis that they are decomposable.

That C has a junction tree follows immediately because it is a single clique.

Consider the effect on the junction tree T of the removal of nodes in R . Each node in R is contained only in C and its neighbors are therefore fully connected (i.e., nodes in R are simplicial). Removing any such node therefore leaves the remaining nodes in C fully connected, and thus C remains a clique and the junction tree T is unaltered. When all nodes in R have been removed, all that remains of C is the separator S , which is a subset of the neighboring clique in T . By simply pruning the C clique and its separator S from T we therefore obtain a junction tree for $X \setminus R$. \square

17.15 Historical remarks and bibliography

Chapter 18

The HMM and State Space Model Revisited

In this chapter we revisit the Hidden Markov model and the Linear Gaussian model from the more general point of view of the previous two chapters. It is illuminating to see the relationship between the recursive algorithms that we developed in Chapters 12 and 15 and the general junction tree constructions in Chapter 17.

18.1 Hidden Markov models

Recall that the Hidden Markov model (HMM) can be represented as the chain-structured graphical model shown in Figure 18.1(a). The multinomial *state variables* q_t form the backbone of the model; from this backbone hang the observable *output variables* y_t .

We parameterize the model by endowing the first node with an *initial probability* π , where $\pi_i \triangleq P(q_1^i = 1)$, and each subsequent state node with a *transition matrix* A , where $a_{ij} \triangleq P(q_{t+1}^j = 1 | q_t^i = 1)$. The output nodes are assigned the local conditional probability $P(y_t | q_t)$. For concreteness

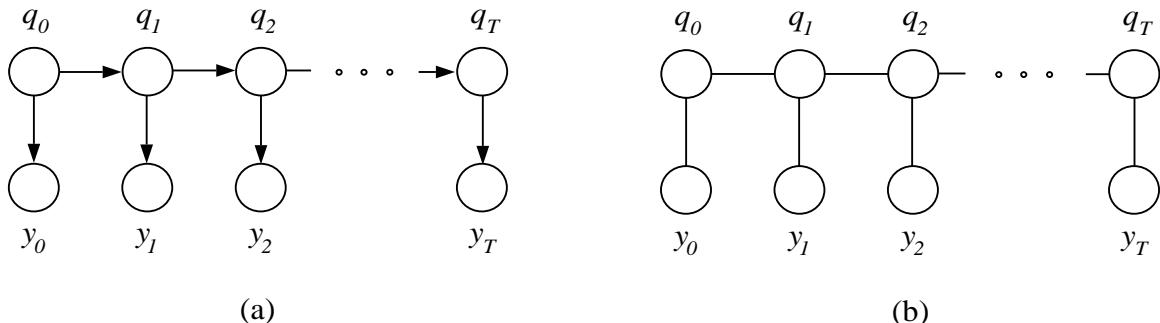


Figure 18.1: (a) The representation of a HMM as a graphical model. (b) The moralized, triangulated HMM graph.

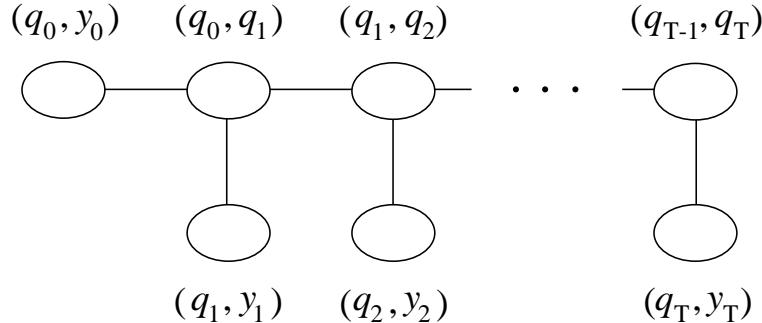


Figure 18.2: A maximal spanning tree on the cliques of the HMM graph.

we will assume that y_t is a multinomial node, so that $P(y_t|q_t)$ can be viewed as a matrix B , where $b_{ij} \triangleq P(y_t^j = 1|q_t^i = 1)$. However, given that y_t is always observed, this assumption will play no critical role in our discussion.

To convert the HMM into a junction tree, we proceed as outlined in the previous chapter. The moralization step is vacuous in this case, given that each node has at most a single parent. Moreover, the triangulation step is also vacuous, given that the graph has no cycles. We are left with the moralized, triangulated graph shown in Figure 18.1(b).

The cliques in the graph in Figure 18.1(b). are given by the pairs (q_t, q_{t+1}) and (q_t, y_t) . There are several ways to connect these pairs so as to obtain a maximal spanning tree; with a bit of foresight we choose the maximal spanning tree shown in Figure 18.2.¹ This then is our junction tree.

Figure 18.3 shows the junction tree in which the separator sets have been made explicit and the potentials have been labeled. We make the following choice for the assignment of local conditional probabilities to potentials. The initial probability $P(q_0)$ as well as the conditional prob-

¹In Exercise XXX we ask the reader to explore some of the alternative inference algorithms generated by the alternative choices of maximal spanning tree.

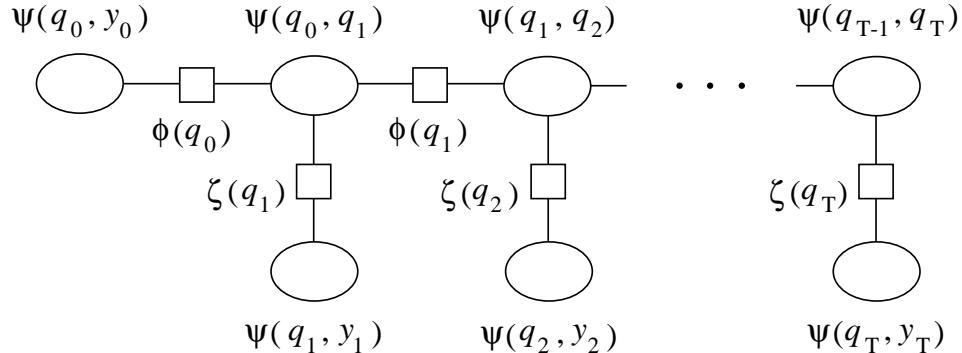


Figure 18.3: A junction tree for the HMM with the potentials labeled.

ability $P(y_0|q_0)$ is assigned to the potential $\psi(q_0, y_0)$. Note that this implies that this potential is initially set to the marginal $P(q_0, y_0)$. The state-to-state potentials are given the assignment $\psi(q_t, q_{t+1}) = P(q_{t+1}|q_t)$; note that these are conditional probabilities rather than marginals. Finally, the remaining output probabilities $P(y_t|q_t)$ are assigned to the potentials $\psi(q_t, y_t)$ and all separator potentials are initialized to one.

18.1.1 Unconditional inference

It is instructive to consider running the junction tree inference algorithm before any evidence has been observed. Suppose that we designate the node (q_{T-1}, q_T) as the root and collect to the root.

Consider first the operation of passing a message upward from a clique (q_t, y_t) to its neighbor (q_{t-1}, q_t) , for $t > 1$. The marginalization operation in this case yields $\sum_{y_t} \psi(q_t, y_t) = \sum_{y_t} P(y_t|q_t) = 1$; thus the separator potential $\zeta^*(q_t)$ remains set at one. This implies that the update factor $\zeta^*(q_t)\zeta(q_t)$ is one, and thus the potential $\psi(q_{t-1}, q_t)$ remains unchanged. In general, the messages that are passed upward from the leaves (q_t, y_t) have no effect when no evidence is observed.

Now consider the message from (q_0, y_0) to (q_0, q_1) (see Figure 18.3). We have:

$$\phi^*(q_0) = \sum_{y_0} \psi(q_0, y_0) = \sum_{y_0} P(q_0, y_0) = P(q_0) \quad (18.1)$$

$$\psi^*(q_0, q_1) = \psi(q_0, q_1)\phi^*(q_0) = P(q_1|q_0)P(q_0) = P(q_0, q_1). \quad (18.2)$$

This transformation propagates forward along the chain, changing the separator potentials on q_t into the marginals $P(q_t)$ and the clique potentials on (q_t, q_{t+1}) into the marginals $P(q_t, q_{t+1})$. Thus, all potentials along the backbone of the chain become marginals.

A subsequent pass of DistributeEvidence will have no effect on the potentials along the backbone of the chain (as the reader can verify), but it will convert the potentials $\zeta(q_t)$ into marginals $P(q_t)$ and the potentials $\psi(q_t, y_t)$ into marginals $P(q_t, y_t)$. Thus all potentials throughout the junction tree become marginal probabilities. This result is not surprising, given that it is an easy special case of Theorem 2, but it is reassuring.

Our result also helps to clarify the representation of the joint probability as the product of the clique potentials divided by the product of the separator potentials (cf. Eq. 17.11). While we would not expect to be able to represent the joint in general as the product of marginals such as $P(q_t, q_{t+1})$, we do get this representation if we divide by the separator potentials and those separator potentials are also marginals. Thus, for example, each pairing of a clique potential and a separator potential along the backbone contributes a factor $P(q_t, q_{t+1})/P(q_t)$ to the joint, which is nothing but the original local conditional $P(q_{t+1}|P(q_t))$.

18.1.2 Introducing evidence

We now suppose that the outputs y are observed. We wish to calculate the likelihood $P(y)$ as well as marginal posterior probabilities such as $P(q_t|y)$ and $P(q_t, q_{t+1}|y)$. We return to the original junction tree in which the separator potentials are initialized to unity.

The first step is to alter the potentials to reflect the introduction of the evidence. In the case that y_t is a multinomial node, recall that the potential $\psi(q_t, y_t)$ can be viewed as a matrix B ,

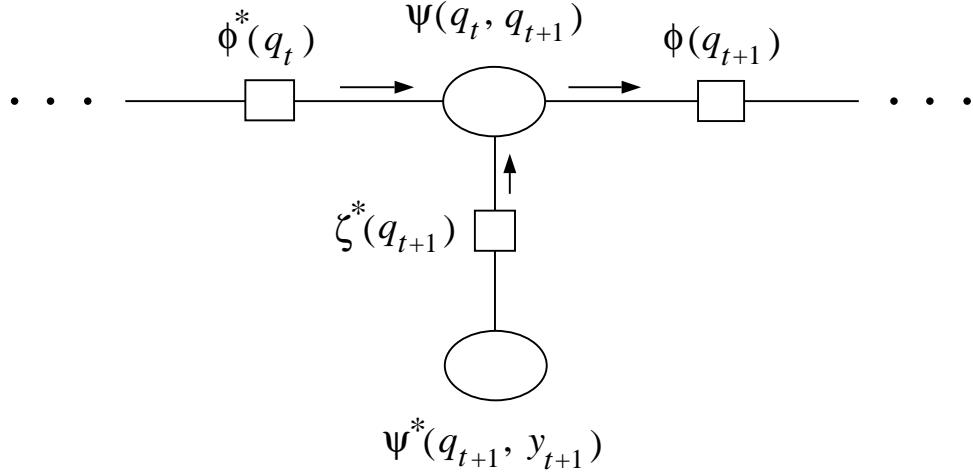


Figure 18.4: A fragment of the junction tree for the HMM.

with columns labeled by the possible values of y_t . Conceptually, observing y_t to be in its i th state corresponds to setting all other columns to zero, so that a subsequent marginalization over y_t simply picks out the i th column of the matrix. In practice we would simply set the separator potential to the desired column. Thus, we have:

$$\zeta^*(q_t) = P(y_t|q_t), \quad (18.3)$$

where y_t is viewed as a fixed constant.

18.1.3 Collecting to the root

We designate the clique (q_{T-1}, q_T) as the root of the junction tree and collect to the root.

Consider the update of clique (q_t, q_{t+1}) , as shown in Figure 18.4. We suppose that the preceding separator potential $\phi^*(q_t)$ has already been updated and consider the computation of $\psi^*(q_t, q_{t+1})$ and $\phi^*(q_{t+1})$. Combining the update of clique (q_t, q_{t+1}) based on both of its neighbors (q_{t-1}, q_t) and (q_{t+1}, y_{t+1}) , we have:

$$\psi^*(q_t, q_{t+1}) = \psi(q_t, q_{t+1})\phi^*(q_t)\zeta^*(q_{t+1}) \quad (18.4)$$

$$= a_{q_t, q_{t+1}}\phi^*(q_t)P(y_{t+1}|q_{t+1}), \quad (18.5)$$

where, as in Chapter 12, we utilize the shorthand $a_{q_t, q_{t+1}} \triangleq P(q_{t+1}|q_t)$. Proceeding forward along the chain, we obtain:

$$\phi^*(q_{t+1}) = \sum_{q_t} \psi^*(q_t, q_{t+1}) \quad (18.6)$$

$$= \sum_{q_t} a_{q_t, q_{t+1}}\phi^*(q_t)P(y_{t+1}|q_{t+1}). \quad (18.7)$$

Defining $\alpha(q_t) \triangleq \phi^*(q_t)$, we see that we have recovered exactly the alpha algorithm from Chapter 12 (cf. Eq. 12.22).

Although this definition of $\phi^*(q_t)$ achieves a formal equivalence of the update formulas, is it a reasonable definition? Is $\phi^*(q_t)$ equal to $P(y_0, \dots, y_t, q_t)$? Indeed, we have $\phi^*(q_0) = P(y_0, q_0)$ by definition, and recursively:

$$\phi^*(q_{t+1}) = \sum_{q_t} a_{q_t, q_{t+1}} \phi^*(q_t) P(y_{t+1} | q_{t+1}) \quad (18.8)$$

$$= \sum_{q_t} P(q_{t+1} | q_t) P(y_0, \dots, y_t, q_t) P(y_{t+1} | q_{t+1}) \quad (18.9)$$

$$= \sum_{q_t} P(y_0, \dots, y_t, y_{t+1}, q_t, q_{t+1}) \quad (18.10)$$

$$= P(y_0, \dots, y_t, y_{t+1}, q_{t+1}), \quad (18.11)$$

so the definition is justified.

It is also possible to develop an alternative approach to the forward inference problem by specifying a recurrence on the (q_t, q_{t+1}) clique potentials. In fact, given that $\phi^*(q_t) = \sum_{q_{t-1}} \psi^*(q_{t-1}, q_t)$, substitution in Eq. 18.5 yields:

$$\psi^*(q_t, q_{t+1}) = a_{q_t, q_{t+1}} \sum_{q_{t-1}} \psi^*(q_{t-1}, q_t) P(y_{t+1} | q_{t+1}), \quad (18.12)$$

The reader can verify that $\psi^*(q_t, q_{t+1}) = P(y_0, \dots, y_{t+1}, q_t, q_{t+1})$. Thus, defining the variable $\rho(q_t, q_{t+1}) \triangleq P(y_0, \dots, y_{t+1}, q_t, q_{t+1})$, we have established the recurrence relation:

$$\rho(q_t, q_{t+1}) = a_{q_t, q_{t+1}} \sum_{q_{t-1}} \rho(q_{t-1}, q_t) P(y_{t+1} | q_{t+1}), \quad (18.13)$$

which is an alternative to the traditional alpha algorithm. Indeed, in Section 18.1.5 we will establish a backward recurrence involving the cliques (q_t, q_{t+1}) which, together with Eq. 18.13 will yield an alternative approach to HMM inference that produces the “xi” variables directly and the “alpha/gamma” variables indirectly.

The collect phase of the algorithm terminates with the update of $\psi(q_{T-1}, q_T)$. The updated potential will equal $P(y_0, \dots, y_T, q_t, q_{t+1})$, and thus by marginalization:

$$P(y) = \sum_{q_{T-1}, q_T} \psi^*(q_{T-1}, q_T) \quad (18.14)$$

we obtain the likelihood $P(y)$.

18.1.4 An alternative root

Suppose that instead of designating clique (q_{T-1}, q_T) as the root node of the junction tree, we instead utilize (q_0, q_1) as the root. Exercise XXX studies this case, showing that the result is the

beta algorithm. Thus, we find that $\phi^*(q_t) = P(y_{t+1}, \dots, y_T | q_t)$ and, moreover, the junction tree recursion linking $\phi^*(q_t)$ and $\phi^*(q_{t+1})$ is exactly the beta recursion of Eq. 12.30.

It is not necessary, however, to change the root of the junction tree to derive the beta algorithm. As we show in Section 18.1.5, the beta algorithm arises during the DistributeEvidence pass when utilizing clique (q_{T-1}, q_T) as the root. This is the preferred way to map the traditional HMM inference algorithms onto the junction tree machinery.

A final comment regarding the forward/backward algorithms and the notion of *filtering*; that of obtaining the conditional expectation of a state given a partial observation sequence. Note that the beta variables are conditionals $P(y_{t+1}, \dots, y_T | q_t)$, involving the probability of a partial evidence sequence given the state. The alpha variables, on the other hand, are marginals $P(y_0, \dots, y_t, q_t)$, involving the probability of a partial evidence sequence *and* the state at time t . Converting the latter variables to filtered quantities of the form $P(q_t | y_0, \dots, y_t)$ is straightforward; one simply normalizes. Converting the beta variables to (backward) filtered quantities of the form $P(q_t | y_{t+1}, \dots, y_T)$, on the other hand, is not so straightforward. Obtaining this conditional requires us to know $P(q_t)$, which is not available in the junction tree.

Suppose, however, that we consider a pre-initialized junction tree in which the inference algorithm has been performed without evidence. We saw in Section 18.1.1 that this procedure converts the potentials throughout the tree, including the separator potentials, into marginal probabilities. Thus, a marginal such as $P(q_t)$ is available in this tree, and we might expect to obtain a marginal version of the beta algorithm if we subsequently collect to the root. Indeed, in Exercise XXX we verify that this procedure yields $\phi^*(q_t) = P(y_{t+1}, \dots, y_T, q_t)$. This quantity is readily converted to the filtered estimate $P(q_t | y_{t+1}, \dots, y_T)$ by normalization.

18.1.5 Distributing from the root

We now return to our main thread, and consider running the DistributeEvidence algorithm from the root (q_{T-1}, q_T) . We assume that we have already collected to this root, and thus the potentials at the outset of the DistributeEvidence are those discussed in Section 18.1.3.

The DistributeEvidence phase proceeds backward along the backbone of the state-to-state cliques as well as downward into the state-to-output cliques. Given that we have already obtained the likelihood, which is the only information regarding the probability of the outputs that is generally of interest, we restrict our attention to the updates along the backbone.

Referring to Figure 18.5, we suppose that the preceding separator potential $\phi^{**}(q_{t+1})$ has already been updated and consider the update of $\psi^{**}(q_t, q_{t+1})$ and $\phi^{**}(q_t)$. We have:

$$\psi^{**}(q_t, q_{t+1}) = \psi^*(q_t, q_{t+1}) \frac{\phi^{**}(q_{t+1})}{\phi^*(q_{t+1})}, \quad (18.15)$$

and proceeding backward a further step:

$$\begin{aligned} \phi^{**}(q_t) &= \sum_{q_{t+1}} \frac{\psi^*(q_t, q_{t+1})}{\phi^*(q_{t+1})} \phi^{**}(q_{t+1}) \\ &= \sum_{q_{t+1}} \frac{\psi^*(q_t, q_{t+1})}{\sum_{q_t} \psi^*(q_t, q_{t+1})} \phi^{**}(q_{t+1}). \end{aligned} \quad (18.16)$$

$$(18.17)$$

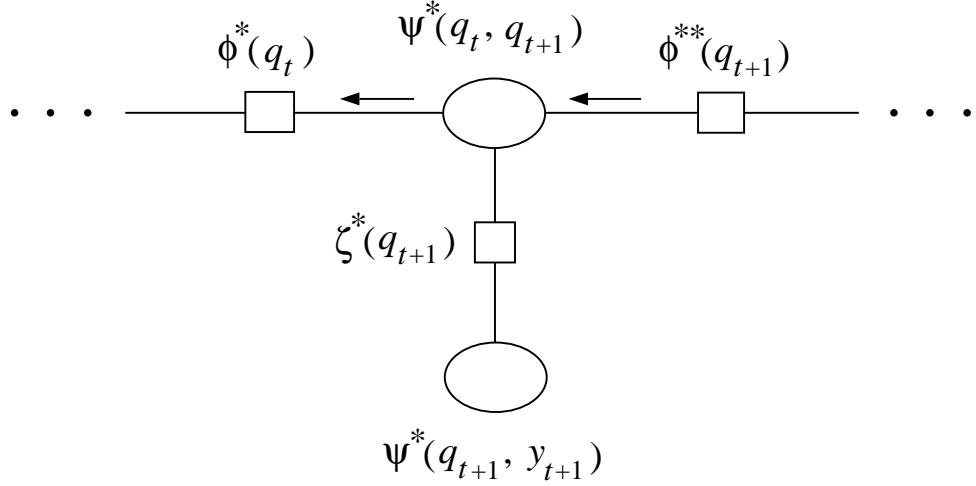


Figure 18.5: A fragment of the junction tree for the HMM with backward-going messages.

Substituting from Eq. 18.5 we have:

$$\phi^{**}(q_t) = \sum_{q_{t+1}} \frac{a_{q_t, q_{t+1}} \phi^*(q_t)}{\sum_{q_t} a_{q_t, q_{t+1}} \phi^*(q_t)} \phi^{**}(q_{t+1}) \quad (18.18)$$

$$= \sum_{q_{t+1}} \frac{a_{q_t, q_{t+1}} \alpha(q_t)}{\sum_{q_t} a_{q_t, q_{t+1}} \alpha(q_t)} \phi^{**}(q_{t+1}). \quad (18.19)$$

Defining $\gamma(q_t)$ as equal to $\phi^{**}(q_t)$, up to a constant of proportionality, we see that we have recovered the gamma recursion from Chapter 12 (cf. Eq. 12.41).

Once again we can reassure ourselves that $\gamma(q_t)$ defined this way is indeed proportional to the posterior probability $P(q_t | y_0, \dots, y_T)$. This can be done by direct calculation (cf. Exercise XXX). Alternatively, we can trust the general theory of Chapter 17, which assures us that the potentials produced by the junction tree algorithm must be proportional to the posterior probabilities. Indeed, it is easy to verify that $\phi^{**}(q_t) = P(y_0, \dots, y_T, q_t)$ and thus the proportionality constant is the likelihood $P(y)$. Once the junction tree algorithm has run, we can obtain the likelihood by normalizing any of the potentials.

In summary, we have identified the alpha and the gamma variables in the junction tree algorithm, and have derived the “alpha-gamma” recursion discussed in Chapter 12. We can also derive the “alpha-beta” recursion via the junction tree algorithm. Consider in particular the update factor $\phi^{**}(q_t)/\phi^*(q_t)$:

$$\frac{\phi^{**}(q_t)}{\phi^*(q_t)} = \frac{\sum_{q_{t+1}} \psi^{**}(q_t, q_{t+1})}{\phi^*(q_t)} \quad (18.20)$$

$$= \sum_{q_{t+1}} \frac{\psi^*(q_t, q_{t+1})}{\phi^*(q_t)} \frac{\phi^{**}(q_{t+1})}{\phi^*(q_{t+1})} \quad (18.21)$$

$$= \sum_{q_{t+1}} \frac{a_{q_t, q_{t+1}} \phi^*(q_t) P(y_{t+1} | q_{t+1})}{\phi^*(q_t)} \frac{\phi^{**}(q_{t+1})}{\phi^*(q_{t+1})} \quad (18.22)$$

$$= \sum_{q_{t+1}} a_{q_t, q_{t+1}} P(y_{t+1} | q_{t+1}) \frac{\phi^{**}(q_{t+1})}{\phi^*(q_{t+1})}. \quad (18.23)$$

where we have used Eq. 18.5 in the third equality. Defining $\beta(q_t) \triangleq \phi^{**}(q_t)/\phi^*(q_t)$, we have recovered the beta recursion from Chapter 12 (cf. Eq. 12.30). Moreover, putting together our definitions, we have:

$$\alpha(q_t) \beta(q_t) = \phi^*(q_t) \frac{\phi^{**}(q_t)}{\phi^*(q_t)} = \phi^{**}(q_t) \propto \gamma(q_t) \quad (18.24)$$

and thus we have also recovered the original definition of $\gamma(q_t)$ in Eq. 12.13.

Finally, it is also of interest to note that the junction tree algorithm gives us an explicit recursion in the variables $\xi(q_t, q_{t+1})$ (cf. Eq. 12.42). Starting from Eq. 18.15, we have:

$$\psi^{**}(q_{t-1}, q_t) = \psi^*(q_{t-1}, q_t) \frac{\phi^{**}(q_t)}{\phi^*(q_t)} \quad (18.25)$$

$$= \frac{\psi^*(q_{t-1}, q_t)}{\phi^*(q_t)} \sum_{q_{t+1}} \psi^{**}(q_t, q_{t+1}) \quad (18.26)$$

$$= \frac{\rho(q_{t-1}, q_t)}{\sum_{q_{t-1}} \rho(q_{t-1}, q_t)} \sum_{q_{t+1}} \psi^{**}(q_t, q_{t+1}). \quad (18.27)$$

Defining $\xi(q_t, q_{t+1})$ to be equal to $\psi^{**}(q_t, q_{t+1})$, again up to a constant of proportionality which turns out to be the likelihood, we have obtained an explicit recursion for the $\xi(q_t, q_{t+1})$ variables:

$$\xi(q_{t-1}, q_t) = \frac{\rho(q_{t-1}, q_t)}{\sum_{q_{t-1}} \rho(q_{t-1}, q_t)} \sum_{q_{t+1}} \xi(q_t, q_{t+1}). \quad (18.28)$$

This “rho-xi algorithm” is the analog of the “alpha-gamma algorithm.”

Once the rho-xi algorithm has run, $\alpha(q_t)$ and $\gamma(q_t)$ can be obtained via:

$$\alpha(q_t) = \sum_{q_{t-1}} \rho(q_{t-1}, q_t) \quad (18.29)$$

$$\gamma(q_t) = \sum_{q_{t-1}} \xi(q_{t-1}, q_t), \quad (18.30)$$

which are verified by plugging in the corresponding junction tree definitions.

18.2 Linear Gaussian models

We now turn to the linear Gaussian model (the “LG-HMM”). We rederive two of the LG-HMM inference algorithms—a forward (filtering) algorithm and a backward (smoothing) algorithm—from Chapter 15 in order to exemplify the relationships between the junction tree and the earlier material.

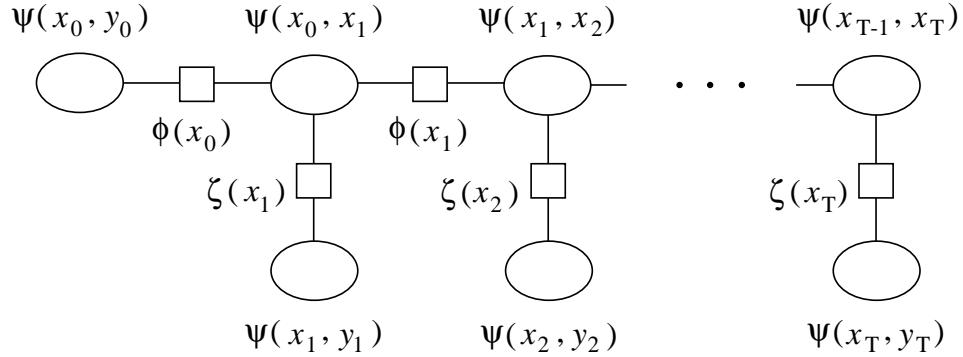


Figure 18.6: A junction tree for the LG-HMM with the potentials labeled.

Recall from Chapter 15 that the LG-HMM has the same graphical structure as the HMM, the difference being the node types and the parameterization. Thus, as before, Figure 18.1(a) is the graphical model for the LG-HMM (substituting “ x ” for “ q ”) and Figure 18.1(b) is the moralized, triangulated graph. Also, the junction tree for the LG-HMM, shown in Figure 18.6, is identical to that for the HMM.² Figure 18.6

The potentials in the junction tree are Gaussian potentials on pairs of nodes (the cliques) or singletons (the separators). We set up an assignment of local conditional probabilities to clique potentials that parallels that of the HMM. In particular, clique (x_0, y_0) is assigned the initial state probability $P(x_0)$ and the conditional $P(y_0|x_0)$, and thus $\psi(x_0, y_0)$ is initialized to the marginal $P(x_0, y_0)$. All of the other clique potentials are conditionals; in particular the state-to-state clique potential $\psi(x_t, x_{t+1})$ is set equal to $P(x_{t+1}|x_t)$ and the state-to-output potential $\psi(x_t, y_t)$ is set equal to $P(y_t|x_t)$.

Consider first the potential $\psi(x_t, x_{t+1})$. Given the dynamical equation:

$$x_{t+1} = Ax_t + Gw_t, \quad (18.31)$$

the conditional probability $P(x_{t+1}|x_t)$ is a multivariate Gaussian with mean Ax_t and covariance matrix GQG^T , where Q is the covariance matrix of w_t . Letting $H \triangleq GQG^T$, we have:

$$P(x_{t+1}|x_t) \propto \exp \left\{ -\frac{1}{2}(x_{t+1} - Ax_t)^T H^{-1}(x_{t+1} - Ax_t) \right\}. \quad (18.32)$$

Expressing the quadratic form as a function of two variables, we obtain:

$$\psi(x_{t+1}|x_t) = \exp \left\{ -\frac{1}{2} \begin{pmatrix} x_t \\ x_{t+1} \end{pmatrix}^T \begin{bmatrix} A^T H^{-1} A & -A^T H^{-1} \\ H^{-1} A & H^{-1} \end{bmatrix} \begin{pmatrix} x_t \\ x_{t+1} \end{pmatrix} \right\} \quad (18.33)$$

Note that in this representation, and in all subsequent potential function representations of Gaussians, we carry along only the exponential factor and ignore the Gaussian normalization factor.

²As in the case of the HMM, it is a worthwhile exercise to investigate the algorithms that arise from alternative choices for the junction tree.

The reason for this is as follows. At the end of the junction tree algorithm we are guaranteed that the potentials are equal to marginal probabilities, up to a normalization factor. Now, all of the potentials in the LG-HMM are Gaussian, both before and after the running of the inference algorithm. This implies that we can simply read off the normalization factors from the final form of the potentials; we are not required to explicitly normalize before or during the inference procedure.

From the output equation of the LG-HMM:

$$y_t = Cx_t + v_t, \quad (18.34)$$

we have that $P(y_t|x_t)$ is a multivariate Gaussian with mean Cx_t and covariance matrix R , where R is the covariance matrix of v_t . This yields:

$$P(y_t|x_t) \propto \exp \left\{ -\frac{1}{2}(y_t - Cx_t)^T R^{-1} (y_t - Cx_t) \right\}. \quad (18.35)$$

We can proceed as before and convert this to a bivariate representation for $\psi(x_t, y_t)$. Note, however, that y_t is an observed constant in applications of the LG-HMM. This implies that the marginalization step that yields $\zeta^*(x_t)$ simply reproduces Eq. 18.35 for that observed value of y_t (i.e., we integrate $\psi(x_t, y_t)$ against a delta function). We thus leave Eq. 18.35 unexpanded in anticipation of its later role as $\zeta^*(x_t)$.

Multivariate Gaussian potentials can be represented in terms of either moments (μ, Σ) , or canonical parameters (ξ, Λ) . In Chapter 13 we derived the formulas for marginalization and conditioning in both representations. In the context of the junction tree algorithm we also need to multiply potentials. This is significantly easier in the canonical parameterization. Thus, if we have potentials $\psi_1(x) = \exp \xi_1^T x - 1/2x^T K_1 x$ and $\psi_2(x) = \exp \xi_2^T x - 1/2x^T K_2 x$, the product is a potential $\psi(x) = \exp \xi^T x - 1/2x^T K x$, where:

$$K = K_1 + K_2 \quad (18.36)$$

$$\xi = \xi_1 + \xi_2. \quad (18.37)$$

Expressing this operation in terms of moments requires an application of the inverse matrix lemma. On the other hand, if our interest is in obtaining filtered or smoothed estimates of the states—i.e., moments—then we if we develop an algorithm in terms of canonical parameters we will require an application of the inverse matrix lemma at the end. The situation is “pay now or pay later.”

Moreover, in the context of linear Gaussian systems, moments behave particularly nicely and calculations on moments can save substantial labor. If we take the junction tree algorithm literally and require ourselves to multiply the Gaussian potentials then we miss out on this opportunity. Indeed, in doing the necessary inverse matrix operations we are essentially rederiving the fact that the parameters μ and Σ are moments of the Gaussian distribution. It is also possible to replace the multiplication step of the junction tree algorithm with a step that calculates the moments of the product directly, utilizing the probabilistic interpretation of the potentials. For example, suppose that we know that $\phi(x)$ is proportional to the marginal of x and we have that $\psi(x, y)$ is proportional to the conditional of y given x . Moreover, let $y = Cx + v$. In this case we can directly compute $E[y]$, $\text{Var}[y]$ and $\text{Cov}[x, y]$ in terms of the moment parameterization of $\phi(x)$, thereby obtaining the moment parameterization of $\psi^*(x, y) = \psi(x, y)\phi(x)$.

In this chapter we generally take the more literal interpretation of the junction tree algorithm and work in the canonical representation (with the notable exception of the following section, where we exploit moment calculations). Exercise XXX asks the reader to develop the moment-based approach more generally, in particular relating this approach to the derivation of the Kalman filter that we presented in Chapter 15.

18.2.1 Unconditional inference

As in the HMM case it is useful to consider running the junction tree inference algorithm before any evidence has been observed. Suppose that we designate the node (x_{T-1}, x_T) as the root and collect to the root.

The derivation that we carried out in Section 18.1.1 for the HMM was entirely generic, and (replacing the sums with integrals) we obtain the same results for the LG-HMM. In particular, once again the operation of passing a message upward from the leaves (x_t, y_t) to the cliques (x_t, x_{t+1}) , for $t > 1$, has no effect. The operation of passing messages from (x_0, y_0) to (x_0, x_1) and subsequently along the backbone of cliques (x_t, x_{t+1}) has the effect of changing all of those clique potentials, as well as the separator potentials, to marginal probabilities.

The form that these marginal probabilities take is readily obtained via moment calculations. Let (μ_t, Σ_t) denote the mean and covariance matrix of x_t . Given the dynamical equation $x_{t+1} = Ax_t + Gw_t$, and given our assumption that the initial state has mean zero, we see that $\mu_t = 0$ for all t . As for the covariance matrix, we obtain:

$$\Sigma_{t+1} = A\Sigma_t A^T + H, \quad (18.38)$$

which is the *Lyapunov equation* of Chapter 15 (cf. Eq. 15.6). Thus after the CollectEvidence phase, the separator potentials can be represented by $(0, \Sigma_t)$, or by $(0, \Sigma_t^{-1})$ in canonical parameters. The clique potential $\psi(x_t, x_{t+1})$ is represented in moment parameters by:

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} \Sigma_t & -\Sigma_t A \\ -A^T \Sigma_t & A\Sigma_t A^T + H \end{bmatrix} \quad (18.39)$$

The representation in terms of canonical parameters can be obtained by inverting the covariance matrix (using the partitioned matrix inverse theorem, Eq. 13.16):

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} A^T H^{-1} A + \Sigma_t^{-1} & -A^T H^{-1} \\ -H^{-1} A & H^{-1} \end{bmatrix} \quad (18.40)$$

18.2.2 Introducing evidence

We return to the original junction tree in which the separator potentials are initialized to unity and now suppose that the outputs y are observed.

As in the case of the HMM we are not interested in the potential on (x_t, y_t) beyond its effect on the separator potential $\zeta^*(x_t)$. Moreover, $\zeta^*(x_t)$ is obtained by simply evaluating $P(y_t|x_t)$ at the observed value y_t :

$$\zeta^*(x_t) = \exp \left\{ -\frac{1}{2} (y_t - Cx_t)^T R^{-1} (y_t - Cx_t) \right\}. \quad (18.41)$$

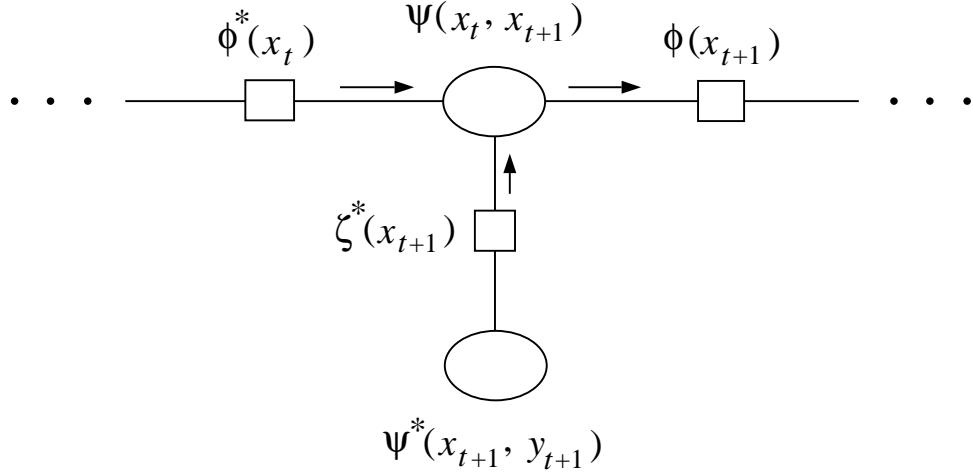


Figure 18.7: A fragment of the junction tree for the LG-HMM.

In canonical parameters this potential can be represented as: $(-C^T y_t, C^T R^{-1} C)$.

18.2.3 A forward algorithm

We designate the clique (x_{T-1}, x_T) as the root of the junction tree and collect to the root.

Let $(\hat{\xi}_{t|t}, S_{t|t})$ be the canonical parameters of the distribution of x_t conditioned on (y_0, \dots, y_t) . Similarly, let $(\hat{\xi}_{t+1|t}, S_{t+1|t})$ be the canonical parameters of the distribution of x_t conditioned on (y_0, \dots, y_{t+1}) . Our goal is to establish recursions for these quantities.

Consider the junction tree fragment shown in Figure 18.7. We suppose that the preceding separator potential Suppose that we have already obtained an updated $\phi^*(x_t)$ and wish to update $\psi(x_t, x_{t+1})$. We know from our general theory and (our experience with the HMM) that $\phi^*(x_t)$ must be proportional to $P(x_t | y_0, \dots, y_t)$; thus, we immediately identify the canonical parameters of $\phi^*(x_t)$ with $(\hat{\xi}_{t|t}, S_{t|t})$. Using Eq. 18.33 we now update $\psi(x_t, x_{t+1})$ by multiplying by $\phi^*(x_t)$. This yields the following canonical parameters for the updated potential:

$$\begin{bmatrix} \hat{\xi}_{t|t} \\ 0 \end{bmatrix}, \quad \begin{bmatrix} S_{t|t} + A^T H^{-1} A & -A^T H^{-1} \\ H^{-1} A & H^{-1} \end{bmatrix}. \quad (18.42)$$

If we now marginalize this potential with respect to x_t , we should expect to obtain a representation for $P(x_{t+1} | y_0, \dots, y_t)$. Indeed, applying Eq. 13.29 and Eq. 13.29, we obtain:

$$\hat{\xi}_{t+1|t} = H^{-1} A (S_{t|t} + A^T H^{-1} A)^{-1} \hat{\xi}_{t|t} \quad (18.43)$$

$$S_{t+1|t} = H^{-1} - H^{-1} A (S_{t|t} + A^T H^{-1} A)^{-1} A^T H^{-1}. \quad (18.44)$$

These two equations are identical to the time updates for the information filter in Chapter 15 (cf. Eq. 15.49 and Eq. 15.39).

The next step is to incorporate the evidence y_{t+1} by multiplying the updated clique potential on (x_t, x_{t+1}) by $\zeta^*(x_{t+1})$. This yields the following canonical parameterization for (x_t, x_{t+1}) :

$$\begin{bmatrix} \hat{\xi}_{t|t} \\ C^T R^{-1} y_{t+1} \end{bmatrix}, \quad \begin{bmatrix} S_{t|t} + A^T H^{-1} A & -A^T H^{-1} \\ H^{-1} A & H^{-1} + C^T R^{-1} C \end{bmatrix}. \quad (18.45)$$

Once again we marginalize this potential with respect to x_t . The result is the canonical parameterization of $\phi^*(x_{t+1})$:

$$\hat{\xi}_{t+1|t+1} = C^T R^{-1} y_{t+1} + H^{-1} A (S_{t|t-1} + A^T H^{-1} A)^{-1} \hat{\xi}_{t|t} \quad (18.46)$$

$$= \hat{\xi}_{t+1|t} + C^T R^{-1} y_{t+1} \quad (18.47)$$

$$S_{t+1|t+1} = H^{-1} + C^T R^{-1} C - H^{-1} A (S_{t|t} + A^T H^{-1} A)^{-1} A^T H^{-1} \quad (18.48)$$

$$= S_{t+1|t} + C^T R^{-1} C. \quad (18.49)$$

These results are identical to the measurement updates for the information filter in Chapter 15 (cf. Eq. 15.54 and Eq. 15.43).

18.2.4 A backward algorithm

In Section 15.7.2 we derived a backward algorithm for the LG-HMM by explicitly inverting the dynamics of the linear-Gaussian model and applying the information filter to the inverted dynamics. Recall that this approach yielded filtered estimates, i.e., conditional probabilities $P(x_t|y_{t+1}, \dots, y_T)$, rather than the usual “beta variables” $P(y_{t+1}, \dots, y_T|x_t)$.

In this section we see that this algorithm emerges in a straightforward way from the junction tree algorithm. In particular, suppose that we pre-initialize the junction tree by running a forward pass without evidence. From Section 18.2.1 we know that this pre-initialization pass leaves marginal probabilities on both the clique and the separator potentials. We would expect that a subsequent backward pass in the pre-initialized tree should yield filtered estimates $P(x_t|y_{t+1}, \dots, y_T)$.

Note in particular that whereas in Section 15.7.2 we had to invert the dynamics explicitly, this is not necessary in the current approach. The junction tree algorithm effectively inverts the dynamics for us. (In particular the derivation does not assume that A is an invertible matrix).

The derivation of the algorithm is similar to that of the previous section; the main difference being that we must remember to divide by the pre-initialized separator potentials. These potentials have the canonical representation $(0, \Sigma_t^{-1})$. Recall also that the initial values of the clique potentials are given by Eq. 18.40.

To obtain a two-step procedure in the backwards direction, it is useful to utilize an alternative junction tree in which the clique (x_t, y_t) is attached to the clique (x_t, x_{t+1}) rather than (x_{t-1}, x_t) . This junction tree is shown in Figure 18.8 where we see that it is the final clique and not the initial clique that has two state-to-output neighbors. Note that in this junction tree the evidence node (x_t, y_t) is to the left of the separator (x_{t+1}) , just as the evidence node (x_{t+1}, y_{t+1}) was to the right of the separator (x_t) in the earlier junction tree (Figure 18.2). As we will see, this allows us to obtain the usual two-step time and measurement updates from the junction tree.

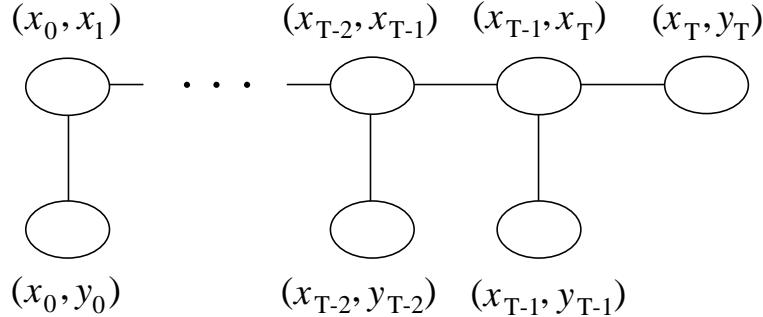


Figure 18.8: An alternative maximal spanning tree for the LG-HMM.

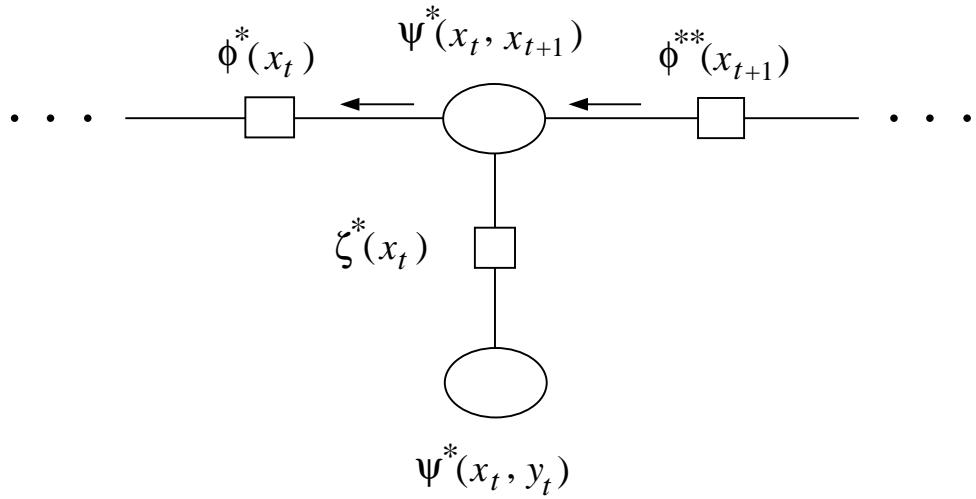


Figure 18.9: A fragment of the alternative junction tree for the LG-HMM.

We initially set the potentials of all cliques and separators to be proportional to the corresponding (unconditional) marginal probabilities. In particular, the potential $\psi(x_T, y_T)$ at the end of the chain is set proportional to $P(x_T, y_T) = P(y_T|x_T)P(x_T)$. We can conceptually view this assignment of potentials as resulting from a forward pass in the unconditioned graph (cf. Section 18.2.1).

Let us denote the canonical parameters of $\phi^*(x_{t+1})$ by $(\hat{\xi}_{t+1|t+1}, S_{t+1|t+1})$; note that this potential reflects the evidence from y_{t+1} onward. We now update $\psi(x_t, x_{t+1})$ by multiplying by $\phi^*(x_{t+1})$ and dividing by $\phi(x_{t+1})$, where the latter refers to the potential that was obtained unconditionally (see Figure 18.9). The parameters of the updated potential are:

$$\begin{bmatrix} 0 \\ \hat{\xi}_{t+1|t+1} \end{bmatrix}, \quad \begin{bmatrix} A^T H^{-1} A + \Sigma_t^{-1} & -A^T H^{-1} \\ H^{-1} A & S_{t+1|t+1} + H^{-1} - \Sigma_{t+1}^{-1} \end{bmatrix}, \quad (18.50)$$

where the $S_{t+1|t+1}$ in the lower-right-hand corner is due to multiplication by $\psi(x_t, x_{t+1})$ and the Σ_{t+1}^{-1} is due to division by $\phi(x_{t+1})$.

We now marginalize with respect to x_{t+1} to obtain the canonical representation of $P(x_t|y_{t+1}, \dots, y_T)$:

$$\hat{\xi}_{t|t+1} = A^T H^{-1} (S_{t+1|t+1} + H^{-1} - \Sigma_{t+1}^{-1})^{-1} \hat{\xi}_{t+1|t+1} \quad (18.51)$$

$$S_{t|t+1} = A^T H^{-1} (S_{t+1|t+1} + H^{-1} - \Sigma_{t+1}^{-1})^{-1} H^{-1} A. \quad (18.52)$$

These updates are identical to Eq. 15.98 and Eq. 15.96.

We now update the clique potential on (x_t, x_{t+1}) to reflect the evidence y_t . This amounts to adding the term $C^T R^{-1} C$ to the inverse covariance matrix and $C^T R^{-1} y_t$ to the linear term:

$$\begin{bmatrix} C^T R^{-1} y_t \\ \hat{\xi}_{t+1|t+1} \end{bmatrix}, \quad \begin{bmatrix} A^T H^{-1} A + \Sigma_t^{-1} + C^T R^{-1} C & -A^T H^{-1} \\ H^{-1} A & S_{t+1|t+1} + H^{-1} - \Sigma_{t+1}^{-1} \end{bmatrix}. \quad (18.53)$$

Marginalizing with respect to x_{t+1} yields the canonical representation of $P(x_t|y_t, \dots, y_T)$:

$$\hat{\xi}_{t|t} = \hat{\xi}_{t|t+1} + C^T R^{-1} y_t \quad (18.54)$$

$$S_{t|t+1} = S_{t|t+1} + C^T R^{-1} C, \quad (18.55)$$

which are identical to Eq. 15.98 and Eq. 15.97.

18.3 Summary

In this chapter we have shown how to derive many of the classical algorithms associated with the HMM and the LG-HMM from the point of view of the junction tree framework. Although we have not derived all possible algorithms, we have provided a representative sampling that shows several of the tricks of the trade.

A virtue of the junction tree formalism is that it displays all of relevant dependencies and their interrelationships; in particular, the pairwise clique potentials in the case of HMMs and LG-HMMs. This is a useful display for deriving algorithms, whether or not one uses the junction tree update formulas literally as we have done in this chapter, or uses the junction tree structure to derive alternative update formulas (e.g., based on moments).

18.4 Historical remarks and bibliography

Modelling Sequential Data using Graphical Models

Kevin P. Murphy
murphyk@ai.mit.edu

22 October 2002

1 Introduction

Chapter ?? introduced hidden Markov models (HMMs), and Chapter ?? introduced state space models (SSMs), both of which are popular, but somewhat inflexible, models of sequential data. In this chapter, we consider more complex models. The key generalization is to represent the hidden state in terms of a set of random variables, instead of a single random variable.¹ Similarly we can represent the observations in a factorized or distributed manner. We can then use graphical models to represent conditional independencies between these variables, both within and across positions within the sequence.

Sequential data comes in two main forms: temporal (time-series) data, which is generated sequentially by some causal process, and sequence data (e.g., bio-sequences or natural language), where we are more agnostic about the generating mechanism. For modelling time-series data, it is natural to use directed graphical models, which can capture the fact that time flows forward. Arcs within a time-slice can be directed or undirected, since they model “instantaneous” correlation. If all arcs are directed, both within and between slices, the model is called a dynamic Bayesian network (DBN). (The term “dynamic” means we are modelling a dynamic system, and does not mean the graph structure changes over time.) DBNs are quite popular because they are easy to interpret and learn: because the graph is directed, the conditional probability distribution (CPD) of each node can be estimated independently. In this chapter, we will focus on DBNs.

For modelling atemporal sequence data, it is possible to use directed or undirected graphical models. Much of our discussion of offline inference in temporal models is also applicable to atemporal models. The online inference methods we discuss can be applied to temporal models, but can also be used for sequential learning of static models, which is useful if the data is non-stationary or too large for batch methods.

We will not discuss learning (parameter estimation and model selection) in this chapter, since the techniques are just simple extensions of the methods for learning static models. Also, we defer discussion of sequential decision making (control/reinforcement learning problems) to Chapter ??.

2 Representation

In an HMM, the hidden state is represented in terms of a single discrete random variable, which can take on M possible values, $Q_t \in \{1, \dots, M\}$. In an SSM, the hidden state is represented in terms of a single vector-valued random variable, $X_t \in \mathbb{R}^M$. In a DBN, the hidden state is represented in terms of a set of N_h random variables, $Q_t^{(i)}$, $i \in \{1, \dots, N_h\}$, each of which can be discrete or continuous. Similarly, the observation can be represented in terms of N_o random variables, each of which can be discrete or continuous.

In an HMM/SSM, we have to define the transition model, $P(Q_t|Q_{t-1})$, the observation model, $P(Y_t|Q_t)$, and the initial state distribution, $P(Q_1)$. (These distributions may be conditioned on an input (control signal) U_t if present, e.g., the transition model would become $P(Q_t|Q_{t-1}, U_{t-1})$.) In a DBN, Q_t , Y_t and U_t represent sets of variables, so we define the corresponding conditional distributions using a two-slice temporal Bayes net (2TBN), which we shall

¹It is always possible to combine continuous-valued variables into a single vector-valued variable, but this obscures any conditional independencies that may exist between the components, defeating the purpose of using a graphical model. See Section 2.10.

denote by B_{\rightarrow} . The transition and observation models are then defined as a product of the CPDs in the 2TBN:

$$P(Z_t|Z_{t-1}) = \prod_{i=1}^N P(Z_t^{(i)}|\text{Pa}(Z_t^{(i)}))$$

where $Z_t^{(i)}$ is the i 'th node in slice t (which may be hidden or observed; hence $N = N_h + N_o$), and $\text{Pa}(Z_t^{(i)})$ are the parents of $Z_t^{(i)}$, which may be in the same or previous time-slice (assuming we restrict ourselves to first-order Markov models). We can represent the unconditional initial state distribution, $P(Z_1^{(1:N)})$, using a standard (one-slice) Bayes net, which we shall denote by B_1 . Together, B_1 and B_{\rightarrow} define the DBN. The joint distribution for a sequence of length T can be obtained by “unrolling” the network until we have T slices, and then multiplying together all of the CPDs:

$$P(Z_{1:T}^{(1:N)}) = \prod_{i=1}^N P_{B_1}(Z_1^{(i)}|\text{Pa}(Z_t^{(i)})) \times \prod_{t=2}^T \prod_{i=1}^N P_{B_{\rightarrow}}(Z_t^{(i)}|\text{Pa}(Z_t^{(i)}))$$

For example, Figure 1 shows a 2TBN for a standard HMM/SSM, and an unrolled version for a sequence of length $T = 4$. In this case, $Z_t^{(1)} = Q_t$, $Z_t^{(2)} = Y_t$, $\text{Pa}(Q_t) = Q_{t-1}$ and $\text{Pa}(Y_t) = Q_t$, so the joint becomes

$$P(Q_{1:T}, Y_{1:T}) = P(Q_1)P(Y_1|Q_1) \times \prod_{t=2}^T P(Q_t|Q_{t-1})P(Y_t|Q_t)$$

The parameters for slices $t = 2, 3, \dots$ are assumed to be the same (tied across time), as shown in Figure 2. This allows us to model unbounded amounts of data with a finite number of parameters.

In the following sections, we will present a series of more complex examples which should illustrate the large variety of models which we can define within this framework.

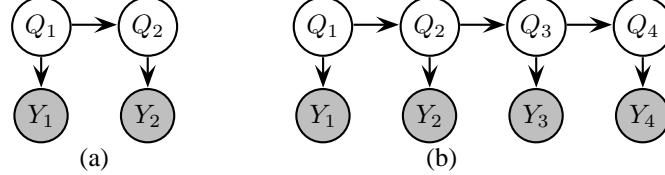


Figure 1: (a) A 2TBN for an HMM/SSM. (b) The model unrolled for $T = 4$ slices.

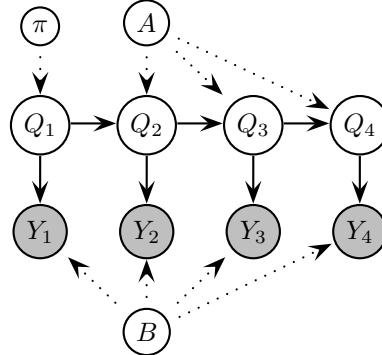


Figure 2: An HMM in which we explicitly represent the parameters (nodes with outgoing dotted arcs) and the fact that they are tied across time-slices. The parameters are $P(Q_1 = i) = \pi(i)$, $P(Q_t = j|Q_{t-1} = i) = A(i, j)$, and $P(Y_t = j|Q_t = i) = B(i, j)$. If the CPD for Y_t is a Gaussian, we would replace the B node with the mean and covariance parameters.

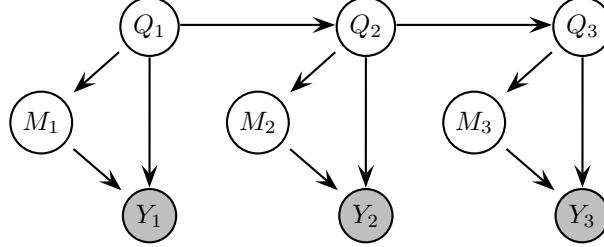


Figure 3: An HMM with mixture of Gaussians output.

2.1 HMMs with mixture-of-Gaussians output

In many applications, it is common to represent $P(Y_t|Q_t = i)$ using a mixture of Gaussians for each state i . We can explicitly model the mixture variable as shown in Figure 3. (M_t and Y_t are examples of transient nodes, since they do not have any children in the next time slice; by contrast, Q_t is a persistent node.) The CPDs for the Y_t and M_t nodes are as follows:

$$\begin{aligned} P(Y_t = y_t|Q_t = i, M_t = m) &= \mathcal{N}(y_t; \mu_{i,m}, \Sigma_{i,m}) \\ P(M_t = m|Q_t = i) &= C(i, m) \end{aligned}$$

The i 'th row of C encodes the mixture weights for state i .

In many applications, it is common that the observations are high-dimensional vectors (e.g., in speech recognition, Y_t is often a vector of cepstral coefficients and their derivatives, $Y_t \in \mathbb{R}^{39}$), so estimating a full covariance matrix for each value of Q_t and M_t requires a lot of data. An alternative is to assume there is a single global pool of Gaussians, and each state corresponds to a different mixture over this pool. This is called a semi-continuous or tied-mixture HMM. In this case, there is no arc from Q_t to Y_t , so the CPD for Y_t becomes

$$P(Y_t = y_t|M_t = m) = \mathcal{N}(y_t; \mu_m, \Sigma_m)$$

The effective observation model becomes

$$\begin{aligned} P(Y_t|Q_t = i) &= \frac{\sum_m P(Y_t, M_t = m, Q_t = i)}{P(Q_t = i)} \\ &= \frac{\sum_m P(Q_t = i)P(M_t = m|Q_t = i)P(Y_t|M_t = m)}{P(Q_t = i)} \\ &= \sum_m P(M_t = m|Q_t = i)\mathcal{N}(y_t; \mu_m, \Sigma_m) \end{aligned}$$

Hence all information about Y_t gets to Q_t via the “bottleneck” M_t . The advantages of doing this are not only reducing the number of parameters, but also reducing the number of Gaussian density calculations, which speeds up inference dramatically. (We discuss inference in Section 4.) For instance, the system can calculate $\mathcal{N}(y_t; \mu_m, \Sigma_m)$ for each m , and then reuse these in a variety of different models by simple reweighting: typically $P(M_t = m|Q_t = i)$ is non-zero for only a small number of m 's. The $M_t \rightarrow Y_t$ arc is like vector quantization, and the $Q_t \rightarrow M_t$ arc is like a dynamic reweighting of the codebook.

2.2 Auto-regressive HMMs

The standard HMM assumption that the observations are conditionally independent given the hidden state (i.e., $Y_t \perp Y_{t'}|Q_t$) is quite strong, and can be relaxed at little extra cost as shown in Figure 4. This model is sometimes called an auto-regressive HMM. This model reduces the effect of the Q_t “bottleneck”, by allowing Y_t to be predicted by Y_{t-1} as well as Q_t ; this results in models with higher likelihood (although not necessarily better classification performance).

If Y_t is discrete, its CPD can be represented as a 3-dimensional table, $P(Y_t|Q_t, Y_{t-1})$. If Y_t is continuous, one possibility for its CPD is a conditional linear Gaussian:

$$P(Y_t = y_t|Q_t = i, Y_{t-1} = y_{t-1}) = \mathcal{N}(y_t; W_i y_{t-1} + \mu_i, \Sigma_i)$$

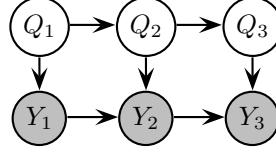


Figure 4: An auto-regressive HMM.

where W_i is the regression matrix given that Q_t is in state i . This model has a variety of different names: correlation HMM, conditionally Gaussian HMM, switching regression model, switching Markov model, etc.

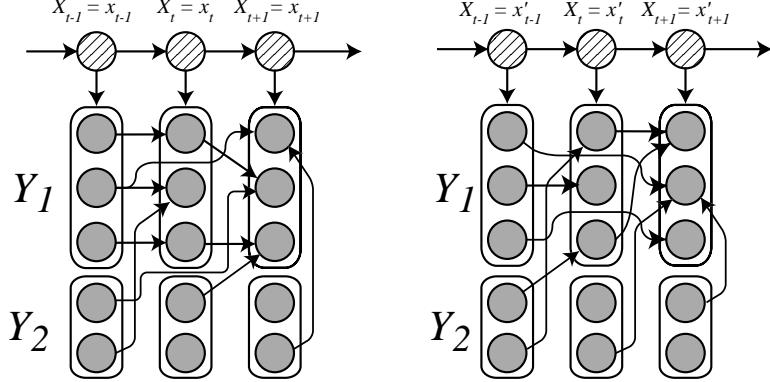


Figure 5: A buried Markov model. Depending on the value of the hidden variables, Q_t , the effective graph structure between the components of the observed variables (i.e., the non-zero elements of the regression matrix), can change. Two different instantiations are shown. Thanks to Jeff Bilmes for this figure.

“Buried” Markov models generalize auto-regressive HMMs by allowing non-linear dependencies between the observable nodes. Furthermore, the nature of the dependencies can change depending on the value of Q_t : see Figure 5 for an example. Such a model is called a dynamic Bayesian “multi net”, since it is a mixture of different networks.

2.3 Mixed-memory Markov models

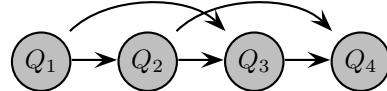


Figure 6: A trigram (second-order Markov) model, which defines $P(Q_t|Q_{t-1}, Q_{t-2})$.

One of the simplest approaches to modelling sequential data is to construct a Markov model of order k . These are often called n -gram models (where $n = k + 1$), e.g., standard first-order Markov models are bigram models ($n = 2$), second-order Markov models are trigram models ($n = 3$), etc. See Figure 6 for an example.

When Q_t is a discrete random variable with many possible values (e.g., if Q_t represents words), then there might not be enough data to reliably estimate $P(Q_t = k|Q_{t-1} = j, Q_{t-2} = i)$. A common approach is to create a mixture of lower-order Markov models:

$$P(Q_t|Q_{t-1}, Q_{t-2}) = \alpha_3(Q_{t-1}, Q_{t-2})f(Q_t|Q_{t-1}, Q_{t-2}) + \alpha_2(Q_{t-1}, Q_{t-2})f(Q_t|Q_{t-1}) + \alpha_1(Q_{t-1}, Q_{t-2})f(Q_t)$$

where the α coefficients may optionally depend on the history, and $f(\cdot)$ is an arbitrary (conditional) probability distribution. This is the basis of the method called deleted interpolation.

We can explicitly model the latent mixture variable implicit in the above equation as shown in Figure 7. Here $S_t = 1$ means use a unigram, $S_t = 2$ means use bi- and unigrams, and $S_t = 3$ means use tri-, bi- and uni-grams. S_t is

called a switching parent, since it determines which of the other parents links are active, i.e., the effective topology of the graph changes depending on the value of the switch. Since S_t is hidden, the net effect is to use a mixture of all of these.

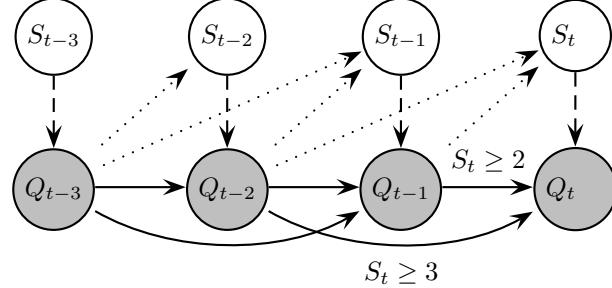


Figure 7: A mixed-memory Markov model. The dashed arc from S_t to Q_t means S_t is a switching parent: it is used to decide which of the other parents to use, either Q_{t-1} or Q_{t-2} or both. The conditions under which each arc is active are shown for the Q_t node only, to reduce clutter. In deleted interpolation, the $Q_{t-1} \rightarrow Q_t$ arc is active for states $S_t \in \{2, 3\}$, and the $Q_{t-2} \rightarrow Q_t$ arc is active for states $S_t \in \{3\}$. (If $S_t = 1$, then $Q_t \perp (Q_{t-1}, Q_{t-2})$.) In mixed memory models, the $Q_{t-1} \rightarrow Q_t$ arc is active iff $S_t = 1$, and the $Q_{t-2} \rightarrow Q_t$ arc is active iff $S_t = 2$. The dotted arcs from Q_{t-1} and Q_{t-2} to S_t are optional, and reflect the fact that the coefficients can depend on the identity of the words in the window. See text for details. Based on Figure 15 of [Bil01].

A very similar approach, called mixed-memory Markov models, is to always use mixtures of bigrams at different lags, e.g., if $S_t = 1$, we use $P(Q_t|Q_{t-1})$, and if $S_t = 2$, we use $P(Q_t|Q_{t-2})$. This can be achieved by simply changing the “guard condition” on the Q_{t-1} to Q_t link to be $S_t = 1$ instead of $S_t \geq 2$; similarly, the guard on the Q_{t-2} to Q_t link becomes $S_t = 2$ instead of $S_t \geq 3$. Hence S_t acts like the input to a multiplexer, choosing one of the parents to feed into Q_t . Using the terminology of [BZ02], S_t is the switching parent, and the other parents are the conditional parents. The overall effect is to define the CPD as follows:

$$P(Q_t|Q_{t-1}, \dots, Q_{t-n}) = \sum_{i=1}^n P(Q_t|Q_{t-i})P(S_t = i)$$

2.4 Factorial HMMs

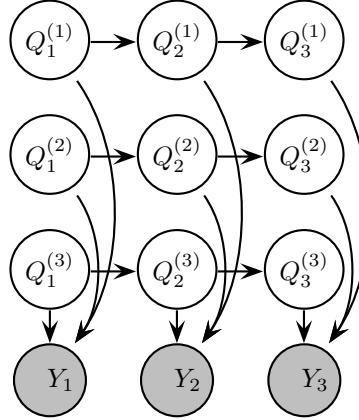


Figure 8: A factorial HMM with 3 hidden chains.

Figure 8 shows a factorial HMM, which has a single output (observation) variable but has a distributed representation for the hidden state. Note that, although all the hidden chains are a priori independent, once we condition on the evidence, they all become correlated. Intuitively, this is because all the chains “compete” to explain each observation (the explaining away phenomenon); more formally, this can be seen by noticing that all the chains become connected

in the moral graph, because they share a common observed child. This fact means exact inference in this model takes $O(M^{N_h})$ operations per time step. Unfortunately this is true for most DBNs, even ones which do not have densely connected nodes such as Y_t , as we discuss in Section 4. Hence in general we will need to use approximations, which we discuss later.

The CPD for each hidden node, $P(Q_t^{(i)}|Q_{t-1}^{(i)})$, can be represented by an $M \times M$ stochastic matrix (assuming each chain can be in one of M states). A naive representation for the CPD for the observed nodes, $P(Y_t|Q_t^{(1:N_h)})$, would require $O(M^{N_h})$ parameters, for each possible combination of the parent values. A more parsimonious representation can be obtained by treating each discrete parent as a continuous random variable in $\{0, 1\}^M$, i.e., by using a 1-in- M (distributed) encoding. Let us denote this transformation of $Q_t^{(i)}$ by $\widehat{Q}_t^{(i)}$, and denote the concatenation of $\widehat{Q}_t^{(1)}, \dots, \widehat{Q}_t^{(N_h)}$ by \widehat{Q}_t . Then we can define the observation distribution using a generalized linear model. For example, if Y_t is continuous, we can use a Gaussian:

$$P(Y_t|Q_t^{(1:N_h)}) = \mathcal{N}(Y_t; W\widehat{Q}_t, \Sigma)$$

For example, if $Q_t^{(1)} \in \{1, 2\}$ has value 1, $Q_t^{(2)} \in \{1, 2, 3\}$ has value 3, and $Q_t^{(3)} \in \{1, 2\}$ has value 2, then $\widehat{Q}_t^{(1)} = (1 \ 0)$, $\widehat{Q}_t^{(2)} = (0 \ 0 \ 1)$, and $\widehat{Q}_t^{(3)} = (0 \ 1)$, so the mean is given by

$$W\widehat{Q}_t = \begin{pmatrix} W_{11}^1 & W_{12}^1 & W_{11}^2 & W_{12}^2 & W_{13}^2 & W_{11}^3 & W_{12}^3 \\ W_{21}^1 & W_{22}^1 & W_{21}^2 & W_{22}^2 & W_{23}^2 & W_{21}^3 & W_{22}^3 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} W_{11}^1 \\ W_{21}^1 \end{pmatrix} + \begin{pmatrix} W_{12}^2 \\ W_{22}^1 \end{pmatrix} + \begin{pmatrix} W_{13}^2 \\ W_{23}^2 \end{pmatrix} + \begin{pmatrix} W_{11}^3 \\ W_{21}^3 \end{pmatrix}$$

In other words, each parent selects a column from its own weight matrix W^i according to its value, and we sum the results. (In this example, we select columns 1, 3 and 2 from W^1 , W^2 and W^3 , respectively.)

A factorial HMM, like any discrete-state DBN, can be converted to a regular HMM by creating a single “mega” variable, Q_t , whose state space is the Cartesian product of each of the discrete hidden variables. However, this is a bad idea: the resulting “flat” representation is hard to interpret, inference in the flat model will be exponentially slower (see Section 4.2), and learning will be harder because there will be exponentially many more parameters.

This last statement needs some explanation. Although it is true that the entries of the transition matrix of the flat HMM have constraints between them (so the number of free parameters remains the same in both the flat and the factored representation), it is hard to exploit these constraints either in inference or learning. For example, if $N_h = 2$, the flat transition matrix can be computed as follows:

$$P(Q_t^{(1)} = j_1, Q_t^{(2)} = j_2 | Q_{t-1}^{(1)} = i_1, Q_{t-1}^{(2)} = i_2) = P(Q_t^{(1)} = j_1 | Q_{t-1}^{(1)} = i_1) \times P(Q_t^{(2)} = j_2 | Q_{t-1}^{(2)} = i_2)$$

Hence the free parameters, which are the pairwise transition matrices on the right hand side of this equation, get combined in a way which is hard to disentangle.²

2.5 Coupled HMMs

Figure 9 shows another example of a DBN, called a coupled HMM, in which the hidden variables are assumed to interact locally with their neighbors. Also, each hidden node has its own “private” observation.

This model has many applications. For example, in audio-visual speech recognition, one chain might represent the speech signal, and the other chain the visual signal (e.g., obtained from a lip tracking system). Another example concerns monitoring the state of a freeway using magnetic loop detectors distributed down the middle lane. Let $Y_t^{(i)}$ represent the measurement of sensor i at time t , and $Q_t^{(i)}$ represent the hidden state of location i at time t . It is natural

²If all CPDs are linear-Gaussian, so the resulting joint distribution is a multivariate Gaussian, sparse graphical structure corresponds to sparse matrices in the traditional sense of having many zeros. In this special case, the free parameters of the DBN simply get concatenated into a large matrix, rather than being multiplied together, as we will see in Section 2.10. However, even for Gaussian systems, the graphical representation can be helpful.

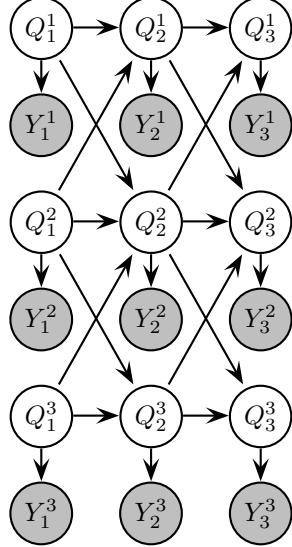


Figure 9: A coupled HMM with 3 chains.

to assume the hidden state at location i depends on the previous hidden state at i , plus the previous hidden states at locations immediately upstream and downstream. This can easily be modelled using a coupled HMM.

The CPDs for the hidden nodes in the “middle” have the form $P(Q_t^{(i)}|Q_{t-1}^{(i)}, Q_{t-1}^{(i+1)}, Q_{t-1}^{(i-1)})$. If each chain can have M states, this transition matrix has $O(M^4)$ parameters. One way to reduce this is to represent this higher order transition matrix as a mixture of pair-wise transition matrices, as in a mixed-memory Markov model (see Section 2.3):

$$P(Q_t^{(i)}|Q_{t-1}^{(i)}, Q_{t-1}^{(i+1)}, Q_{t-1}^{(i-1)}) = P(Q_t^{(i)}|Q_{t-1}^{(i)})P(S_t = 1) + P(Q_t^{(i)}|Q_{t-1}^{(i+1)})P(S_t = 2) + P(Q_t^{(i)}|Q_{t-1}^{(i-1)})P(S_t = 3)$$

where S_t is a latent mixture variable (not shown in the DBN), whose prior probability encodes the mixing weights of each pairwise transition matrix. We can fit such a CPD using EM. (That is, the M step for this CPD will involve a local EM algorithm, inside of the EM algorithm for the whole model.)

2.6 Variable-duration (semi-Markov) HMMs

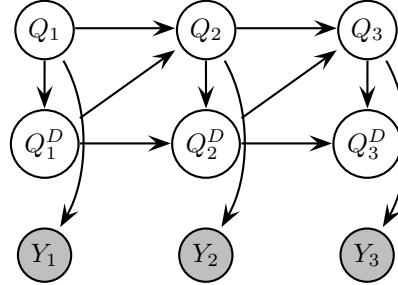


Figure 10: A variable-duration HMM. Q_t represents the state, and Q_t^D represents how long we have been in that state (duration).

The self-arc on a state in an HMM defines a geometric distribution over waiting times. Specifically, the probability we remain in state i for exactly d steps is $p_i(d) = (1 - p)p^{d-1}$, where $p = A(i, i)$ is the self-loop probability. To allow for more general durations, we can use a semi-Markov model. (It is called semi-Markov because to predict the next state, it is not sufficient to condition on the past state: we also need to know how long we’ve been in that state.) $p_i(d)$ can be represented as a table (a non-parametric approach) or as some kind of parametric function. If $p_i(d)$ is a geometric distribution, the model becomes equivalent to a standard HMM.

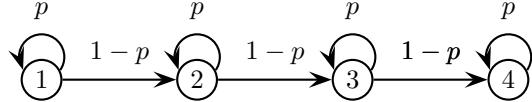
A first attempt at modelling this as a DBN is shown in Figure 10. We explicitly add Q_t^D , the remaining duration of state Q_t , to the state-space. (Even though Q_t is constant for a long period, we copy its value across every time slice, to ensure a regular structure.) When we first enter state i , Q_t^D is set to a value from $q_i(\cdot)$; it then deterministically counts down to 0. When $Q_t^D = 0$, the state is free to change, and Q_t^D is set to the duration of the new state. We can encode this behavior by defining the CPDs as follows:

$$P(Q_t = j | Q_{t-1} = i, Q_{t-1}^D = d) = \begin{cases} \delta(i, j) & \text{if } d > 0 \text{ (remain in same state)} \\ A(i, j) & \text{if } d = 0 \text{ (transition)} \end{cases}$$

$$P(Q_t^D = d' | Q_{t-1}^D = d, Q_t = k) = \begin{cases} p_k(d') & \text{if } d = 0 \text{ (reset)} \\ \delta(d', d - 1) & \text{if } d > 0 \text{ (decrement)} \end{cases}$$

Since we have expanded the state space, inference in a variable-duration HMM is slower than in a regular HMM. The naive approach to inference in this DBN takes $O(TD^2M^2)$ time, where D is the maximum number of steps we can spend in any state, and M is the number of HMM states. However, we can exploit the fact that the CPD for Q^D is deterministic to reduce this to $O(TDM^2)$.

A more efficient, but less flexible, way to model non-geometric waiting times is to replace each HMM state with n new states, for some n to be determined, each with the same emission probabilities as the original state. For example, consider this model.



Obviously the smallest sequence this can generate is of length $n = 4$. Any path of length l through the model has probability $p^{l-n}(1-p)^n$; the number of possible paths is $\binom{l-1}{n-1}$, so the total probability of a path of length l is

$$p(l) = \binom{l-1}{n-1} p^{l-n}(1-p)^n$$

This is the negative binomial distribution. By adjusting n and p , we can model a wide range of waiting times.

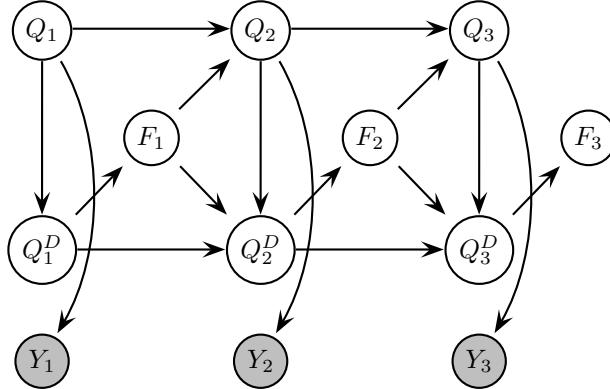


Figure 11: A variable-duration HMM with explicit finish nodes. Q_t represents the state, and Q_t^D represents how long we have been in that state (duration), and F_t is a binary indicator variable that turns to indicate that Q_t^D has finished.

The model with explicit duration variables will turn out to be easier to generalize. To facilitate such generalizations, we introduce deterministic “finish” nodes, that “turn on” when the duration counter reaches 0. This is a signal that Q_t can change state, and that a new duration should be chosen. See Figure 11. We define the CPDs as follows:

$$\begin{aligned}
P(Q_t = j | Q_{t-1} = i, F_{t-1} = f) &= \begin{cases} \delta(i, j) & \text{if } f = 0 \text{ (remain in same state)} \\ A(i, j) & \text{if } f = 1 \text{ (transition)} \end{cases} \\
P(Q_t^D = d' | Q_{t-1}^D = d, Q_t = k, F_{t-1} = 1) &= p_k(d') \\
P(Q_t^D = d' | Q_{t-1}^D = d, Q_t = k, F_{t-1} = 0) &= \begin{cases} \delta(d', d - 1) & \text{if } d > 0 \\ \text{undefined} & \text{if } d = 0 \end{cases} \\
P(F_t = 1 | Q_t^D = d) &= \delta(d, 0)
\end{aligned}$$

Note that $P(Q_t^D = d' | Q_{t-1}^D = d, Q_t = k, F_{t-1} = 0)$ is undefined if $d = 0$, since if $Q_{t-1}^D = 0$, then $F_{t-1} = 1$, by construction.

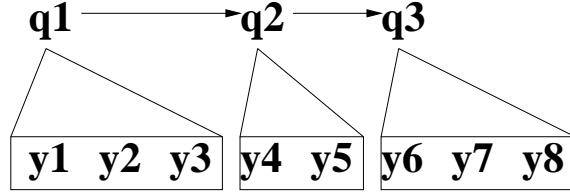


Figure 12: A time-series is divided into 3 segments of different lengths.

The overall effect of using a variable duration HMM is illustrated in Figure 12: the sequence is divided into a series of segments of different lengths; the observations within each segment are assumed to be iid, conditioned on the state. (We will generalize this in Section 2.7.) The assignment of values to the nodes in Figure 11 corresponding to this example is as follows:

Q_t	q_1	q_1	q_1	q_2	q_2	q_3	q_3	q_3
D_t	2	1	0	1	0	2	1	0
F_t	0	0	1	0	1	0	0	1
Y_t	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8

We see that the places where F turns on represent segment boundaries.

2.7 Segment models

The basic idea of a segment model is that each HMM state can generate a sequence of observations, as in Figure 12, instead of just a single observation. The difference from a variable-duration HMM is that we do not assume the observations within each segment are conditionally independent; instead we can use an arbitrary model for their joint distribution. The likelihood is given by

$$P(y_{1:T}) = \sum_{\tau} \sum_{q_{1:\tau}} \sum_{l_{1:\tau}} \prod_{i=1}^{l_i} P(\tau) P(q_i | q_{i-1}, \tau) P(l_i | q_i) P(y_{t_0(i):t_1(i)} | q_i, l_i)$$

where τ is the number of segments, l_i is the length of the i 'th segment (that satisfies the constraint $\sum_{i=1}^{\tau} l_i = T$), and $t_0(i) = \sum_{j=1}^{i-1} l_j + 1$ and $t_1(i) = t_0(i) + l_i - 1$ are the start and ending times of segment i .

A first attempt to represent this as a graphical model is shown in Figure 13. This is not a fully specified graphical model, since the L_i 's are random variables, and hence the topology is variable. To specify a segment model as a DBN, we must make some assumptions about the form of $P(y_{t:t+l} | q, l)$. Let us consider a particular segment and renumber so that $t_0 = 1$ and $t_1 = l$. If we assume the observations are conditionally independent,

$$P(y_{1:l} | Q_t = k, l) = \prod_{t=1}^l P(y_t | Q_t = k)$$

we recover the variable-duration HMM in Figure 11. (If $p(l|q)$ is a geometric distribution, this becomes a regular HMM.) Note that the number of segments is equal to the number of transitions of the Q_t nodes, or equivalently, the

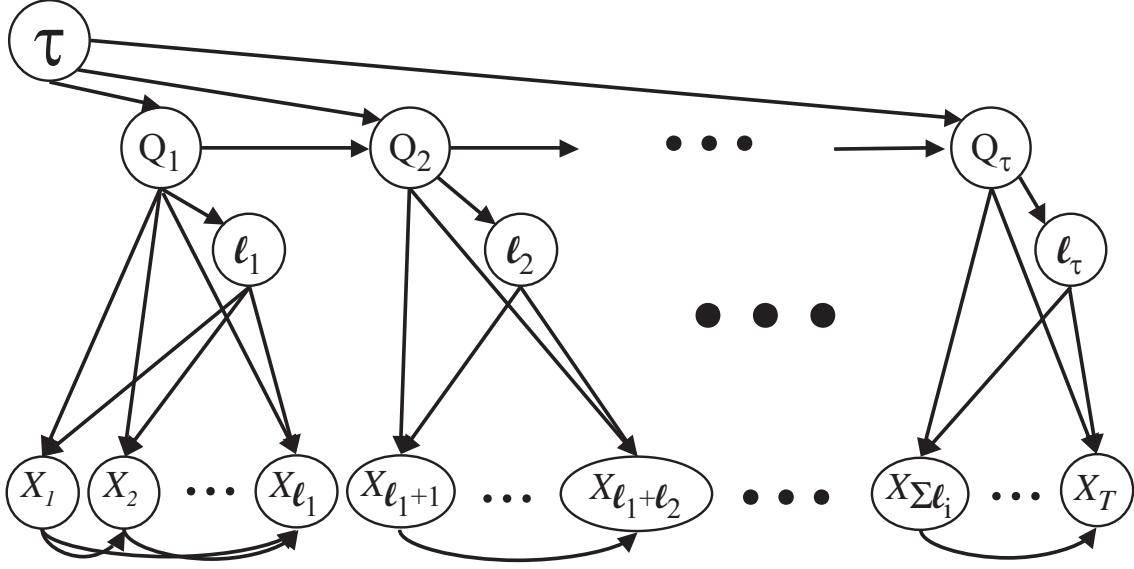


Figure 13: A schematic depiction of a segment model. The X_t nodes are observed, the rest are hidden. The X_t 's within a segment need not be fully connected. Also, there may be dependencies between the observables in adjacent segments (not shown). This is not a valid DBN since the l_i 's are random variables, and hence the structure is not fixed. Thanks to Jeff Bilmes for this Figure.

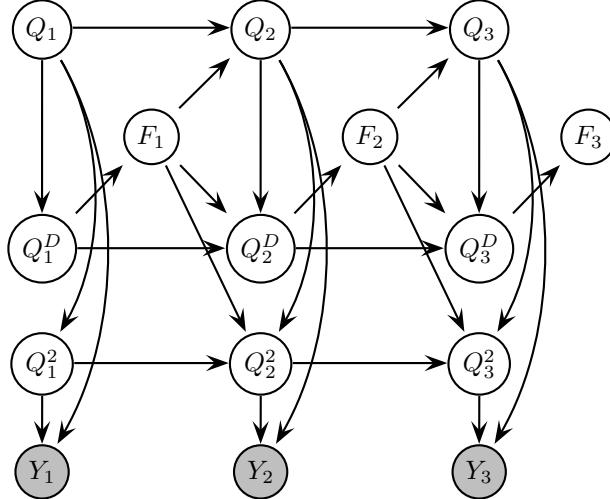


Figure 14: A segment model where each segment is modelled by an HMM.

number of times F_t turns on. By considering all possible assignments to Q_t and Q_t^D (and hence to F_t), we consider all possible segmentations of the data.

The next simplest segment model is to model each segment $P(y_{1:l}|Q_t = k, l)$ using an HMM, i.e.,

$$P(y_{1:l}|Q_t = k, l) = \sum_{q_{1:l}} \pi_k(q_1) P(y_1|Q_1 = k, Q_1^2 = q_1) \prod_{\tau=2}^l A_k(q_{\tau-1}, q_\tau) P(y_\tau|Q_t = k, Q_\tau^2 = q_\tau)$$

where Q_t^2 represents the state of the HMM within this particular segment. We can model this as a DBN as shown in Figure 14. (Naturally we could allow arcs between the Y_t arcs, as in Section 2.2). The CPDs for Q_t , Q_t^D and F_t are the same as in Figure 11. The CPD for Y_t gets modified because it must condition on both Q_t and the state within the segment-level HMM, Q_t^2 . The CPD for Q_t^2 is as follows:

$$P(Q_t^2 = j | Q_{t-1}^2 = i, Q_t = k, F_{t-1} = f) = \begin{cases} \pi_k^2(j) & \text{if } f = 0 \text{ (reset)} \\ A_k^2(i, j) & \text{if } f = 1 \text{ (transition)} \end{cases}$$

It is straightforward to use other models to define the segment likelihood $P(y_{1:l} | Q_t = k, l)$, e.g., a second-order Markov model, or a state-space model (Section 2.10).

In Section 2.8, we discuss a special case of segment HMMs, where the length of each segment is known in advance. In Section 2.9, we discuss a generalization of segment HMMs, where we allow multiple levels of hierarchy.

2.8 Embedded HMMs

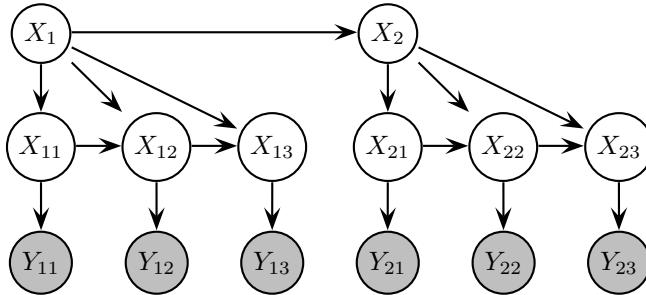


Figure 15: An embedded HMM. Here we have 2 sequences of length 3, both of which are modelled by sub-HMMs. The length of each segment is fixed in advance.

If we know the length of each segment in advance, we can eliminate the Q_t^D and F_t nodes, and simply “unroll” the HMM defining each segment a pre-specified number of times. See Figure 15 for an example. This model has been called a “hierarchical mixture of Markov chains”, an “embedded HMM”, and a “pseudo-2D HMM”. The reason for the term pseudo-2D is because this model provides a way of extending HMMs to 2D data such as images. The basic idea is that each row of an image is considered as a segment of known size. These lower-level/row HMMs can then be used to define the observation likelihood for each state of the higher-level/column HMM. The overall effect is to allow dynamic warping in both the horizontal and vertical directions (although the warping in each row is independent). For example, this has been applied to do face recognition: the states for the top-level HMM represent forehead, eyes, nose, mouth or chin; each one of these states then generates a lower level HMM for modelling each row.

Note that inference in such models is very easy, since, conditioned on the states of the top-level HMM, the model reduces to a set of independent HMMs, one per row. Hence one only has to slightly modify Pearl’s algorithm for trees. We leave the details as an exercise.

2.9 Hierarchical HMMs (HHMMs)

The Hierarchical HMM (HHMM) is a generalization of the segment HMM. It allows segment HMMs to be composed of segment sub-HMMs in a hierarchical fashion. See Figure 16 for an illustration of the basic idea. In addition, instead of specifying the length of each segment with a random variable, the length is defined implicitly by the amount of time it takes the sub-HMM to enter one of its end states. Entering an end-state terminates that segment, and returns control to the calling state, which is then free to change.

The calling context is memorized on a depth-limited stack. Hence an HHMM is less powerful than stochastic context-free grammars (SCFGs) and recursive transition networks (RTNs), both of which can handle recursion to an unbounded depth. However, HHMMs are sufficiently expressive for many practical problems, which often only involve tail-recursion (i.e., self transitions to an abstract state). Furthermore, HHMMs can be made much more computationally efficient than SCFGs. In particular, the inference algorithm for SCFGs, which is called the inside-outside algorithm, takes $O(T^3)$, whereas we can use any of the techniques we discuss in Section 3 to do inference in an HHMM in $O(T)$ time.

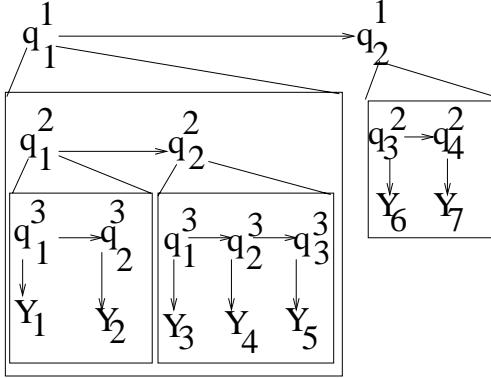


Figure 16: A time-series is hierarchically divided into segments. The transition/observation distributions of the sub-HMMs may be conditioned on their immediate context, or their entire surrounding context. In a context free system, information is not allowed to flow across segment boundaries, but this restriction can be lifted.

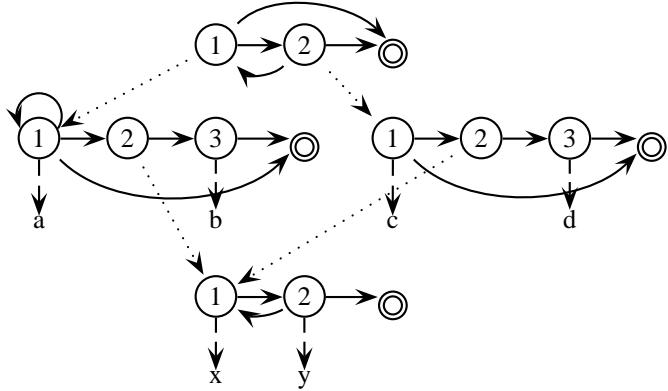


Figure 17: State transition diagram for a three-level HHMM representing the regular expression $(a^+ | a^+(xy)^+ b | c | c(xy)^+ d)^+$. Solid arcs represent horizontal transitions between states; dotted arcs represent vertical transitions, i.e., calling a sub-HMM. Double-ringed states are accepting (end) states. Dashed arcs are emissions from production (concrete) states. In this example, each production state emits a unique symbol, but in general, it can have a distribution over output symbols.

Figure 17 shows the state transition diagram of an example HHMM which models the regular expression

$$(a^+ | a^+(xy)^+ b | c | c(xy)^+ d)^+.^3$$

The key difference from using a flat HMM to model this is that the model for the sub-expression $(xy)^+$ is re-used in both the $a^+(xy)^+ b$ and $c(xy)^+ d$ subexpressions. In this small example, the benefits of re-use are minimal, but in general a hierarchical model with reuse needs substantially fewer parameters.

The state of the HHMM can be defined by specifying the value of the state variables at each level of the hierarchy, call them $Q_t^{(1)}, \dots, Q_t^{(D)}$, where D is the depth. States which emit single observations are called “production states”, and those that emit strings (by calling sub-HMMs) are termed “abstract states”. For example, in Figure 17, state $(1, 2, -)$ is an abstract state and $(1, 1, -)$ and $(1, 2, 1)$ are concrete states which emit a and x respectively. States $(1, 1, 1)$ and $(2, 1, 1)$ are the same (we could denote them by $(-, 1, 1)$), since the bottom-level HMM is reused in both top-level contexts.

The DBN that corresponds to the above example is shown in Figure 18. We define the CPDs below, but first we

³This notation means the model must produce one or more a 's, or one or more a 's followed by one or more x 's and y 's followed by a single b , or a c , or a c followed by one or more x 's and y 's followed by a d . Having created one of these strings, it is free to create another one. An HHMM cannot make a horizontal transition before it makes a vertical one; hence it cannot produce the empty string. For example, in Figure 17, it is not possible to transition directly from state 1 to 2.

give an example. Consider the string $aaxybc$; there are two possible parses, $(aaxyb)(c)$ or $(a)(axyb)(c)$, corresponding to whether the two a 's belong together (i.e., both are generated from $a^+(xy)^+b$), or are separate (i.e., one is from a^+ and one is from $a^+(xy)^+b$). In addition, the c could be the beginning of the subexpression $c(xy)^+d$, or a single c ; we assume we know this is a complete string (rather than a substring), hence the c is terminal (so $F_T^2 = 1$). The assignments to the nodes in the DBN for the joint aa hypothesis are as follows (– represents a don't care value):

Q_t^1	1	1	1	1	1	2
Q_t^2	1	1	2	2	3	1
F_t^2	0	0	0	0	1	1
Q_t^3	–	–	1	2	–	–
F_t^3	–	–	0	1	1	–
Y_t	a	a	x	y	b	c

The assignments to the nodes in the DBN for the separate aa hypothesis are the same as above, except $F_1^2 = 1$, representing the presence of a segmentation boundary. (This corresponds to the following path through the state transition diagram: from state $(1, 1, –)$ to its end state, returning to the root state $(1, –, –)$, and then re-entering $(1, 1, –)$.)

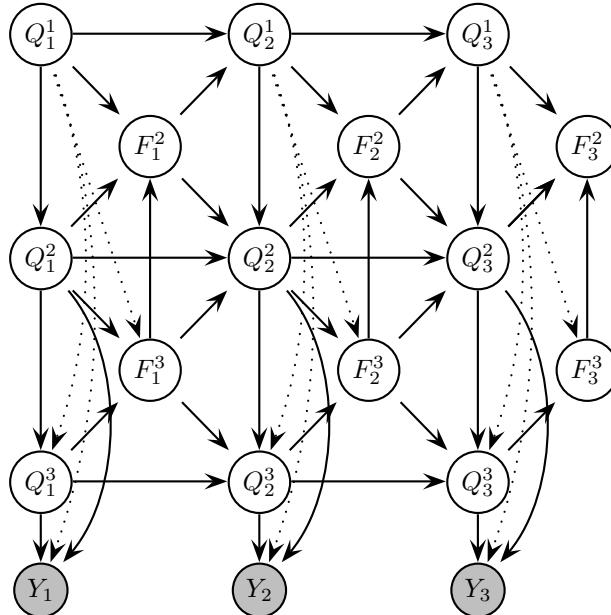


Figure 18: A 3-level HHMM represented as a DBN. Q_t^d is the state at time t , level d ; $F_t^d = 1$ if the HMM at level d has finished (entered its exit state), otherwise $F_t^d = 0$. Shaded nodes are observed; the remaining nodes are hidden. If the length of the string is known to be T , we can set $F_T^d = 1$ for all d , to ensure all sub-models have terminated. If the dotted arcs from level 1 to the bottom are omitted, it represents the fact that the bottom-level HMM is shared amongst different top-level HMMs, as in the example in Figure 17.

We now define the CPDs, which will complete the definition of the model. We consider the bottom, middle and top layers of the hierarchy separately (since they have different local topology), as well as the first, middle and last time slices.

Bottom level ($d = D, t = 2 : T - 1$): Q^D follows a Markov chain with parameters determined by which sub-HMM it is in, which is encoded by the vector of higher-up state variables $Q_t^{1:D-1}$, which we will represent by the integer k for brevity. (If the state transition diagram is sparse, many of these joint configurations will be impossible; k will only index the valid configurations.)

Instead of Q^D entering its end state, it “turns on” F^D to mean it is finished (see Section 2.9.1 for more details). This will be a signal that higher-level HMMs can now change state. In addition, it will be a signal that the

next value of Q^D should be drawn from its prior distribution (representing a vertical transition), instead of its transition matrix (representing a horizontal transition). Formally, we can write this as follows:

$$P(Q_t^D = j | Q_{t-1}^D = i, F_{t-1}^D = f, Q_t^{1:D-1} = k) = \begin{cases} \tilde{A}_k^D(i, j) & \text{if } f = 0 \\ \pi_k^D(j) & \text{if } f = 1 \end{cases}$$

A_k^D is the transition matrix for level D given that the parent variables are in state k , and \tilde{A}_k^D is just a rescaled version of this (see Section 2.9.1 for details on the rescaling). Similarly, π_k^D is the initial distribution for level D given that the parent variables are in state k . The equation for F_D is simply

$$P(F_t^D = 1 | Q_t^{1:D-1} = k, Q_t^D = i) = A_k^D(i, \text{end}).$$

where end represents the end-state for this HMM.

Intermediate levels ($d = 2 : D - 1, t = 2 : T - 1$): As before, Q^d follows a Markov chain with parameters determined by $Q^{1:d-1}$, and F^d specifies whether we should use the transition matrix or the prior. The difference is that we now also get a signal from below, F^{d+1} , specifying whether the sub-model has finished or not; if it has, we are free to change state, otherwise we must remain in the same state. Formally, we can write this as follows:

$$P(Q_t^d = j | Q_{t-1}^d = i, F_{t-1}^{d+1} = b, F_{t-1}^d = f, Q_t^{1:d-1} = k) = \begin{cases} \delta(i, j) & \text{if } b = 0 \\ \tilde{A}_k^d(i, j) & \text{if } b = 1 \text{ and } f = 0 \\ \pi_k^d(j) & \text{if } b = 1 \text{ and } f = 1 \end{cases}$$

F^d should “turn on” only if Q^d is “allowed” to enter a final state, the probability of which depends on the current context $Q^{1:d-1}$. Formally, we can write this as follows:

$$P(F_t^d = 1 | Q_t^d = i, Q_t^{1:d-1} = k, F_t^{d+1} = b) = \begin{cases} 0 & \text{if } b = 0 \\ A_k^d(i, \text{end}) & \text{if } b = 1 \end{cases}$$

Top level ($d = 1, t = 2 : T - 1$): The top level differs from the intermediate levels in that the Q node has no Q parent to specify which distribution to use. The equations are the same as above, except we eliminate the conditioning on $Q_t^{1:d-1} = k$.

Initial slice ($t = 1, d = 1 : D$): The CPDs for the nodes in the first slice are as follows: $P(Q_1^1 = j) = \pi^1(j)$ for the top level and $P(Q_1^d = j | Q_1^{1:d-1} = k) = \pi_k^d(j)$, for $d = 2, \dots, D$.

Final slice ($t = T, d = 1 : D$): To ensure that all sub-HMMs have reached their end states by the time we reach the end of sequence, we can clamp $F_T^d = 1$ for all d . (In fact, it is sufficient to clamp $F_T^1 = 1$, since this implies $F_T^d = 1$ for all d .) This forces all segmentations to be consistent with the known length of the sequence.

Observations: If the observations are discrete symbols, we may represent $P(Y_t | \vec{Q}_t)$ as a table. If the observations are real-valued vectors, we can use a Gaussian for each value of \vec{Q}_t . Of course, more parsimonious representations are possible. Alternatively, we can condition Y_t only on some of its parents.

2.9.1 Handling end states

Unlike the automaton representation, the DBN never actually enters an end state (i.e., Q_t^d can never take on the value “end”); if it did, what should the corresponding observation be? (Unlike an HMM, a DBN must generate an observation at every time step.) Instead, Q_t^d causes F_t^d to turn on, and then enters a new (non-terminal) state at time $t + 1$, chosen from its initial state distribution. This means that the DBN and HHMM transition matrices are not identical. However, they satisfy the following constraint:

$$\tilde{A}_k^d(i, j) (1 - A_k^d(i, \text{end})) = A_k^d(i, j)$$

where A represents the HHMM transition matrix, \tilde{A} represents the DBN transition matrix, and $A_k^d(i, \text{end})$ is the probability of terminating from state i . The reason is that the probability of each horizontal transition in the DBN gets multiplied by the probability that $F_t^d = 0$, which is $1 - A_k^d(i, \text{end})$; this product should match the automaton probability. It is easy to see that the new matrix is stochastic (i.e., $\sum_{j=1}^M \tilde{A}_k^d(i, j) = 1$), since the original matrix satisfies $\sum_{j=1}^M A_k^d(i, j) + A_k^d(i, \text{end}) = 1$, where M is the number of states.

2.9.2 Applications of HHMMs

The most obvious application of HHMMs is to speech recognition. It is natural to think of words as composed of phones, which are composed of sub-phones, etc. In practice, since the mapping from words to phones is fixed by a phonetic dictionary, and the mapping from phones to sub-phones is fixed by assuming a 3-state left-to-right topology, the whole hierarchy can be “collapsed” into a flat HMM, with appropriate parameter tying to represent the fact that bottom levels of the hierarchy can be reused in different higher level contexts. However, if one were interested in learning the hierarchical decomposition, an HHMM might be useful. See Chapter ?? for more on speech recognition.

Another application of HHMMs is to map learning by mobile robots. It is natural to think of an indoor office-like environment as composed of floors, which are composed of corridors and rooms, which are composed of finer-grained locations. For this application, it is necessary to modify the model somewhat. Firstly, we typically condition all state transitions on the action performed by the robot. Secondly, and more subtly, when we leave one abstract state (e.g., corridor), the new concrete state (e.g., location within corridor) depends on the old concrete state, i.e., not all the old context is “popped off the stack” upon exiting. (In other words, the corridor models are not context free.) Instead, we condition the new starting state on some function of the old state. We leave the details as an exercise.

2.10 State-space models

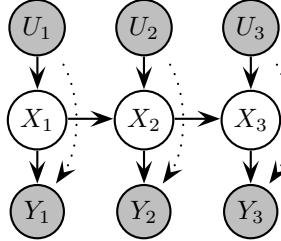


Figure 19: An HMM/SSM with input. The dotted arcs are optional, and reflect the fact that the input may affect the output directly.

The graph structure for an SSM looks identical to that for an HMM, since it makes the same conditional independence assumptions. Typically one also includes an input node, as in Figure 19. In an SSM, all the nodes are continuous, and all the CPDs are linear-Gaussian, i.e.,

$$\begin{aligned} P(X_t = x_t | X_{t-1} = x_{t-1}, U_t = u) &= \mathcal{N}(x_t; Ax_{t-1} + Bu, Q) \\ P(Y_t = y | X_t = x, U_t = u) &= \mathcal{N}(y; Cx + Du, R) \end{aligned}$$

We do not need to define $P(U_t)$, since this is an exogenous input variable that is always observed.

In the case of SSMs, unlike for HMMs, there is a one-to-one correspondence between sparse graphical structure and sparse parametric structure. In particular, $A(i, j) > 0$ iff there is a directed arc from $X_{t-1}^{(i)}$ to $X_t^{(j)}$, and similarly for the other regression matrices, B , C and D . For the covariance matrices, we make use of the fact that, in an *undirected* Gaussian graphical model, zeros in the precision (inverse covariance) matrix correspond to absent arcs. Hence $Q^{-1}(i, j) > 0$ iff there is an undirected arc from $X_t^{(i)}$ to $X_t^{(j)}$ within the same time slice.

For example, consider a vector auto-regressive process of order 2:

$$X_t = A_1 X_{t-1} + A_2 X_{t-2} + \epsilon_t$$

where $\epsilon_t \sim \mathcal{N}(0, \Sigma)$. (We ignore the U_t and Y_t variables for simplicity.) Suppose the parameters are as follows.

$$A_1 = \begin{pmatrix} \frac{3}{5} & 0 & \frac{1}{5} & 0 & 0 \\ 0 & \frac{3}{5} & 0 & -\frac{1}{5} & 0 \\ \frac{2}{5} & \frac{1}{3} & \frac{3}{5} & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2} & \frac{1}{5} \\ 0 & 0 & \frac{1}{5} & 0 & \frac{2}{5} \end{pmatrix}, \quad A_2 = \begin{pmatrix} 0 & 0 & -\frac{1}{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{5} & 0 & \frac{1}{3} \\ 0 & 0 & 0 & 0 & -\frac{1}{5} \end{pmatrix}$$

and

$$\Sigma = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & 0 & 0 \\ \frac{1}{2} & 1 & -\frac{1}{3} & 0 & 0 \\ \frac{1}{3} & -\frac{1}{3} & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad \Sigma^{-1} = \begin{pmatrix} 2.13 & -1.47 & -1.2 & 0 & 0 \\ -1.47 & 2.13 & 1.2 & 0 & 0 \\ -1.2 & 1.2 & 1.8 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Then the resulting dynamic chain graph is illustrated in Figure 20.

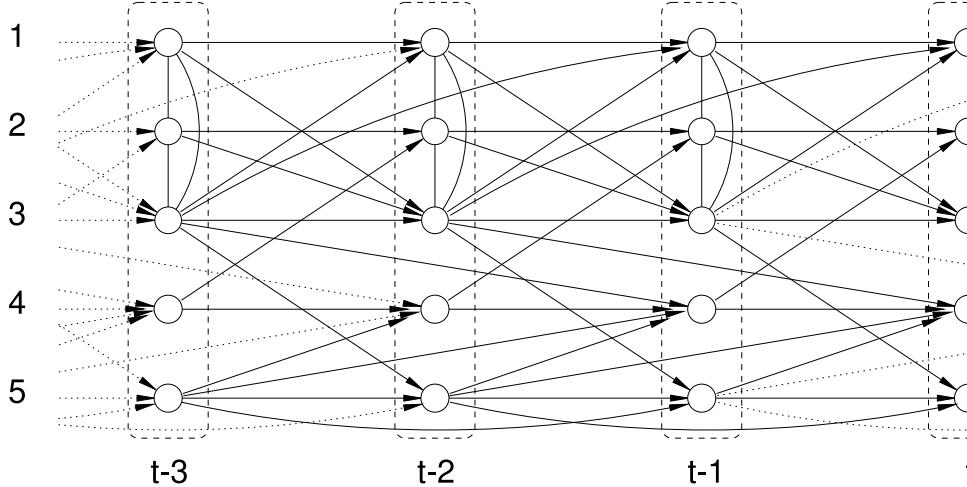


Figure 20: A VAR(2) process represented as a dynamic chain graph. From [DE00].

2.11 Joint state-parameter estimation

The problem of online parameter estimation can be represented by adding the parameters to the state space.

For example, suppose we want to recursively (i.e., sequentially) estimate the coefficients, α , in a linear regression model, where we assume the noise level, R , is known. This can be modelled as shown in Figure 21. The CPDs are as follows:

$$\begin{aligned} P(\alpha_0) &= \mathcal{N}(\alpha_0; 0, \infty I) \\ P(y_t | x_t, \alpha_t) &= \mathcal{N}(y_t; x_t' \alpha_t, R) \\ P(\alpha_t | \alpha_{t-1}) &= \mathcal{N}(\alpha_t; \alpha_{t-1}, 0) \end{aligned}$$

(We do not need to specify the CPD for x_t , since in linear regression, we assume the input is always known, so it suffices to use a conditional likelihood model.) The infinite variance for α_0 is an uninformative prior. The zero variance for α_t reflects the fact that the parameter is constant. If we made the variance non-zero, we could allow slow changes in the parameters, and hence track non-stationarities.

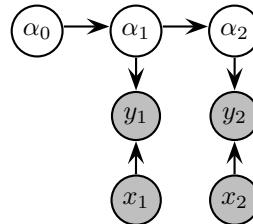


Figure 21: A DBN for the recursive least squares problem.

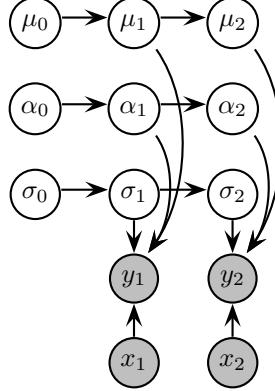


Figure 22: A DBN for sequential Bayesian non-linear regression.

We can perform exact inference in this model using the Kalman filter, where α_t is the hidden state, and we use a time-varying output matrix $C_t = x'_t$; we set the transition matrix to $A = I$, the transition noise to $K = 0$, and the observation noise to R . This is just the recursive least squares algorithm discussed in Section ??.

Now consider a non-linear regression model:

$$y_t = \sum_{j=1}^k a_{j,t} \phi(||x_t - \mu_{j,t}||) + b_t + \beta'_t x_t + \epsilon_t$$

where $\epsilon_t \sim \mathcal{N}(0, \sigma_t)$ is a noise term and $\phi(||x_t - \mu_{j,t}||)$ is e.g., a radial basis function (RBF); if $k = 0$, this reduces to a linear regression model. We can write this in vector-matrix form as follows:

$$y_t = D(\mu_t, x_t)\alpha_t + n_t$$

where $\alpha_t = (a_t, b_t, \beta_t)$ are all the weights. The resulting DBN is shown in Figure 22. If we allow the parameters to change over time (modelled by a random walk), the CPDs are

$$\begin{aligned} P(\mu_t | \mu_{t-1}) &= \mathcal{N}(\mu_t; \mu_{t-1}, \delta_\mu I) \\ P(\alpha_t | \alpha_{t-1}) &= \mathcal{N}(\alpha_t; \alpha_{t-1}, \delta_\alpha I) \\ P(\log \sigma_t | \log \sigma_{t-1}) &= \mathcal{N}(\log \sigma_t; \log \sigma_{t-1}, \delta_\sigma I) \\ P(y_t | x_t, \alpha_t, \mu_t) &= \mathcal{N}(y_t; D(\mu_t, x_t)\alpha_t, \sigma_t) \end{aligned}$$

(I have omitted the priors at time 1 for simplicity.)

Although this model is linear in the α parameters, it is not linear in σ_t or μ_t . Hence it is not possible to perform exact inference (i.e., Bayesian parameter learning) in this model. However, it is possible to sample σ_t and μ_t , and then apply a conditional Kalman filter to α_t : see Section 5.4.3. Other approaches to inference in non-linear models will be discussed in Section 5. Methods for selecting the number of basis functions online will be discussed in Section 5.5.1.

2.12 Switching SSMs

In Figure 23 we show a switching SSM, also called a switching linear dynamical system (LDS), a jump-Markov model, a jump-linear system, a conditional dynamic linear model (DLM), etc. The basic idea is that the model can switch between different kinds of dynamical ‘‘modes’’ or ‘‘regimes’’. (The resulting piece-wise linearity is one way to approximate non-linear dynamics.) The dynamics of the modes themselves are governed by a discrete-state Markov chain. Hence the CPDs are as follows:

$$\begin{aligned} P(X_t = x_t | X_{t-1} = x_{t-1}, S_t = i) &= \mathcal{N}(x_t; A_i x_{t-1}, Q_i) \\ P(Y_t = y | X_t = x) &= \mathcal{N}(y; Cx, R) \\ P(S_t = j | S_{t-1} = i) &= M(i, j) \end{aligned}$$

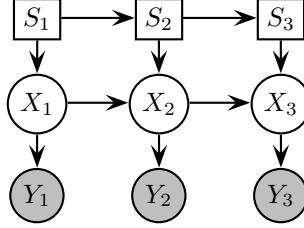


Figure 23: A switching state-space model. In this figure, square nodes are discrete, round nodes are continuous. We have omitted the input U for simplicity.

Since this model has both discrete and continuous hidden variables, it is sometimes called a hybrid DBN. Exact inference in such model is very difficult, as we discuss in Section 5.2.4. Contrast this with an auto-regressive HMM (Section 2.2) in which all the hidden variables are discrete, making exact inference easy.

In addition to switching X_t , it is possible to switch Y_t . This can be used to represent $P(Y_t|X_t)$ as a mixture, which can approximate non-Gaussian observation densities. For example, in order to be robust to outliers, it is common to assume $P(Y_t|X_t)$ is a mixture of a Gaussian distribution and a uniform distribution (which can be approximated using a Gaussian with very large covariance). We will see some other applications of this below.

2.12.1 Fault diagnosis

One important application of switching SSMs is for diagnosing faults. For example, consider the “two tank” system in Figure 24, a benchmark problem in the fault diagnosis community (although one typically considers n tanks, for $n \gg 2$). This is a nonlinear system, since flow = pressure / resistance (or flow = pressure \times conductance). More problematically, the values of the resistances can slowly drift, or change discontinuously due to burst pipes. Also, the sensors can fail intermittently and give erroneous results. We can model all of this using a DBN as shown in Figure 25.

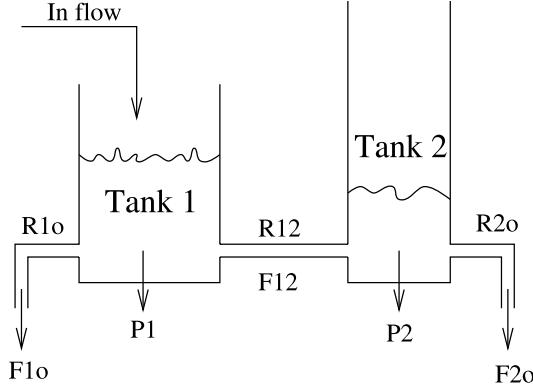


Figure 24: The two-tank system. The goal is to infer when pipes are blocked or have burst, or sensors have broken, from (noisy) observations of the flow out of tank 1, F_{1o} , out of tank 2, F_{2o} , or between tanks 1 and 2, F_{12} . R_{1o} is a hidden variable representing the resistance of the pipe out of tank 1, P_1 is a hidden variable representing the pressure in tank 1, etc. From Figure 11 of [KL01].

2.12.2 Data association

Another important application of switching SSMs is for modelling the data association (correspondence) problem. In this problem, we are interested in tracking the state of several “objects” $X_t^{(1)}, \dots, X_t^{(D)}$. At each time step, we get an observation, Y_t , but we do not know which (if any) of the objects caused it.

The data association problem is extremely common. For example, consider visually tracking a person as they move around a cluttered room. At each frame, we must decide which body part (if any) caused which pixel. Another

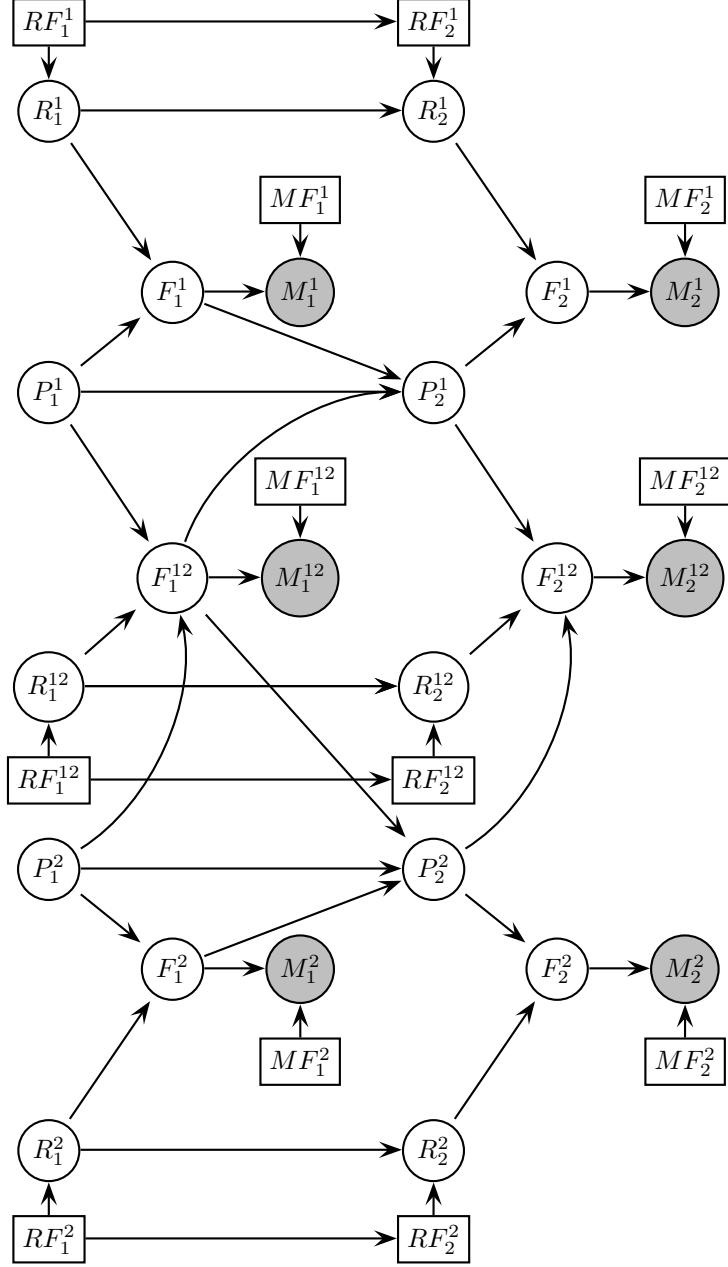


Figure 25: The two tanks system of Figure 24 modelled as a 2TBN. For multiple tanks, the structure would be repeated in the obvious way. Discrete nodes are squares, continuous nodes are circles. Abbreviations: R = resistance, P = pressure, F = flow, M = measurement, RF = resistance failure, MF = measurement failure. The flow out of tank i at time t , F_t^i , equals the pressure in tank i , P_t^i , divided by the resistance, R_t^i ; M_t^i is a noisy measurement of this flow rate; this sensor may fail if MF_t^i turns on. The new pressure, P_{t+1}^i , depends on the old pressure, P_t^i , and the old outflow, F_t^i . Adapted from Figure 12 of [KL01].

example concerns tracking missiles using radar. At each scan, we must decide which missile (if any) caused which “blip” on the screen. This is much easier than visual tracking because the sensor measurements are sparse.

A simple model for this is shown in Figure 26. S_t is a latent variable which specifies the identity of the source of the observation Y_t . It is a switching parent, and is used to “select” which of the hidden chains to “pass through” to the

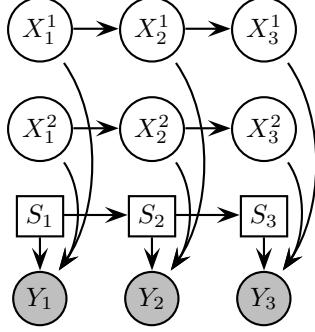


Figure 26: A DBN for the data association problem. The observation Y_t comes from one of two independent sources, as determined by the multiplexer node S_t . Square nodes are discrete, round nodes are continuous.

output, i.e., the CPD for $P(Y_t|X_t^{1:D}, S_t)$ is a (Gaussian) multiplexer:

$$P(Y_t = y|S_t = i, X_t^{(1)}, \dots, X_t^{(D)}) = \mathcal{N}(y; W_i X_t^{(i)} + \mu_i, R_i)$$

Because of its military importance, a lot of effort has been expended on the problem of tracking multiple targets in clutter. See Section 5.5 for further discussion.

3 Inference

Having seen a large variety of different DBNs, we now discuss how to perform inference in such models. There are a variety of inference problems we might be interested in (see Figure 27 for a summary):

Filtering Computing $P(X_t|y_{1:t})$, i.e., monitoring (tracking) the state over time.

Prediction Computing $P(X_{t+h}|y_{1:t})$ for some horizon $h > 0$ into the future.

Fixed-lag smoothing Computing $P(X_{t-l}|y_{1:t})$, i.e., estimating what happened $l > 0$ steps in the past given all the evidence up to the present (hindsight).

Fixed-interval smoothing (offline) Computing $P(X_t|y_{1:T})$. This is used as a subroutine for offline training.

Viterbi decoding Computing $\arg \max_{x_{1:t}} P(x_{1:t}|y_{1:t})$, i.e., figuring out the most likely cause/ explanation of the observed data.

Classification Computing $P(y_{1:t}) = \sum_{x_{1:t}} P(x_{1:t}, y_{1:t})$. This can be used to compute the likelihood of a sequence under different models.

We will focus on online filtering and offline smoothing; the other problems can be solved by similar methods.⁴

It will be helpful to distinguish models in which all the hidden nodes are discrete from models in which some (or all) hidden nodes are continuous, since they require different solution techniques (just as we saw that inference in an HMM required different techniques than inference in an SSM). We will find that exact inference in most DBNs is computationally intractable, so we shall consider a variety of different approximation techniques.

4 Exact inference in discrete-state models

In this section, we discuss exact filtering and smoothing algorithms for DBNs in which all the hidden variables are discrete. (Continuous nodes that are observed do not cause a problem, no matter what their distribution, since they only affect the inference by means of the conditional likelihood, c.f., the $O_t(i, i) = P(y_t|X_t = i)$ term for HMMs.)

⁴For example, the quantity needed to classify a sequence, $P(y_{1:t})$, is computed as a by-product of filtering. Also, any filtering algorithm can be converted to a Viterbi algorithm by replacing the “sum” operator with the “max” operator. Finally, any offline smoothing algorithm can be made online by simply using a sliding window.

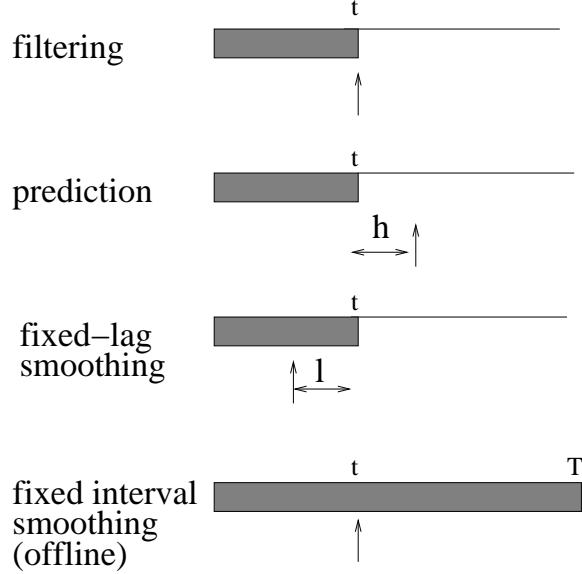


Figure 27: The main kinds of inference for DBNs. The shaded region is the interval for which we have data. The arrow represents the time step at which we want to perform inference. t is the current time, and T is the sequence length. h is a prediction horizon and l is a time lag.

4.1 Forwards-backwards algorithm

The simplest inference method for a discrete-state DBN is to convert it to an HMM, and then to apply the forwards-backwards algorithm. If there are N_h hidden variables per slice, and each such variable can have up to M values, the resulting HMM will have $S = M^{N_h}$ states. Provided S is not too large, this is the method of choice, since the forwards-backwards algorithm is exact, and is very simple to implement. Unfortunately, in general the number of states will be too large, since it is exponential in the number of hidden variables; hence we must look for more efficient methods.

4.2 Unrolled junction tree

The second simplest inference method is to unroll the DBN for T slices (where T is the length of the sequence) and then to apply any static Bayes net inference algorithm. For filtering, it is natural to apply the variable elimination algorithm (Chapter ??). This need only keep two slices in memory at a time: starting with slice 1, we add slice 2, then marginalize out slice 1, then add slice 3, etc. For smoothing, it is more efficient to use the junction tree algorithm, since most of the work in computing $P(Q_T|y_{1:T})$ can be reused when computing $P(Q_{T-1}|y_{1:T})$, etc. Later we will see how to modify the junction tree algorithm to only work with two slices at a time (which is essential for online filtering), but in this section, we consider the more naive approach of treating the network as a single large unrolled graph.

Unfortunately, when we create a junction tree from an unrolled DBN, the cliques tend to be very large, often making exact inference intractable. In particular, for every time slice (except possibly near the beginning and end of the sequence), there will usually be a clique that contains all the nodes within that slice that have inter-slice connections. (We will be more precise below). See Figure 28 for an example. The intuitive reason for this is illustrated in Figure 29: even if nodes within a slice are not directly correlated, they will eventually become correlated by virtue of sharing common ancestors in the past.

4.2.1 Constrained elimination orderings

The size of the maximum clique (or largest factor in the variable elimination algorithm) depends on the elimination ordering. Let us define $m(\pi)$ to be the size of the maximum clique induced by ordering π . (A related quantity is the

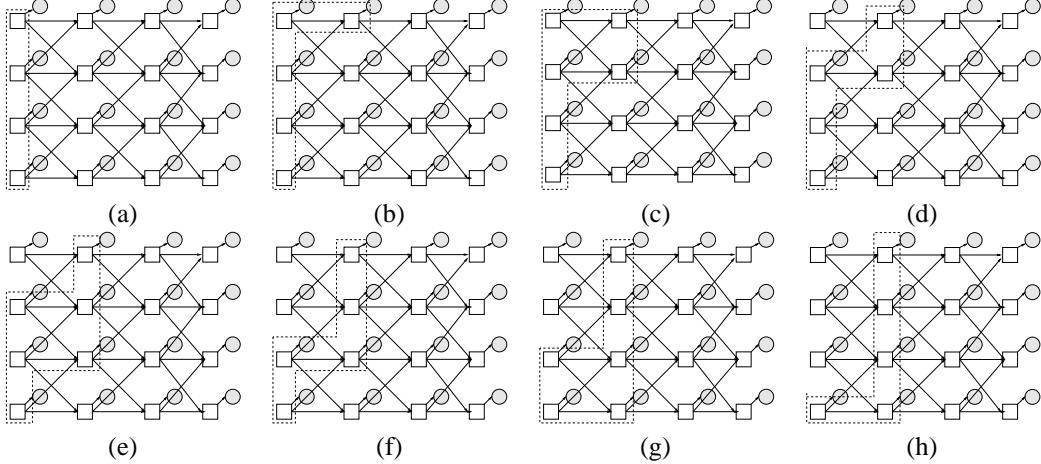


Figure 28: Some of the (non maximal) cliques in the junction tree constructed from a coupled HMM with 4 chains. The junction tree will contain these cliques connected together in a chain; the cliques containing $(Q_t^{(i)}, Y_t^{(i)})$ (not shown) “hang off” this “backbone” c.f., Figure 36. (a) The clique initially contains $Q_{t-1}^{(1:4)}$; then we update it as follows: (b) add $Q_t^{(1)}$; (c) add $Q_t^{(2)}$; (d) remove $Q_{t-1}^{(1)}$; (e) add $Q_t^{(3)}$; (f) remove $Q_{t-1}^{(2)}$; (g) add $Q_t^{(4)}$; (h) remove $Q_{t-1}^{(3)}$; (last step not shown) remove $Q_{t-1}^{(4)}$ to yield $Q_t^{(1:4)}$. It is possible to combine consecutive add/remove steps, resulting in just the maximal cliques shown in Figures c, e and g. This is what the standard junction tree algorithm does.

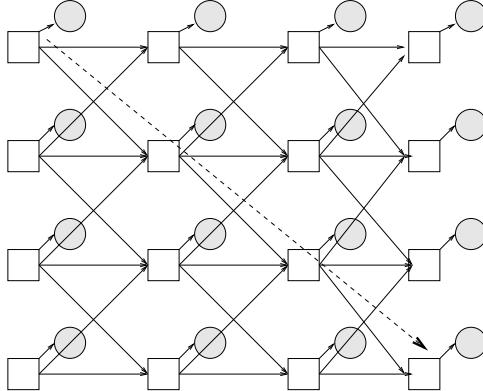


Figure 29: A coupled HMM with 4 chains. Even though chain 1 is not directly connected to chain 4, they become correlated once we unroll the DBN, as indicated by the dotted line. That inference in this model is intractable should not be surprising, since the graph is very similar to a 2D lattice MRF.

width of an ordering, defined as $w(\pi) = m(\pi) - 1$.) The max clique size of a graph, $m(G)$, is defined as $m(\pi^*(G))$, where $\pi^*(G)$ is the optimal elimination ordering for G . Exact inference takes time and space exponential in $m(\pi)$, as we saw in Chapter ??.

A natural ordering is to work left-to-right, i.e., we eliminate all nodes in slice t (in some optimal order) before any in slice $t+1$. Unfortunately, such a constrained ordering is not always globally optimal. For example, Figure 30 shows a DBN where $m(\pi^*) = 2$ (since the graph is a set of trees). However, if we unroll the graph for $T \geq 4$ slices, and eliminate all the nodes in slice t before eliminating any nodes in slice $t+1$, we find $m(\pi) = 5$. This is a consequence of the following theorem:

Theorem (Constrained elimination ordering) [RTL76]. Let A_1, \dots, A_n be an elimination sequence triangulating the (moral) graph G , and let A_i, A_k be two non-neighbors in G , $i < k$. Then the elimination sequence introduces the fill-in $A_i - A_k$ iff there is a path $A_i - A_j - \dots - A_k$ such that all intermediate nodes A_j are eliminated before A_i .

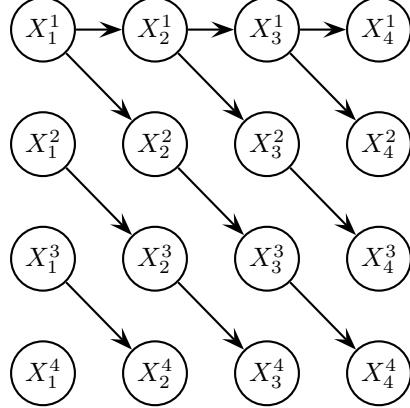


Figure 30: A DBN in which all the nodes in the last slice become connected when we eliminate nodes in a slice-by-slice order π , implying $m(\pi) = 4 + 1 = 5$, even though, using an unconstrained order that exploits the tree structure, $m(\pi^*) = 2$.

To apply this theorem to the example in Figure 30, let $A_i = X_t^{(i)}$ and $A_k = X_t^{(k)}$. Suppose we first eliminate all the $X_\tau^{(i)}$ for all $\tau < t$; these become the A_j 's. Since A_i is connected to A_k via some (undirected) path through the past A_j 's (if $t \geq 4$), we will add a fill-in between them. Hence all the nodes in slice t will become connected.

To precisely characterize the consequences of using a slice-by-slice elimination ordering, let us first make the following definition.

Definition. Let the set of *temporal arcs* between slices $t - 1$ and t be denoted by $E^{\text{tmp}}(t) = \{(u, v) \in E | u \in V_{t-1}, v \in V_t\}$, where V_t are the nodes in slice t . The *incoming interface* I_t^\leftarrow is defined as all nodes v s.t. v , or one of its children, has a parent in slice $t - 1$, i.e., $I_t^\leftarrow = \{v \in V_t | (u, v) \in E^{\text{tmp}}(t) \text{ or } \exists w \in \text{ch}(v) : (u, w) \in E^{\text{tmp}}(t), u \in V_{t-1}\}$. See Figure 31 for an example. (We call it the *incoming* interface to distinguish it from the *outgoing* interface, which we shall define shortly.)

Now we can put tight lower and upper bounds on the complexity of inference using a slice-by-slice elimination ordering.

Theorem. Let G be a DBN with N nodes per slice, unrolled for $T > 1$ slices, which forms a single connected component.⁵ If π is any slice-by-slice elimination ordering for G , then $|I^\leftarrow| + 1 \leq m(\pi) \leq |I^\leftarrow| + N$, and the bounds are tight.

Proof. When we eliminate the nodes in slice t , the only variables that are involved are $V_t \cup I_{t+1}^\leftarrow$. Hence $m(\pi) \leq |V_t \cup I_{t+1}^\leftarrow| = N + |I^\leftarrow|$. Figure 32 shows a DBN where $I^\leftarrow = N$ and $m(\pi^*) = 2N$; since $m(\pi) \leq m(\pi^*)$, we see that the upper bound is tight. To establish the lower bound, note that since the graph is a single connected component, at least one of the nodes in the incoming interface must have a parent in the previous slice; hence there must be some stage in the elimination process which contains this “old” node plus all the interface nodes, so $|I^\leftarrow| + 1 \leq m(\pi)$. To see that the bound is tight, consider a Markov chain: this has $|I^\leftarrow| = 1$, so a max clique size of 2 can be achieved. ■

Hence the constrained junction tree algorithm takes $\Omega(M^{|I^\leftarrow|+1})$ operations per time step (assuming all nodes in the incoming interface can have M possible values). Note that this may be exponentially faster than the time required by the forwards-backwards algorithm, which performs $O(M^2 N_h)$ operations per time step; nevertheless, it is still exponential in $|I^\leftarrow|$.

⁵Figure 33 shows a counter example to the theorem when the DBN is not a single connected component.

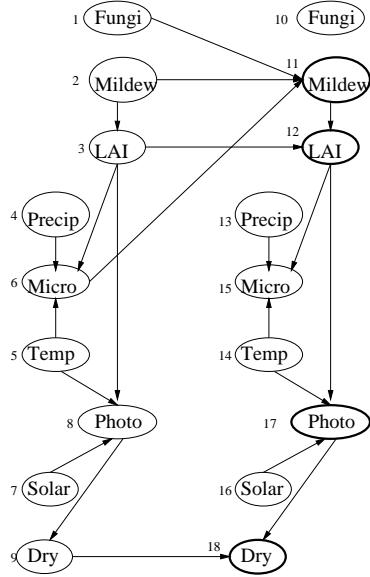


Figure 31: The Mildew DBN, designed for forecasting the gross yield of wheat based on climatic data, observations of leaf area index (LAI) and extension of mildew, and knowledge of amount of fungicides used and time of usage [Kja95]. The nodes in the incoming interface are shown in bold outline. Note how the transient node photo is in the incoming interface, because it is the parent of the persistent node dry, which has an incoming temporal arc.

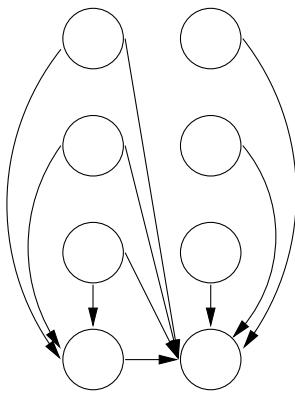


Figure 32: A worst case DBN from a complexity point of view, since we must create a clique which contains all nodes in both slices. Hence the max clique size is $2N$.

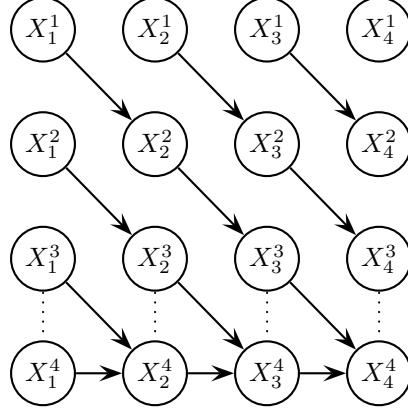


Figure 33: A DBN in which the nodes in the last slice do *not* become connected even when we use a slice-by-slice elimination ordering; hence $m(\pi^*) = 3 < |I^-| + 1 = 4$, contrary to the theorem. This is because the nodes in the last slice do not belong to one connected component. Contrast this with Figure 30. Dotted lines represent moralization arcs.

4.2.2 Unconstrained elimination orderings

What happens if we use an unconstrained elimination ordering? In principle we can do better. However, in practice, this is rarely the case. Recall that finding the optimal elimination ordering is NP-hard. Hence most elimination orderings are computed using local search algorithms. Empirically it has been found by several researchers that providing a constrained space to search in often improves the quality of the local optima that are found. For example, an unconstrained algorithm might choose to eliminate the first node in every slice, then the second, etc., creating wide “horizontal” cliques that span many time slices, as opposed to narrow “vertical” cliques that are temporally localized. Such horizontal orderings usually have larger clique sizes than vertical orderings (for large enough T), and hence would not be chosen by a good search algorithm. However, a greedy search algorithm (e.g., based on the min-fill heuristic) may easily choose such a sub-optimal ordering.

An additional problem with unconstrained orderings is the following. We don’t want to have to compute a new elimination ordering every time we change the length of the sequence, because it is too slow. Instead we would like to triangulate a fixed size DBN, identify the cliques in the resulting junction tree, and then reuse them for all sequence lengths by copying the repeating structure. In order to guarantee that such repeating structure occurs in the junction tree, we must use a temporally constrained elimination ordering. Even then, the book-keeping involved with dynamically changing the structure of the junction tree becomes quite tricky.

4.3 The frontier algorithm

An alternative approach, which turns out to be simpler and better suited to online inference, is to think explicitly in terms of time slices. The forwards-backwards algorithm for HMMs works because conditioning on Q_t d-separates the past from the future. This idea can be generalized to DBNs by noting that all the nodes in a slice d-separate the past from the future. We will call this set the “frontier” and denote it by Z_t . Hence the belief state, $P(Z_t|y_{1:t})$, can act as a sufficient statistic for filtering; for smoothing, we can define an analogous quantity for the backwards direction.

The filtering problem reduces to computing $P(Z_t|y_{1:t})$ from $P(Z_{t-1}|y_{1:t-1})$. In an HMM, this can be done with a single matrix multiply, but we wish to avoid constructing, yet alone multiplying by, an $O(S \times S)$ matrix, where $S = M^{N_h}$. The following rules will advance the frontier one node at a time such that at each step the frontier d-separates all the nodes to its left from the nodes to its right: (1) a node can be added to the frontier if all its parents are already in the frontier, and (2) a node can be removed from the frontier as soon as all its children have been added. See Figure 28 for an example. Adding a node means multiplying its CPD onto the frontier distribution, and removing a node means marginalizing it out of the frontier distribution. We leave the details as an exercise, since we are about to develop a more efficient algorithm based on similar ideas.

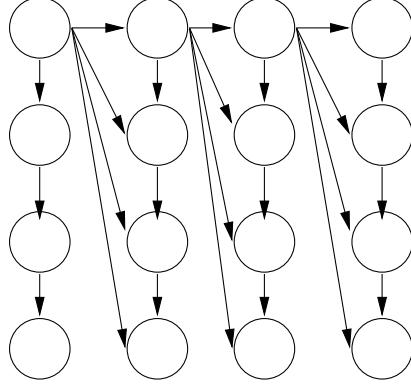


Figure 34: This graph has a max clique size of 3, an outgoing interface of size 1, and an incoming interface of size 4. Based on Figure 2 of [Dar01].

4.4 The interface algorithm

The frontier distribution, $P(Z_t|y_{1:t})$, has size $O(M^{N_h})$. Fortunately, it is easy to see that can exclude from the frontier all nodes that do not have any children in the next slice. We will prove this below. First let us make the following definition:

Definition. The *outgoing interface* I_t^\rightarrow is the set of nodes which have children in the next slice, i.e., $I_t^\rightarrow \stackrel{\text{def}}{=} \{u \in V_t | (u, v) \in E^{\text{tmp}}(t+1), v \in V_{t+1}\}$.

Now we prove that the outgoing interface d-separates the past from the future, and hence can be used as a sufficient statistic for inference.

Theorem. $(V_{1:t-1} \cup (V_t \setminus I_t^\rightarrow)) \perp V_{t+1:T} | I_t^\rightarrow$, i.e., the outgoing interface d-separates the past from the future, where the past is all the nodes in slices prior to t , plus the non-interface nodes in slice t , and the future is all nodes in slices after t .

Proof. Let I be a node in the outgoing interface, connected to a node P in the past and a node F in the future (which must be a child of I , by definition). If P is a parent, the graph looks like this: $P \rightarrow I \rightarrow F$. If P is a child, the graph looks like this: $P \leftarrow I \rightarrow F$. Either way, we have $P \perp F | I$, since I is never at the bottom of a v-structure. Since all paths between any node in the past and any node in the future are blocked by some node in the outgoing interface, the result follows. ■

Figure 34 shows a DBN for which $|I_t^\rightarrow| = 1 < |I_t^\leftarrow| = 4$, but Figure 31 shows a DBN for which $|I_t^\rightarrow| = 5 > |I_t^\leftarrow| = 4$. Although the outgoing interface is not necessarily smaller than the incoming interface, in practice it often is. (If all temporal arcs are persistence arcs (arcs of the form $Q_{t-1}^{(i)} \rightarrow Q_t^{(i)}$), then it is easy to see that the outgoing interface is never larger than the incoming interface.) In addition, the outgoing interface lends itself to simpler algorithms, which are better suited to online inference, as we will see below.

4.4.1 Generalized $\alpha - \gamma$ algorithm

The filtering problem now reduces to computing $P(I_t^\rightarrow | y_{1:t})$ from $P(I_{t-1}^\rightarrow | y_{1:t-1})$. To do this, we will use a modified junction tree on a modified 2TBN; this will allow us to exploit any sparse structure that may exist within a slice (something the frontier algorithm does not do). We will call the modified 2TBN a 1.5DBN, since it contains “one-and-a-half” slices. Specifically, the 1.5DBN contains all the nodes in slice 2, but only the outgoing interface nodes from slice 1. See Figure 35 for an example.

Let us denote the 1.5DBN by $G_{1.5}$. To convert this to a modified junction tree, we proceed as follows. First we moralize $G_{1.5}$ to create $M(G_{1.5})$. Then we add undirected edges between all nodes in I_{t-1}^\rightarrow and between all nodes in I_t^\rightarrow to create $M^+(G_{1.5})$; this ensures there will be cliques that are large enough to contain $P(I_{t-1}^\rightarrow | y_{1:t-1})$ and

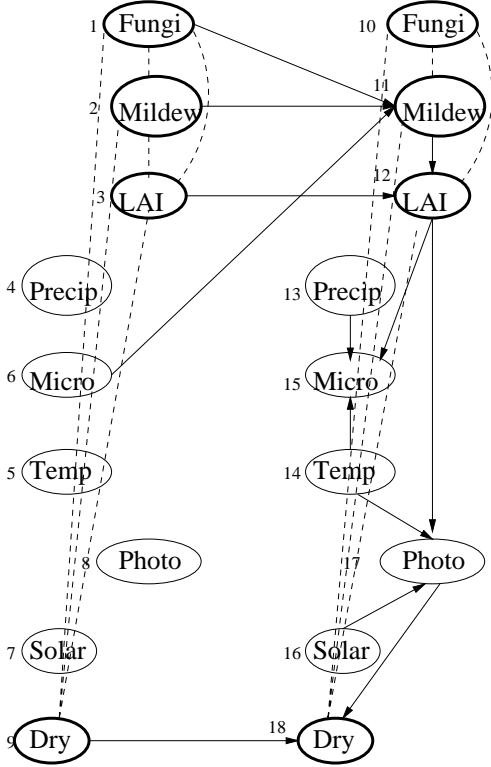


Figure 35: The Mildew 1.5DBN. Nodes in the outgoing interface are shown in bold outline. Nodes which are not in the outgoing interface of slice 1 are disconnected, and can be removed. Dashed arcs are added to ensure the outgoing interfaces are fully connected.

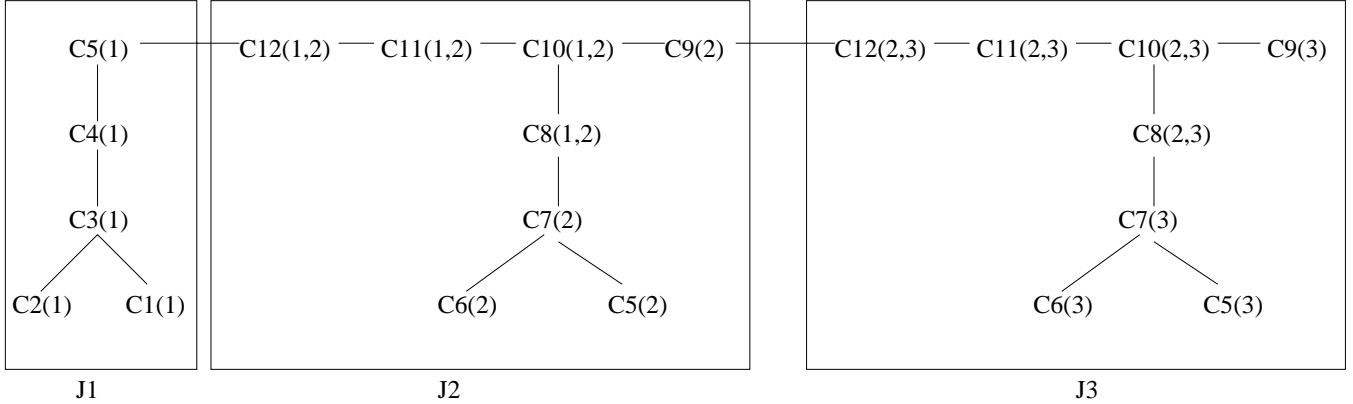


Figure 36: The junction tree constructed from 3 slices of the DBN in Figure 35. The notation $Cn(t, t+1)$ means clique n containing nodes from slices t and $t + 1$. Thus $C12(2, 3)$ is the same as $C12(1, 2)$, except all the node numbers are shifted up by N , the number of nodes per slice. (Cliques in J_t , $t > 1$, start with number 5 because cliques 1 to 4 correspond to disconnected nodes in slice 1 of the 1.5DBN, and can be ignored.) For J_t , $t > 1$, the incoming clique is $C12$ and the outgoing clique is $C9$; for J_1 , the incoming/outgoing clique is $C5$. α_t represents the potential on the separator between J_t and J_{t+1} on the forwards pass; γ_t represents this potential on the backwards pass.

$P(I_t^\rightarrow | y_{1:t})$. Then we triangulate using an unconstrained elimination ordering to create $T(M^+(G_{1.5}))$. Finally we create a junction tree from the maximal cliques of $T(M^+(G_{1.5}))$. If this contains all the nodes in slice t and the interface nodes in slice $t-1$, we will call it J_t . (J_1 just contains slice 1, of course.) Let in-clq denote the clique in J_t which contains I_{t-1}^\rightarrow , and out-clq denote the clique which contains I_t^\rightarrow . See Figure 36 for an example.

We now define an abstract operator, $(\alpha_t, J_t) = \text{fwd}(\alpha_{t-1}, y_t)$, that does one step of Bayesian updating, i.e., which computes $\alpha_t \stackrel{\text{def}}{=} P(I_t^\rightarrow | y_{1:t})$ from $\alpha_{t-1} = P(I_{t-1}^\rightarrow | y_{1:t-1})$, and which returns the potentials ϕ of all the cliques and separators in J_t : these will be needed in the backwards smoothing pass. The forwards operator is implemented as follows.

1. Initialize $\phi(J_t)$ using the CPDs from slice 2 and the evidence y_t .
2. $\phi_{in} = \phi_{in} * \alpha_{t-1}$.
3. Collect to out-clq.
4. $\alpha_t = \phi_{out} \downarrow I_t^\rightarrow$.

The \downarrow operator represents (sum or max) marginalization; thus $\phi_{out} \downarrow I_t^\rightarrow$ means marginalize the clique potential on out-clq onto the forwards interface. (Note that only the out-clq will have been conditioned on all the evidence; hence to get filtered estimates of all nodes in slice t , a distribute evidence pass will be required within J_t if we omit the backwards pass.)

The base case, $(\alpha_1, J_1) = \text{fwd1}(y_1)$, initializes with the CPDs from slice 1 and skips the multiplication by α_{t-1} .

We can define a backwards operator, $(\gamma_{t-1}, J_t) = \text{back}(\gamma_t, \alpha_t, J_t)$, as follows.

1. $\phi_{out} = \phi_{out} * \frac{\gamma_t}{\alpha_t}$.
2. Distribute from out-clq.
3. $\gamma_{t-1} = \phi_{in} \downarrow I_{t-1}^\rightarrow$.

(When $t = T$, the first step of the back operator has no effect, since $\alpha_T = \gamma_T$.) This is the analog of the equation (12.45) for HMMs:

$$\xi(q_{t-1}, q_t) = \frac{\alpha(q_{t-1})P(y_t|q_t)A(q_{t-1}, q_t)\gamma(q_t)}{\alpha(q_t)}$$

since $\alpha(q_{t-1})P(y_t|q_t)A(q_{t-1}, q_t)$ is included in J_t , and the back operator multiplies by $\gamma(q_t)/\alpha(q_t)$.

The base case, $J_1 = \text{back1}(\gamma_1, \alpha_1, J_1)$, simply omits the final marginalization onto I_{t-1}^\rightarrow . The effect is to distribute the information contained in and to the right of the interface to all the other nodes in the first slice.

Finally, we can define a smoothing algorithm as follows.

1. $(\alpha_1, J_1) = \text{fwd1}(y_1)$
2. For $t = 2, \dots, T$,
 $(\alpha_t, J_t) = \text{fwd}(\alpha_{t-1}, y_t)$
3. $b_{T|T} = \alpha_t$
4. For $t = T, \dots, 2$
 $(\gamma_{t-1}, J_t) = \text{back}(\gamma_t, \alpha_t, J_t)$
5. $J_1 = \text{back1}(\gamma_1, \alpha_1, J_1)$

We can compute any single node marginal $P(Q_t^{(i)} | y_{1:T})$ by marginalizing γ_t . Similarly, we can compute any family marginal $P(Q_t^{(i)}, \text{Pa}(Q_t^{(i)}) | y_{1:T})$ by marginalizing the appropriate potential in J_t .

4.4.2 Generalized $\alpha - \beta$ algorithm

The above algorithm is the analog of the $\alpha - \gamma$ algorithm for HMMs. It is also useful (as we discuss in Section 4.4.3) to define an analog to the $\alpha - \beta$ algorithm. We can do this by defining a modified backwards operator, $\beta_{t-1} = \text{back}(\beta_t, y_t)$, as follows.

1. Initialize $\phi(J_t)$ using the CPDs from slice 2 and the evidence y_t .
2. Collect to in-clq.
3. $\beta_{t-1} = \phi_{in} \downarrow I_{t-1}^{\rightarrow}$.

In addition, we need to define a new operator, $J_t = \text{combine}(\alpha_{t-1}, \beta_t, y_t)$, that combines the α and β messages, and returns all the clique potentials.

1. Initialize $\phi(J_t)$ using the CPDs for slice 2 and y_t .
2. $\phi_{in} = \phi_{in} * \alpha_{t-1}$
3. $\phi_{out} = \phi_{out} * \beta_t$
4. Calibrate J_t .

This is the analog of the equation (12.44) for HMMs:

$$\xi(q_{t-1}, q_t) = \frac{\alpha(q_{t-1})P(y_t|q_t)A(q_{t-1}, q_t)\beta(q_t)}{P(y)}$$

Note that $y_{1:t-1}$ has already been included in α_{t-1} , and $y_{t+1:T}$ has already been included in β_t . After calibration, all cliques in J_t have seen all the evidence.

The base case, $J_1 = \text{combine1}(\alpha_1, \beta_1, y_1)$, is defined as follows.

1. Initialize $\phi(J_1)$ using the CPDs for slice 1 and y_1 .
2. $\phi_{out} = \phi_{out} * \beta_1$
3. Callibrate the tree.

This ignores α_1 , and recomputes it by collecting to the root. The reason is that we need access to all the clique potentials, not just α_1 . The root absorbs the effect of $y_{2:T}$ from β_1 and distributes it to all cliques in J_1 .

The overall smoothing algorithm becomes

1. $\alpha_1 = \text{fwd1}(y_1)$
2. For $t = 2 : T$
 $\alpha_t = \text{fwd}(\alpha_{t-1}, y_t)$
3. $\beta_T = 1$
4. For $t = T : 2$
 $\beta_{t-1} = \text{back}(\beta_t, y_t)$
 $J_t = \text{combine}(\alpha_{t-1}, \beta_t, y_t)$
5. $J_1 = \text{combine1}(\alpha_1, \beta_1, y_1)$

We can compute any single node marginal $P(Q_t^{(i)}|y_{1:T})$ or family marginal $P(Q_t^{(i)}, \text{Pa}(Q_t^{(i)})|y_{1:T})$ by marginalizing the appropriate potential in J_t .

As presented above, the $\alpha - \beta$ algorithm must initialize the potentials of J_t three times, once in the forwards operator, once in the backwards operator, and once in the combine operator. This can be optimized by pre-computing $P(Y_t^{(i)}|\text{Pa}(Y_t^{(i)}))$ for each observed node $Y_t^{(i)}$; call the result $O_t^{(i)}$. This is often called the soft evidence (observation

likelihood vector) for the clique containing nodes $\text{Pa}(Y_t^{(i)})$. (For example, in an HMM with mixture of Gaussians output (Section 2.1), we pre-compute $O_t(i, m) = P(y_t|Q_t = i, M_t = m) = \mathcal{N}(y_t; \mu_{i,m}, \Sigma_{i,m})$. For simple DBNs, this is often more expensive than inference itself, because it involves evaluating a large number of Gaussian pdfs.) The O_t terms can then be combined with a generic, pre-initialized J_t , which contains the CPDs for all the hidden nodes in slice 2 of the DBN. This optimization also means that we do not need to store the evidence sequence $y_{1:T}$, which might be large (e.g., if Y_t is an image); instead, we just store $O_{1:T}$.

4.4.3 $\alpha - \beta$ vs $\alpha - \gamma$

There are at least three relevant factors for choosing whether to use $\alpha - \beta$ or $\alpha - \gamma$: space, time and numerical underflow. We discuss each in turn.

The generalized $\alpha - \beta$ algorithm can do exact smoothing in a DBN in $O(TS')$ time and $O(T(S + Y))$ space, where $M^{|I^\rightarrow|+1} \leq S' \leq M^{|I^\rightarrow|+N_h}$ is the max clique size in each 1.5-slice junction tree J_t , $S = M^{|I^\rightarrow|}$ is the size of the belief state, and $Y = \min\{|y_t|, |O_t|\}$ is the size of each observation or observation likelihood vector, whichever is smaller.⁶ The generalized $\alpha - \gamma$ algorithm takes $O(S'T)$ time and space, since it does not need to store the observation stream $y_{1:T}$, but does need to store each J_t . Since $S < S'$ and usually $Y \ll S'$, the $\alpha - \beta$ algorithm is more space-efficient.

The $\alpha - \beta$ algorithm does twice as much work as the $\alpha - \gamma$ algorithm, since it performs two collect/distribute passes within each junction tree: one collect to root on the forwards pass, one distribute from root on the backwards pass, plus a full calibration to combine α_t and β_t . (For an HMM, this corresponds to multiplying by the transition matrix once to compute each α_t , once for each β_t , and once for each $P(Q_{t-1}, Q_t|y_{1:T})$ in the combine operation; the $\alpha - \gamma$ algorithm avoids this last step.) For a DBN with a lot of non-temporal nodes (e.g., Figure 31), the $\alpha - \gamma$ algorithm will be faster (although, as we saw, it uses more space).

Finally, the issue of numerical underflow. $\gamma_t = P(I_t^\rightarrow|y_{1:T})$ is a normalized probability distribution, whereas $\beta_t = P(y_{t+1:T}|I_t^\rightarrow)$ is not. For discrete distributions, this means that γ_t , like α_t , will not underflow, whereas β_t needs to be scaled. For Gaussian distributions, this means we can represent γ_t , like α_t , in moment form, whereas β_t must be represented in canonical (information) form. Moment form has the advantage of being more interpretable and easier to use for approximate inference and deterministic CPDs (Gaussians with zero covariance).

4.5 Space-efficient smoothing

We saw above that we can do exact smoothing with the generalized $\alpha - \beta$ algorithm in $O(TS')$ time and $O(TS)$ space (assuming Y is negligible for simplicity). In general, reducing the dependence on the S/S' term requires approximation methods, such as state-pruning or using factorized belief states (see Section 5.1). However, we can reduce the space requirements to sublinear in T , without sacrificing accuracy, as we discuss below. This space savings is very useful, and often essential, for training large DBNs on long sequences (e.g., for speech recognition or bio-sequence analysis).

If the goal is to compute J_t for all t , then storing the answer takes $O(T)$ space. However, often we want to compute some function of J_1, \dots, J_T , whose result has size independent of T , e.g., for learning, we only need to compute the expected sufficient statistics (ESS) $\sum_{t=1}^T f(J_t)$, for some function f . The log-space smoothing algorithm in Section 4.5.1 allows us to compute such functions by storing only $O(\log T)$ α messages. In Section 4.5.2, we discuss smoothing methods that use $O(1)$ space.

4.5.1 Log-space smoothing

In this section, we show how to reduce the space requirements from $O(ST)$ to $O(Sk \log_k T)$, where k is a tunable parameter we will explain below, using a simple divide-and-conquer algorithm. The catch is that the running time increases from $O(S'T)$ to $O(S'T \log_k T)$. However, if we use $k = \sqrt{T}$, the running time only doubles (since $\log_{\sqrt{T}} T = 2$). The minimal space usage is obtained if we use $k = \lceil e \rceil = \lceil 2.71 \rceil = 3$.

⁶For example, for an HMM with discrete or Gaussian outputs, this is $O(T(Q^2 + Q))$ time and $O(2TQ)$ space, where Q is the number of states, since we must evaluate and store $O_t(i) = P(y_t|Q_t = i)$ and $\alpha_t(i) = P(Q_t = i|y_{1:t})$. (If the transition matrix is sparse, the time requirements drop to $O(2TQ)$.) For an HMM with mixture of Gaussian outputs, this is $O(T(Q^2 + QM))$ time and $O(T(Q + QM))$ space, where M is the number of mixture components, because we must evaluate and store $O_t(i, m) = P(y_t|Q_t = i, M_t = m) = \mathcal{N}(y_t; \mu_{i,m}, \Sigma_{i,m})$ in addition to α_t .

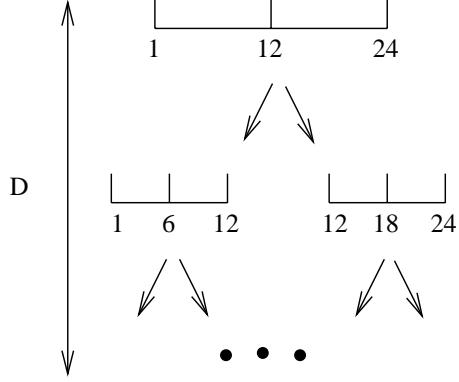


Figure 37: The basic idea behind the Island algorithm, with $T = 24$, $k = 2$. At the top level, we compute α_t and β_t for $t = 1, 12, 24$. We then call the algorithm recursively on each sub-sequence, using the boundary conditions (α_1, β_{12}) and $(\alpha_{12}, \beta_{24})$. D is the depth of the recursion. In fact, it is only necessary to store the α 's, since we can update β in place by working right to left, as shown in Figure 39.

The key idea behind the algorithm is that instead of storing $\alpha_1, \dots, \alpha_T$, we only store some of the α messages at certain positions called checkpoints or “islands”. We can regenerate $\alpha_t, \dots, \alpha_{t+\tau}$ provided we store the left hand boundary condition α_{t-1} .⁷ We can then do a backwards pass in constant space, and combine each β_t with each α_t . See Figure 37 for a sketch of the basic idea.

In Figure 38, we give pseudo-code for the algorithm, and in Figure 39, we give a detailed example. The algorithm “emits” the J_t 's to some “consumer” process (which e.g., sums them up in order to compute the expected sufficient statistics needed for learning) in a non-sequential order.

The running time of the algorithm on a sequence of length T , $R(T)$, satisfies the following recurrence:

$$R(T) = T + kR\left(\frac{T}{k}\right)$$

since we first do a forwards sweep, which takes $O(T)$ time, and then we divide the sequence into k sub-sequences of length T/k , calling the algorithm recursively on each such piece. Solving this recurrence gives $R(T) = O(T \log k T)$.

The total amount of space is bounded by the depth of the recursion, $D \leq \log_k T$, times the amount of space needed at each recursive invocation, which is $O(kS)$. (This is because we must store checkpoints at k positions at every level on the call stack.) So the total space required is $O(Sk \log_k T)$. Another way to control the amount of space we use, in addition to k , is to vary the depth at which we “bottom out”, i.e., switch over to the linear space algorithm. If this occurs for sequences of length T_{\min} , then the space usage is $O(ST_{\min} + Sk(\lfloor \log_k T \rfloor - 1))$: see Figure 39 for an example.

4.5.2 Constant-space smoothing

Can we go further, and use an amount of space independent of T ? Here is a trivial algorithm to do this: for each $1 \leq k \leq T$, do the following: first run a forwards filter from $t = 1 : k$, to compute α_k in constant space, then run a backwards filter from $t = T : k$, to compute β_k in constant space, and then combine the results. Unfortunately this takes $O(T)$ for each k , and hence $O(T^2)$ time overall, rendering it useless in practice.

If we could invert the forward operator (i.e., compute $P(X_t | y_{1:t})$ from $P(X_{t+1} | y_{1:t+1})$), we could do smoothing in $O(1)$ space and $O(T)$ time as follows. First run the forwards algorithm, $\alpha_t = \text{Fwd}(\alpha_{t-1}, y_t)$, forgetting all the intermediate results (i.e., only keeping α_T), and then run the backwards algorithm, computing $\alpha_t = \text{Fwd}^{-1}(\alpha_{t+1}, y_{t+1})$ and $\beta_t = \text{Back}(\beta_{t+1}, y_{t+1})$ in parallel, combining them at each step. We call this the “country dance” algorithm, because you can think of the forward pass as running to the end to pick up its “partner”, and the two of them moving

⁷To be able to regenerate the α 's, we must also store the evidence stream in the form of $y_{1:T}$ or $O_{1:T}$, and be able to access it in a random order. Hence the total storage is $O(Sk \log_k T + YT)$. Although we can make a $\alpha - \gamma$ version of the algorithm, doing so requires storing the J 's as well as the α_t 's, which is a bad idea if we want to save space.

```

function Island( $l, \alpha, r, \beta$ ) // Returns  $\beta_l$ , emits  $J_{l+1}, \dots, J_r$ 
if  $l - r < T_{\min}$ 
    Allocate space for  $\alpha[1 : l - r + 1]$ 
     $i = 1$ 
     $\alpha[i] = \alpha$ 
    for  $t = l + 1 : r$ 
         $\alpha[i] = \text{fwd}(\alpha[i - 1], y[t])$ 
         $i = i + 1$ 
    for  $t = r : l + 1$ 
         $J = \text{combine}(\alpha[i - 1], \beta, y[t])$ 
         $i = i - 1$ 
        emit  $J$ 
         $\beta = \text{back}(\beta, y[t])$ 
    return  $\beta$ 
else
    Allocate space for  $\alpha[1 : k]$ 
     $s = \lfloor T/k \rfloor$ 
    left =  $[l, l + s, l + 2s, \dots, l + (k - 1)s]$ 
    right =  $[l + s, l + 2s, \dots, l + (k - 1)s, r]$ 
     $i = 1$ 
    for  $t = l : \max(left)$ 
        if  $t == \text{left}[i]$ 
            store  $\alpha[i] = \alpha$ 
             $i = i + 1$ 
             $\alpha = \text{fwd}(\alpha, y[t])$ 
    for  $i = k : 1$ 
         $\beta = \text{Island}(\text{left}[i], \alpha[i], \text{right}[i], \beta)$ 
    return  $\beta$ 

function smooth( $y_{1:T}$ )
 $\alpha_1 = \text{fwd1}(y_1)$ 
 $\beta_1 = \text{Island}(1, \alpha_1, T, 1)$ 
 $J_1 = \text{combine1}(\alpha_1, \beta_1, y_1)$ 
emit  $J_1$ 

```

Figure 38: Pseudo-code for the Island algorithm (log-space smoothing), $\alpha - \beta$ version.

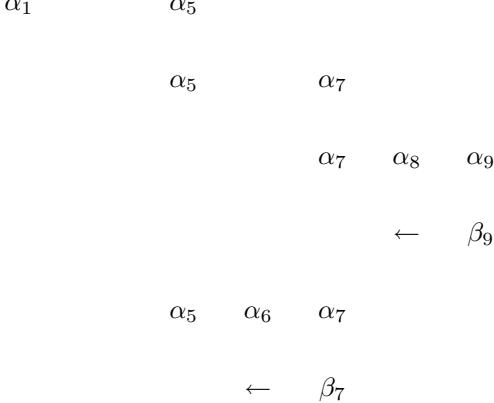


Figure 39: Example of the log-space smoothing algorithm in Figure 38 when $k = 2, T = 9, T_{\min} = 3$. We divide each sequence into $k = 2$ regions, storing α at k checkpoints. We then call the function recursively on each subsequence. For example, we start by calling $\text{Island}(\alpha = \alpha_1, l = 1, \beta = \beta_9 = 1, r = 9)$, call this invocation 1, which stores $\alpha[1] = \alpha_1, \alpha[2] = \alpha_5$. This then calls $\text{Island}(\alpha = \alpha[2] = \alpha_5, l = 5, \beta = \beta_9, r = 9)$, call this invocation 2, which stores $\alpha[1] = \alpha_5, \alpha[2] = \alpha_7$. This then calls $\text{Island}(\alpha = \alpha[2] = \alpha_7, l = 7, \beta = \beta_9, r = 9)$, call it invocation 3, which stores $\alpha[1] = \alpha_7, \alpha[2] = \alpha_8, \alpha[3] = \alpha_9$, and then, in the backwards pass, emits $J_9 = \text{combine}(\alpha_8, \beta_9, y_9), J_8 = \text{combine}(\alpha_7, \beta_8, y_8)$, and returns β_7 . Invocation 3 then calls $\text{Island}(\alpha = \alpha[1] = \alpha_5, l = 5, \beta = \beta_7, r = 7)$, etc. The total number of α messages stored is $k(\lfloor \log_k T \rfloor - 1) + T_{\min} = 2(\lfloor \log_2 9 \rfloor - 1) + 3 = 2(3 - 1) + 3 = 7$.

together back to the start. (If the backwards operator can be inverted, the backwards pass can run to the start to pick up the forwards partner.)

Unfortunately, inverting the forward operator is impossible in general: to compute $P(X_t|y_{1:t})$ from $P(X_{t+1}|y_{1:t+1})$ requires figuring out how to “undo” the effect of the transition and the observation y_{t+1} . This is most easily seen from the forwards equation for HMMs:

$$\alpha_{t+1} \propto O_{t+1} A^T \alpha_t$$

where $O_t(i, i) = P(y_t|Q_t = i)$ is a diagonal matrix containing the observation likelihoods at time t . Inverting this yields

$$\alpha_t \propto (A^T)^{-1} O_{t+1}^{-1} \alpha_{t+1}$$

But in general A will not be invertible (think of a Markov chain in which there is more than one way of reaching a state). Also, if any observations are impossible in a given state, O_t will not be invertible either. (Even if A and O_t were invertible, how would we invert the forwards operator when it is implemented via the junction tree algorithm?) A similar argument shows that the backward operator cannot be inverted in general. Hence it seems impossible to do smoothing in constant space.

Perhaps this is not surprising, since invertibility would also mean we could do that fixed-lag smoothing in constant time per update (independent of the length of the lag)⁸, which seems to violate our intuition that instantaneous “action at a distance” should be impossible.⁹

4.6 Conditionally tractable substructure

The $O(M^{|I^\rightarrow|})$ complexity of filtering is a purely graph-theoretic (worst-case) result. It sometimes happens that the CPDs encode conditional independencies that are not evident in the graph structure. This can lead to significant speedups. For example, consider Figure 40. This is a schematic for two “processes” (here represented by single nodes), B and C , both of which only interact with each other via an intermediate layer, R .

⁸We leave the proof of this fact as an exercise. The tricky part is to compute $\beta_{-l+2|t+1}$ from $\beta_{t-l+1|t}$ in constant time, as required when the window slides right by one step. Hint: Express $\beta_{t-l+1|t}$ as a product of matrices times $\beta_{t+1|t} = 1$; similarly express $\beta_{t-l+2|t+1}$ as a product of matrices times $\beta_{t+2|t+1} = 1$; then find a relationship between them.

⁹Although filtering in a hierarchical model may have a similar effect to fixed lag smoothing, since high level nodes span larger time scales, it cannot provide smoothed estimates for all elements within a window in constant time.

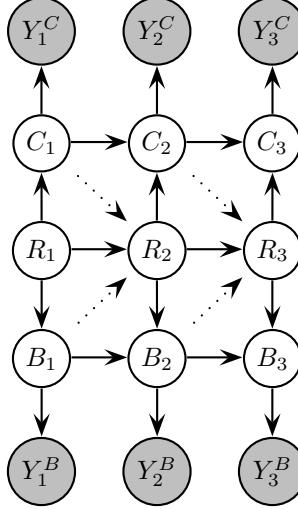


Figure 40: Two “processes”, here represented by single nodes, B and C , only interact with each other via an “interface” layer, R .

The outgoing interface is $\{R, B, C\}$; since B and C represent whole subprocesses, this might be quite large. Now suppose we remove the dotted arcs, so R becomes a “root”, with only outgoing arcs to itself and B and C . Again, the interface is $\{R, B, C\}$. Finally, suppose that R is in fact a static node, i.e., $P(R_t|R_{t-1}) = \delta(R_t, R_{t-1})$. (For example, R might be a fixed parameter.) In this case, the model can be simplified as shown in Figure 41. This model enjoys the property that, conditioned on R , the interface factorizes:

$$P(R, B_t, C_t | y_{1:t}) = P(B_t | R, y_{1:t}) P(C_t | R, y_{1:t}) P(R | y_{1:t}) = P(B_t | R, y_{1:t}^B) P(C_t | R, y_{1:t}^C) P(R | y_{1:t})$$

where $y_t = (y_t^B, y_t^C)$. This follows since $B_t \perp y_{1:t}^C | R$ and $C_t \perp y_{1:t}^B | R$, i.e., evidence which is local to a process does not influence other processes: the R node acts like a barrier. Hence we can recursively update each subprocess separately:

$$\begin{aligned} P(B_t | R, y_{1:t}^B) &= P(B_t | R, y_{1:t-1}^B, y_t^B) \\ &\propto P(y_t^B | B_t) \sum_b P(B_t | B_{t-1} = b, R) P(B_{t-1} = b | R, y_{1:t-1}^B) \end{aligned}$$

The distribution over the root variable can be computed at any time using

$$P(R | y_{1:t}) \propto \sum_b P(R) P(B_t = b | R, y_{1:t})$$

The reason we cannot apply the same factoring trick to the model in Figure 40 is that $P(B_t | R_t, y_{1:t}) \neq P(B_t | R_t, y_{1:t}^B)$, since there is a path (e.g., via R_{t-1}) connecting Y_t^C to B_t . If we could condition on the whole chain $R_{1:t}$ instead of just on R_t , we could factor the problem. Of course, we cannot condition on all possible values of $R_{1:t}$, since there are M^t of them; however, we can *sample* representative instantiations. Given these samples, we can update the processes exactly. This is an example of Rao-Blackwellised particle filtering: see Section 5.4.3.

5 Approximate filtering

So far, we have discussed how to perform exact filtering in discrete-state models. Although this is always possible, it might be too slow, because the belief state, $P(X_t | y_{1:t})$ (or, more accurately, $P(I_t^\rightarrow | y_{1:t})$) is too large. Hence we must resort to approximations.

What about models with continuous, or mixed discrete-continuous, hidden state? It turns out that for nearly all such models, exact (i.e., closed-form, fixed-sized) representations of the belief state do not exist. The exception is models in which the transition and observation models are conjugate. Linear-Gaussian (state-space) models have this

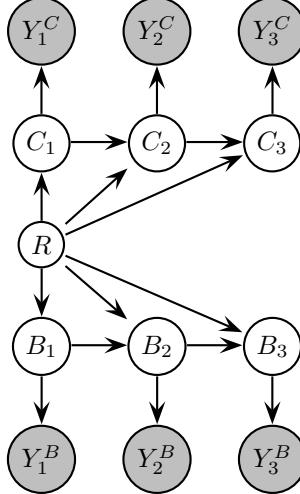


Figure 41: Conditioned on the static root, the interface is fully factored.

property: the belief state, $P(X_t|y_{1:t})$, can always be exactly represented by a Gaussian.¹⁰ However, in general, we must approximate the belief state. In the following sections, we review a number of different algorithms, which use different representations for the approximate belief state. Most of these methods are only suitable for restricted classes of DBNs. At present, discretization and sampling (Section 5.4) are the only methods that can, in theory, handle DBNs with arbitrary topology and arbitrary CPDs. Unfortunately, discretization is not feasible for large networks/nodes, and to make sampling work well in practice, it is usually necessary to make additional restrictions/ assumptions and to use model-specific heuristics. Hence the picture is somewhat less satisfying than in Section 4.

5.1 Belief state = discrete distribution

In this section, we consider problems where the belief state, $P(X_t|y_{1:t})$, is a discrete probability distribution, as in a histogram. This might arise if we discretized all the continuous hidden variables, or if all the variables were all discrete to begin with. Although in principle we can use the exact methods discussed in Section 4, in practice they might be too slow. Hence we develop faster, but approximate, algorithms.

5.1.1 Assumed density filtering (ADF)/ BK algorithm

Basic idea Assumed density filtering (ADF) assumes the belief state is a member of some restricted family \mathcal{F} . Figure 42 illustrates the basic idea. We start with a prior, $\tilde{\alpha}_{t-1} \in \mathcal{F}$, and perform one step of exact Bayesian updating to get $\hat{\alpha}_t$. Usually $\hat{\alpha}_t \notin \mathcal{F}$, so we approximate it by choosing the “closest” approximation in the family: $\tilde{\alpha}_t = \arg \min_{q \in \mathcal{F}} D(\hat{\alpha}_t || q)$. If \mathcal{F} is in the exponential family, this equation can be solved by moment matching.

For a DBN, it is natural to let \mathcal{F} be the set of factorized distributions, i.e., we approximate the joint distribution α_t by a product of marginals over clusters of variables:

$$\alpha_t \approx \tilde{\alpha}_t = \prod_{i=1}^C P(Q_t^{(c_i)} | y_{1:t})$$

where $\cup_i c_i = I^\rightarrow$ is a partition of the interface into clusters (which may overlap). This is often called the “Boyen-Koller” (BK) approximation. The best approximate distribution in this family can be found by moment matching, which in the discrete case means computing the marginals on each cluster. We discuss how to do this efficiently below.

¹⁰To see this, note that since the previous belief state, $P(X_{t-1}|y_{1:t-1})$, is Gaussian (by assumption), then so is $P(X_t|y_{1:t-1})$, since X_t is a linear transformation of X_{t-1} plus additive Gaussian noise. Combining a Gaussian prior, $P(X_t|y_{1:t-1})$, with a Gaussian likelihood, $P(y_t|X_t)$, yields a Gaussian posterior, $P(X_t|y_{1:t})$. In other words, linear-Gaussian models are closed under Bayesian updating.

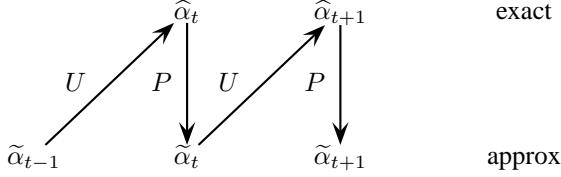


Figure 42: The ADF algorithm as a series of update (U) and projection (P) steps.

Error analysis It can be shown that the error remains bounded over time, in the following sense:

$$E D(\alpha_t || \tilde{\alpha}_t) \leq \frac{\epsilon_t}{\gamma^*}$$

where the expectation is taken over the possible observation sequences, γ^* is the mixing rate of the DBN, and ϵ_t is the additional error incurred by projection, over and above any error inherited from the factored prior:

$$\epsilon_t = D(\alpha_t || \tilde{\alpha}_t) - D(\alpha_t || \hat{\alpha}_t).$$

The intuition is that, even though projection introduces an error at every time step, the stochastic nature of the transitions, and the informative nature of the observations, reduces the error sufficiently to stop it building up.

The accuracy of the BK algorithm depends on the clusters that we use to approximate the belief state. Exact inference corresponds to using a single cluster, containing all the interface nodes. The most aggressive approximation corresponds to using one cluster per variable (a “fully factorized” approximation).

The best case for BK is when the observations are perfect (noiseless), or when the transition matrix is maximally stochastic (i.e., $P(X_t | X_{t-1})$ is independent of X_{t-1}), since in either case, errors in the prior are irrelevant. Conversely, the worst case for BK is when the observations are absent or uninformative, and the transition matrix is deterministic, since in this case, the error grows over time. (This is consistent with the theorem, since deterministic processes can have infinite mixing time.) In Section 5.4, we will see that the best case for BK is the worst case for particle filtering (when we propose from the transition prior), and vice versa.

Implementation A naive implementation of BK uses the HMM filter to perform the belief state update, and then marginalizes the resulting posterior onto each cluster. Of course, this would be no faster than exact inference (in fact, it is slower, because of the extra marginalization step); however, it does allow one to empirically assess the accuracy of the method.

We can implement the BK algorithm much more efficiently by slightly modifying the interface algorithm (Section 4.4). Specifically, we construct the junction tree for the 1.5DBN as before, but instead of adding undirected edges between all nodes in I_{t-1}^- and I_t^+ , we only add them between all nodes in each cluster. The assumption is that this will result in smaller cliques. Figure 44 shows that the max clique size is indeed reduced in some cases.

In addition, we need to slightly modify the forwards operator, as follows.

1. Initialize $\phi(J_t)$ using the CPDs from slice 2 and the evidence y_t , as before.
2. Instead of computing $\phi_{in} = \phi_{in} * \alpha_{t-1}$, for each factor in the prior, we must find a clique c that contains it, and multiply $\phi_c = \phi_c * \alpha_{t-1}^c$.
3. Instead of just collecting to the out-clique, we do a full calibration of the tree.¹¹
4. Instead of computing $\alpha_t = \phi_{out} \downarrow I_t^+$, for each factor, we must find a clique c that contains it, and set $\alpha_t^c = \phi_c \downarrow I_t^c$.

¹¹The reason we must collect *and distribute* evidence is because the clusters which constitute the interface may be in several cliques, and hence all of them must be updated before being used as the prior for the next step. If there is only one cluster, it suffices to collect to the clique that contains this cluster.

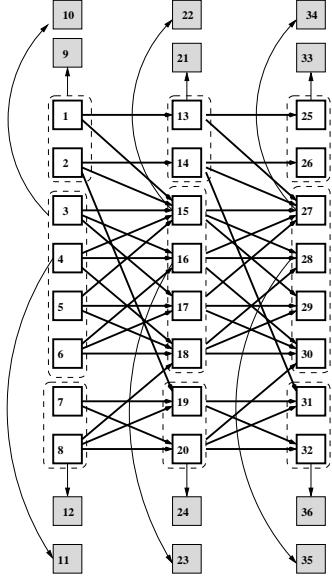


Figure 43: The water DBN, designed to monitor a waste water treatment plant. The dotted arcs group together nodes that will be used in the BK approximation (Section 5.1.1). This model is originally from [JKOP89], and was modified by [BK98b] to include evidence nodes.

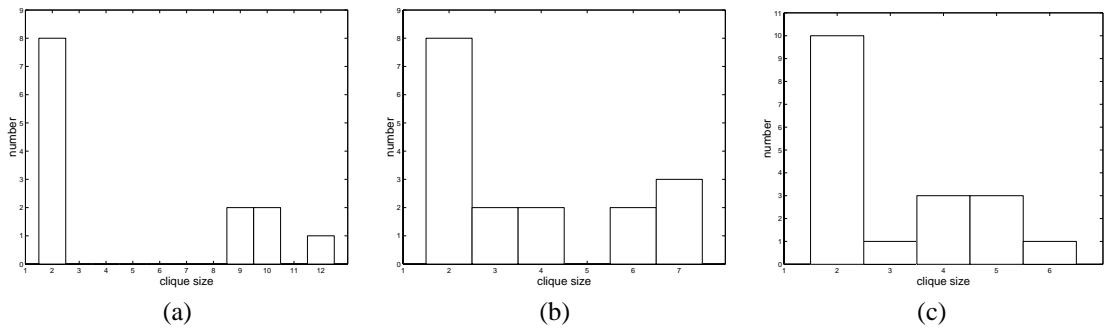


Figure 44: Clique size distribution for different levels of BK approximation applied to the water DBN (Figure 43). The cliques of size 2 contain observed leaves and their parents. (a) Exact: the interface is $\{1, \dots, 8\}$. (b) Partial approximation: the interface sets are $\{1, 2\}$, $\{3, 4, 5, 6\}$ and $\{7, 8\}$, as in Figure 43. (c) Fully factorized approximation: the interface sets are the singletons.

5.1.2 Beam search

A completely different kind of approximation is to assume that there are only a small number of likely hypotheses. From each such likely prior state, we try to find the next most probable set of states. We can either keep the k most likely states, or keep enough states to ensure we cover enough of the probability mass. This method is widely used in speech recognition, where we only keep track of the most probable interpretations of the sentence, and also in fault diagnosis, where we only keep track of the most probable faults.

5.2 Belief state = Gaussian

In this section, we consider algorithms that approximate $P(X_t|y_{1:t})$ by a single Gaussian. (Many of these could be generalized to use any member of the exponential family.)

5.2.1 Kalman filter (KF)

Recall the form of the state-space model from Chapter ??:

$$\begin{aligned} x_t &= Ax_{t-1} + v_t \\ y_t &= Cx_t + w_t \end{aligned}$$

where $v_t \sim \mathcal{N}(0, Q)$ is the process noise, $w_t \sim \mathcal{N}(0, R)$ is the observation noise, and the variables satisfy the conditional independence assumptions implicit in Figure 1. (In this and following sections we adopt the convention that capital letters denote matrices, and lower case letters denote random variables or values of random variables. To simplify notation, we omit the input variable u_t .) In this case, the belief state is Gaussian and can be represented exactly by its first two moments: $\hat{x}_{t|t} \stackrel{\text{def}}{=} E[x_t|y_{1:t}]$ and $P_{t|t} \stackrel{\text{def}}{=} E[(x_t - \hat{x}_t)(x_t - \hat{x}_t)^T|y_{1:t}]$. This belief state can be updated recursively using the Kalman filter, as we saw in Chapter ???. For completeness, we restate the equations here. First we perform the time update (prediction step):

$$\begin{aligned} x_{t|t-1} &= Ax_{t-1|t-1} \\ P_{t|t-1} &= AP_{t-1|t-1}A^T + Q \\ y_{t|t-1} &= Cx_{t|t-1} \end{aligned}$$

Then we perform the measurement update (correction step):

$$\begin{aligned} \tilde{y}_t &= y_t - \hat{y}_{t|t-1} \quad (\text{error}) \\ P_{\tilde{y}_t} &= CP_{t|t-1}C^T + R \quad (\text{covariance of error}) \\ P_{x_t y_t} &= P_{t|t}C^T \quad (\text{cross covariance}) \\ K_t &= P_{x_t y_t}P_{\tilde{y}_t}^{-1} \quad (\text{Kalman gain matrix}) \\ x_{t|t} &= x_{t|t-1} + K_t(y_t - \hat{y}_{t|t-1}) \\ P_{t|t} &= P_{t|t-1} - K_t P_{\tilde{y}_t} K_t^T \end{aligned}$$

We can generalize the Kalman filter equations to apply to any linear-Gaussian DBN by using the junction tree algorithm discussed in Section 4, and modifying the definition of summation, multiplication and division so that these operations can be applied to Gaussian potentials instead of discrete ones.

5.2.2 Extended Kalman filter (EKF)

The extended Kalman filter (EKF) can be applied to models of the form

$$\begin{aligned} x_t &= f(x_{t-1}) + v_t \\ y_t &= g(x_t) + w_t \end{aligned}$$

where f and g are arbitrary, differentiable functions. The basic idea is to linearize f and g about the previous state estimate using a second order Taylor expansion, and then to apply the standard Kalman filter equations. (The noise

variance in the equations (Q and R) is not changed, i.e., the additional error due to linearization is not modeled.) Thus we approximate the stationary nonlinear dynamical system with a non-stationary linear dynamical system. That is, at time t , we approximate the model by

$$\begin{aligned} x_t &= f(\hat{x}_{t-1|t-1}) + A_{\hat{x}_{t|t-1}}(x_{t-1} - \hat{x}_{t-1|t-1}) + v_t \\ y_t &= g(\hat{x}_{t|t-1}) + C_{\hat{x}_{t|t-1}}(x_t - \hat{x}_{t|t-1}) + w_t \end{aligned}$$

where $\hat{x}_{t|t-1} = f(\hat{x}_{t-1|t-1})$ and

$$\begin{aligned} A_{\hat{x}} &\stackrel{\text{def}}{=} \left. \frac{\partial f}{\partial x} \right|_{\hat{x}} \\ C_{\hat{x}} &\stackrel{\text{def}}{=} \left. \frac{\partial g}{\partial x} \right|_{\hat{x}} \end{aligned}$$

We then use these state-dependent matrices in the standard KF equations.

It is possible to improve performance by repeatedly re-linearizing the equations around $\hat{x}_{t|t}$ instead of $\hat{x}_{t|t-1}$; this is called the iterated EKF, and yields better results, especially in the case of highly nonlinear measurement models.

5.2.3 Unscented Kalman filter (UKF)

The unscented Kalman filter (UKF) propagates a deterministically chosen set of points through the non-linear functions f and g , and fits a Gaussian to the resulting transformed points. (This is called an unscented transform, and is closely related to Gaussian quadrature methods for the numerical evaluation of integrals.) The resulting Gaussian approximation is accurate to at least second order, whereas the EKF is only a first order approximation. Furthermore, the UKF does not require the analytic evaluation of any derivatives, making it simpler and more widely applicable than the EKF. In general, both algorithms perform $O(d^3)$ operations per step (where $X_t \in \mathbb{R}^d$).

Before explaining the UKF, we first explain the unscented transform. Assume $P(x) = \mathcal{N}(x; \hat{x}, P_x)$, and consider estimating $P(y)$, where $y = f(x)$ for some nonlinear function f . The unscented transform does this as follows. First we create a matrix X of $2d+1$ sigma vectors x_i , given by

$$\begin{aligned} x_0 &= \hat{x} \\ x_i &= \hat{x} + (\sqrt{(d+\lambda)P_x})_i, \quad i = 1, \dots, d \\ x_i &= \hat{x} - (\sqrt{(d+\lambda)P_x})_i, \quad i = d+1, \dots, 2d \end{aligned}$$

where $\lambda = \alpha^2(d+\kappa) - d$ is a scaling parameter. (In general, the optimal values of α , β and κ are problem dependent, but when $d=1$, they are $\alpha=1$, $\beta=0$, $\kappa=2$.) $(\sqrt{(d+\lambda)P_x})_i$ is the i 'th column of the matrix square root. (These points are just ± 1 standard deviation around the mean.) These sigma vectors are propagated through the nonlinear function to yield $y_i = f(x_i)$, and the mean and covariance for y are computed as follows:

$$\begin{aligned} \hat{y} &= \sum_{i=0}^d W_i^{(m)} y_i \\ P_y &= \sum_{i=0}^d W_i^{(c)} (y_i - \hat{y})(y_i - \hat{y})^T \end{aligned}$$

where

$$\begin{aligned} W_0^{(m)} &= \lambda/(d+\lambda) \\ W_0^{(c)} &= \lambda/(d+\lambda) + (1-\alpha^2 + \beta) \\ W_i^{(m)} &= W_i^{(c)} = 1/(2(d+\lambda)) \end{aligned}$$

See Figure 45 for an example.

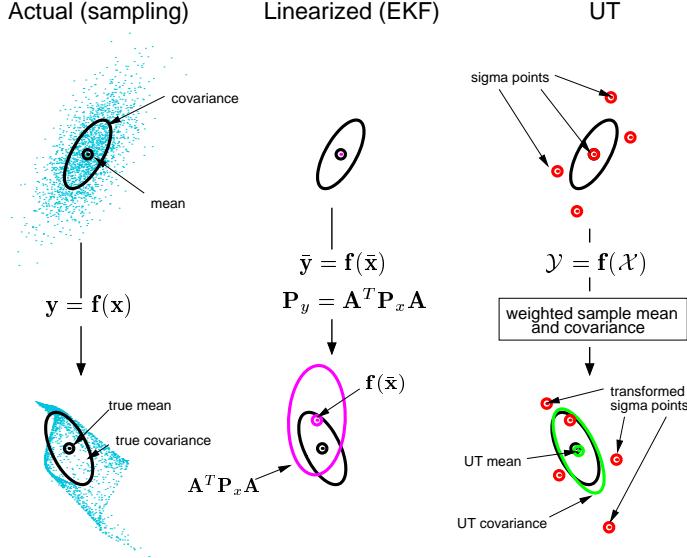


Figure 45: An example of the unscented transform in two dimensions. Thanks to Rudolph van der Merwe for this Figure.

The UKF algorithm is simply two applications of the unscented transform, one to compute $P(X_t|y_{1:t-1})$ and the other to compute $P(X_t|y_{1:t})$. In the case of zero mean additive Gaussian noise, the algorithm is as follows. First we create the vector

$$X_{t-1} = \left[\hat{x}_{t-1|t-1}, \hat{x}_{t-1|t-1} + \gamma \sqrt{P_{t-1|t-1}}, \hat{x}_{t-1|t-1} - \gamma \sqrt{P_{t-1|t-1}} \right]$$

where $\gamma = \sqrt{d + \lambda}$. The time update becomes:

$$\begin{aligned} X_{t|t-1} &= f(X_{t-1}) \\ \hat{x}_{t|t-1} &= \sum_{i=0}^{2d} W_i^{(m)} X_{i,t|t-1} \\ \hat{P}_{t|t-1} &= \sum_{i=0}^{2d} W_i^{(c)} (X_{i,t|t-1} - \hat{x}_{t|t-1})(X_{i,t|t-1} - \hat{x}_{t|t-1})^T + Q \\ \hat{Y}_{t|t-1} &= g(X_{t|t-1}) \\ \hat{y}_{t|t-1} &= \sum_{i=0}^{2d} W_i^{(m)} Y_{i,t|t-1} \end{aligned}$$

The measurement update becomes

$$\begin{aligned} \hat{P}_{\tilde{y}_t} &= \sum_{i=0}^{2d} W_i^{(c)} (Y_{i,t|t-1} - \hat{y}_{t|t-1})(Y_{i,t|t-1} - \hat{y}_{t|t-1})^T + R \\ \hat{P}_{x_t, y_t} &= \sum_{i=0}^{2d} W_i^{(c)} (X_{i,t|t-1} - \hat{x}_{t|t-1})(Y_{i,t|t-1} - \hat{y}_{t|t-1})^T \\ K_t &= P_{x_t, y_t} \hat{P}_{\tilde{y}_t}^{-1} \\ \hat{x}_{t|t} &= \hat{x}_{t|t-1} + K_t (y_t - \hat{y}_{t|t-1}) \\ P_{t|t} &= P_{t|t-1} - K_t \hat{P}_{\tilde{y}_t} K_t^T \end{aligned}$$

We can see that the UKF is a simple modification to the standard KF, but which can handle nonlinearities with greater accuracy than the EKF, and without the need to compute derivatives.

5.2.4 Assumed density filter (ADF)/ moment matching filter

We already encountered ADF in Section 5.1.1, where we assumed the belief state was representable as a product of discrete marginals. In this section, we assume the belief state is represented by a Gaussian. The goal is to find the parameters of this Gaussian.

For example, consider a system with linear-Gaussian dynamics but non-linear, non-Gaussian observations (e.g., a mixture of Gaussians, which could not be handled by the EKF or UKF). We start with an approximate Gaussian prior:

$$P(x_{t-1}|y_{1:t-1}) \approx \tilde{\alpha}(x_{t-1|t-1}) = \mathcal{N}(x_{t-1}; \tilde{x}_{t-1|t-1}, \tilde{P}_{t-1|t-1}).$$

This is propagated through the linear-Gaussian dynamics to get a Gaussian one-step-ahead prediction density:

$$P(x_t|y_{1:t-1}) \approx \tilde{\alpha}(x_{t|t-1}) = \mathcal{N}(x_t; A\tilde{x}_{t-1|t-1}, A\tilde{P}_{t-1|t-1} + Q).$$

The exact update step becomes

$$\begin{aligned} P(x_t|y_{1:t}) &= \frac{P(y_t|x_t)P(x_t|y_{1:t-1})}{\int_{x_t} P(y_t|x_t)P(x_t|y_{1:t-1})} \\ &\approx \frac{P(y_t|x_t)\tilde{\alpha}(x_{t|t-1})}{\int_{x_t} P(y_t|x_t)\tilde{\alpha}(x_{t|t-1})} \\ &\stackrel{\text{def}}{=} \hat{\alpha}(x_{t|t}) \end{aligned}$$

The goal is to compute the closest Gaussian approximation to this posterior. To simplify notation, let us rewrite the above as

$$\hat{\alpha}(x_{t|t}) = \hat{p}(\theta) = \frac{l(\theta)q^{old}(\theta)}{\int_{\theta} l(\theta)q^{old}(\theta)}$$

where $\theta = x_t$, $l(\theta) = P(y_t|x_t)$ is the likelihood, and $q^{old}(\theta) = \tilde{\alpha}(x_{t|t-1})$ is the Gaussian prior. The goal is to minimize $D(\hat{p}(\theta)||q^{new}(\theta))$ subject to the constraint that $q^{new}(\theta) = \mathcal{N}(\theta; m_{\theta}^{new}, V_{\theta}^{new})$ is Gaussian, where $m_{\theta}^{new} = E_{q^{new}}[\theta]$ and $V_{\theta}^{new} = E_{q^{new}}[\theta\theta^T] - E_{q^{new}}[\theta]E_{q^{new}}[\theta]^T$. Taking derivatives wrt m_{θ}^{new} and V_{θ}^{new} gives the expectation constraints

$$\begin{aligned} E_{q^{new}}[\theta] &= E_{\hat{p}}[\theta] = \int_{\theta} \hat{p}(\theta)\theta \\ E_{q^{new}}[\theta\theta^T] &= E_{\hat{p}}[\theta\theta^T] = \int_{\theta} \hat{p}(\theta)\theta\theta^T \end{aligned}$$

We can compute the expectations wrt \hat{p} using the following relations (obtained from integration by parts):

$$\begin{aligned} Z(m_{\theta}^{old}, V_{\theta}^{old}) &= \int_{\theta} l(\theta)q^{old}(\theta) \\ d_m &= \nabla_{m_{\theta}^{old}} \log Z(m_{\theta}^{old}, V_{\theta}^{old}) \\ d_V &= \nabla_{V_{\theta}^{old}} \log Z(m_{\theta}^{old}, V_{\theta}^{old}) \\ E_{\hat{p}}[\theta] &= m_{\theta}^{old} + V_{\theta}^{old}d_m \\ E_{\hat{p}}[\theta\theta^T] - E_{\hat{p}}[\theta]E_{\hat{p}}[\theta]^T &= V_{\theta}^{old} - V_{\theta}^{old} (d_m d_m^T - 2d_V) V_{\theta}^{old} \end{aligned}$$

These equations hold for any likelihood term $l(\theta)$, which could be e.g., a mixture of Gaussians. In general, solving the integral in the definition of Z might require further approximations.

5.3 Belief state = mixture of Gaussians

A natural extension is to allow the belief state to be represented by a mixture of Gaussians. The goal is to take the old belief state, $P(X_{t-1}|y_{1:t-1})$, represented by a mixture of M Gaussians, to pass it through an arbitrary transition model to get the prior $P(X_t|y_{1:t-1})$, to do Bayesian updating with this prior and an arbitrary observation model, $P(Y_t|X_t)$, and then to approximate the result with another mixture of Gaussians. Unfortunately, an efficient, general algorithm to do this has not yet been found.

An important special case for which this problem can be (approximately) solved is filtering in a switching SSM (Section 2.12). The old belief, $P(X_{t-1}, S_{t-1}|y_{1:t-1})$, is naturally represented as a mixture of M Gaussians, one for each value of the switch, S_{t-1} . For each value of the new switch, $S_t = j$, we have a linear dynamical system with parameters θ_j . We apply the standard Kalman filter update to each mode i of the prior, using parameters θ_j . Hence the posterior $P(X_t, S_t = j|y_{1:t}, S_{t-1} = i)$ is a mixture of M^2 Gaussians. The posterior needs to be reduced back to a mixture of M Gaussians, otherwise the size of the belief state will grow exponentially with time. There are a variety of heuristic ways to do this:

- Drop mixture components of low weight.
- Sample mixture components according to their weight.
- Repeatedly merge the most similar pair of mixture components.
- Minimize the divergence between the M^2 -component and the M -component mixture using nonlinear optimization.
- For each value of $S_t = j$, use moment matching (Section 5.2.4) to approximate the mixture $\sum_i P(X_t, S_t = j|S_{t-1} = i, y_{1:t})$ by a single Gaussian. This is sometimes called the second-order generalized pseudo Bayesian (GPB2) algorithm, and we explain it in more detail in Section 5.3.1 below.

5.3.1 GPB2 algorithm

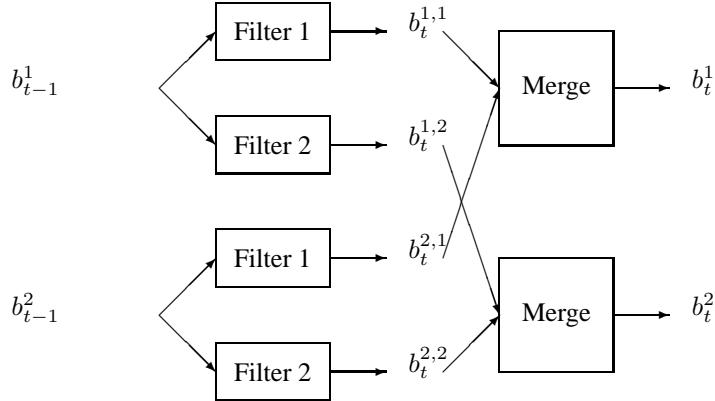


Figure 46: The GPB2 algorithm for the case of a binary switch. $b_{t-1}^i = P(X_{t-1}, S_{t-1} = i|y_{1:t-1})$ is the prior, $b_t^{i,j} = P(X_t, S_{t-1} = i, S_t = j|y_{1:t})$, is the joint posterior, and $b_t^j = P(X_t, S_t = j|y_{1:t})$ is the marginal posterior. The filter box implements the Kalman filter equations. The merge box implements the moment matching equations.

The basic idea behind GPB2 is shown in Figure 46. For each mode of the prior, $P(X_{t-1}, S_{t-1} = i|y_{1:t-1})$, and for each value of the current switch $S_t = j$, we compute $P(X_t, S_t = j, S_{t-1} = i|y_{1:t})$ using a Kalman filter with parameters θ_j . We then collapse the different S_{t-1} components of $P(X_t, S_t = j, S_{t-1} = i|y_{1:t})$ for each j by moment matching. Specifically, if

$$P(X_t, S_t = j, S_{t-1} = i|y_{1:t}) = p_{ij}\mathcal{N}(X_t; \mu_{ij}, \Sigma_{ij})$$

then the new distribution is given by

$$P(X_t, S_t = j | y_{1:t}) = p_j \mathcal{N}(X_t; \mu_j, \Sigma_j)$$

where

$$\begin{aligned} p_j &= \sum_i p_{ij} \\ p_{j|i} &= \frac{p_{ij}}{\sum_j p_{ij}} \\ \mu_j &= \sum_i \mu_{ij} p_{j|i} \\ \Sigma_j &= \sum_i \Sigma_{ij} p_{j|i} + \sum_i (\mu_{ij} - \mu_j) (\mu_{ij} - \mu_j)^T p_{j|i} \end{aligned}$$

In the junction tree literature, this is called “weak marginalization”. It can be applied to any conditionally Gaussian model, not just switching SSMs.

The GPB2 algorithm requires running M^2 Kalman filters at each step. A cheaper alternative, known as interacting multiple models (IMM), can be obtained by first collapsing the prior to a single Gaussian (by moment matching), and then updating it using M different Kalman filters, one per value of S_t : see Figure 47. Unfortunately, it is hard to extend IMM to the smoothing case, unlike GPB2, a smoothing version of which is discussed in Section 6.1.3.

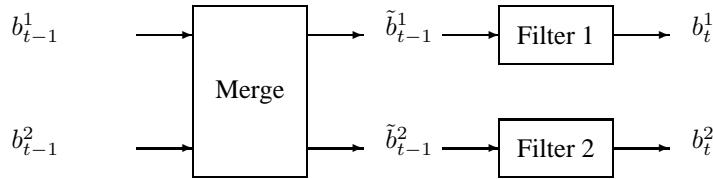


Figure 47: The IMM (interacting multiple models) algorithm for the case of a binary switch.

5.3.2 Viterbi approximation

If there are a large number of discrete variables, it may be too slow to perform M^2 or even M KF updates, as required by GPB2 and IMM. Instead, one can enumerate the discrete values in a priori order of probability. (Computing their posterior probability is as expensive as an exact update step.) This makes sense for a DBN such as the fault diagnosis model in Figure 25, where there is a natural ordering on the discrete values: one fault is much more likely than two faults, etc. However, it would not be applicable to the data association DBN in Figure 26, where there is no such ordering.

5.4 Belief state = set of samples (particle filtering)

The basic idea behind particle filtering¹² is to approximate the belief state by a set of weighted particles or samples:

$$P(X_t|y_{1:t}) \approx \sum_{i=1}^{N_s} w_t^i \delta(X_t, X_t^i)$$

(In this section, X_t^i means the i 'th sample of X_t , and $X_{t,i}$ means the i 'th component of X_t .) This is a non-parametric approach, and hence can handle non-linearities, multi-modal distributions, etc. The advantage over discretization is that the method is adaptive, placing more particles (corresponding to a finer discretization) in places where the probability density is higher.

Given a prior of this form, we can compute the posterior using importance sampling. In importance sampling, we assume the target distribution, $\pi(x)$, is hard to sample from; instead, we sample from a proposal or importance distribution $q(x)$, and weight the sample according to $w^i \propto \pi(x)/q(x)$. (After we have finished sampling, we can normalize all the weights so $\sum_i w^i = 1$). We can use this to sample paths with weights

$$w_t^i \propto \frac{P(x_{1:t}^i|y_{1:t})}{q(x_{1:t}^i|y_{1:t})}$$

The probability of a sample path, $P(x_{1:t}^i|y_{1:t})$, can be computed recursively using Bayes rule. Typically we will want the proposal distribution to be recursive also, i.e., $q(x_{1:t}|y_{1:t}) = q(x_t|x_{1:t-1}, y_{1:t})q(x_{1:t-1}|y_{1:t-1})$. In this case we have

$$\begin{aligned} w_t^i &\propto \frac{P(y_t|x_t^i)P(x_t^i|x_{t-1}^i)P(x_{1:t-1}^i|y_{1:t-1})}{q(x_t^i|x_{1:t-1}^i, y_{1:t})q(x_{1:t-1}^i|y_{1:t-1})} \\ &= \frac{P(y_t|x_t^i)P(x_t^i|x_{t-1}^i)}{q(x_t^i|x_{1:t-1}^i, y_{1:t})} w_{t-1}^i \\ &\stackrel{\text{def}}{=} \hat{w}_t^i \times w_{t-1}^i \end{aligned}$$

where we have defined \hat{w}_t^i to be the incremental weight.

For filtering, we usually only care about the posterior marginal $P(X_t|y_{1:t})$, as opposed to the full posterior $P(X_{1:t}|y_{1:t})$. Hence we use the following proposal: $q(x_t|x_{1:t-1}^i, y_{1:t}) = q(x_t|x_{t-1}^i, y_t)$. This means we only need to store x_t in each particle, instead of the whole trajectory, $x_{1:t}$. With this proposal, the weights simplify to

$$\hat{w}_t^i = \frac{P(y_t|x_t^i)P(x_t^i|x_{t-1}^i)}{q(x_t^i|x_{t-1}^i, y_t)} \quad (1)$$

5.4.1 The resampling step

The algorithm described so far is known as sequential importance sampling (SIS). A well known problem with SIS is that, over time, one of the normalized importance weights tends to 1, while the others tend to zero (even if we use the optimal proposal distribution: see Section 5.4.2). Hence a large number of samples are effectively wasted, since their weight is negligible. This is called particle “impoverishment”.

An estimate of the “effective” number of samples is given by

$$N_{eff} = \frac{1}{\sum_{i=1}^{N_s} (w_t^i)^2} \quad (2)$$

If this drops below some threshold, we can sample with replacement from the current belief state. Essentially this throws out particles with low weight and replicates those with high weight. This is called resampling, and can be done in $O(N_s)$ time using a variety of methods. After resampling, the weights are reset to the uniform distribution: the past weights are reflected in the frequency with which particles are sampled, and do not need to be kept.

¹²Particle filtering is also known as sequential Monte Carlo, sequential importance sampling with resampling (SISR), the bootstrap filter, the condensation algorithm, survival of the fittest, etc.

function $[\{x_t^i, w_t^i\}_{i=1}^{N_s}] = \text{PF}(\{x_{t-1}^i, w_{t-1}^i\}_{i=1}^{N_s}, y_t)$
for $i = 1 : N_s$

Sample $x_t^i \sim q(\cdot | x_{t-1}^i, y_t)$
Compute \hat{w}_t^i from Equation 1

$$w_t^i = \hat{w}_t^i \times w_{t-1}^i$$

Compute $w_t = \sum_{i=1}^{N_s} w_t^i$

Normalize $w_t^i := w_t^i / w_t$

Compute N_{eff} from Equation 2

if $N_{eff} < \text{threshold}$

$$\pi = \text{resample}(\{w_t^i\}_{i=1}^{N_s})$$

$$x_t^i = x_t^\pi$$

$$w_t^i = 1/N_s$$

Figure 48: Pseudo-code for a generic particle filter. The resample step samples indices with replacement according to their weight; the resulting set of sampled indices is called π . The line $x_t^i = x_t^\pi$ simply duplicates or removes particles according to the chosen indices.

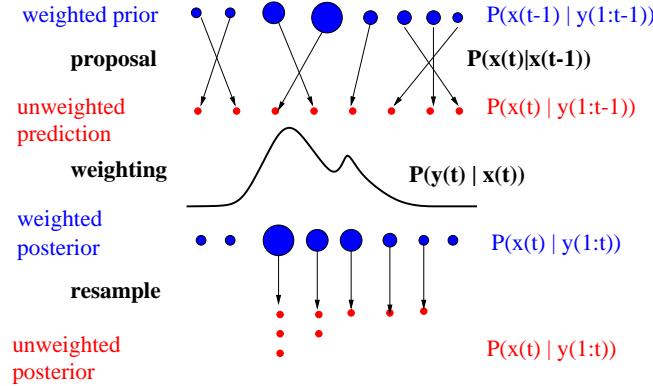


Figure 49: Particle filtering, where the proposal is the transition prior, $q(x_t | x_{t-1}^i, y_t) = P(x_t | x_{t-1}^i)$, and hence the incremental weight is the likelihood, $\hat{w}_t^i = P(y_t | x_t^i)$. In this example, we start with a set of (unequally) weighted particles, representing the prior, and transform them to a set of (equally) weighted particles representing the posterior. Notice how the resampling step decreases diversity of the population, and makes all the weights uniform.

Particle filtering is just sequential importance sampling with resampling. The resampling step was the key innovation in the 1990s; SIS itself has been around since at least the 1950s. The overall algorithm is sketched in Figure 48. Its simplicity and generality is one reason for the algorithm's widespread popularity.

Although resampling kills off unlikely particles, it also reduces the diversity of the population (which is why we don't do it at every time step; if we did, then $w_t^i = \hat{w}_t^i$). This is a particularly severe problem if the system is highly deterministic (e.g., if the state space contains static parameters). A simple solution is to apply a kernel around each particle and then resample from the kernel. An alternative is to use an MCMC step, that samples from $P(X_t | y_{1:t})$, thus introducing diversity without affecting the correctness of the algorithm.

5.4.2 The proposal distribution

By far the most common proposal is to sample from the transition prior: $q(x_t | x_{t-1}^i, y_t) = P(x_t | x_{t-1}^i)$. In this case, the weights simplify to $\hat{w}_t^i = P(y_t | x_t^i)$. This version of particle filtering is sometimes called the condensation algorithm: see Figure 49.

For predicting the future, sampling from the transition prior is adequate, since there is no future evidence. But for monitoring/ filtering, it is not very efficient, since it amounts to “guess until you hit”. For example, if the transitions

are highly stochastic, sampling from the prior will result in particles being proposed all over the state-space; if the observations are highly informative, most of the particles will get “killed off” (i.e., assigned low weight). In such a case, it makes more sense to first look at the evidence, y_t , and then propose:

$$q(x_t|x_{t-1}^i, y_t) = P(x_t|x_{t-1}^i, y_t) \propto P(y_t|x_t)P(x_t|x_{t-1}^i)$$

In fact, one can prove this is the optimal proposal distribution, in the sense of minimizing the variance of the weights. (High variance distributions are wasteful, since they correspond to the case where some weights are high and some are low; the low weight particles are “wasted”.)

If we use the optimal proposal, the incremental weight is given by the one-step-ahead likelihood:

$$\hat{w}_t^i = P(y_t|x_{t-1}^i) = \int_{x_t} P(y_t|x_t)P(x_t|x_{t-1}^i)$$

In general this may be intractable to compute, but if the observation model, $P(Y_t|X_t)$, is linear-Gaussian, and the process noise is Gaussian, i.e.,

$$\begin{aligned} P(X_t|x_{t-1}^i) &= \mathcal{N}(X_t; f_t(x_{t-1}^i), Q_t) \\ P(Y_t|X_t) &= \mathcal{N}(y_t; C_t X_t, R_t) \end{aligned}$$

then one can use the standard Kalman filter rules¹³ to show that

$$\begin{aligned} P(X_t|x_{t-1}^i, y_t) &= \mathcal{N}(X_t; x_{t|t}, P_{t|t}) \\ \hat{w}_t^i = P(y_t|x_{t-1}^i) &= \mathcal{N}(y_t; C_t f_t(x_{t-1}^i), Q_t + C_t R_t C_t') \end{aligned}$$

where

$$\begin{aligned} P_{t|t}^{-1} &= Q_t^{-1} + C_t^T R_t^{-1} C_t \\ x_{t|t} &= \Sigma_t (Q_t^{-1} f_t(x_{t-1}^i) + H_t' R_t^{-1} y_t) \end{aligned}$$

If the model does not satisfy these requirements, one can still use a Gaussian approximation to $P(X_t|x_{t-1}^i, y_t)$ as the proposal. For example, if we store a mean and covariance in each particle, we can compute $P(X_t|x_{t-1}^i, y_t) \approx \mathcal{N}(X_t; \hat{x}_{t|t}^i, P_{t|t}^i)$ using the EKF/UKF. We can then sample from this, and copy $P_{t|t}^i$ to the newly sampled particle. This is called the extended/ unscented particle filter. The sampling step can overcome bias introduced by the deterministic approximation. Not surprisingly, using a clever proposal dramatically improves performance.

If the process noise is non-Gaussian, but the observation model is (conditional) linear-Gaussian, we can propose from the likelihood and weight by the transition prior.

5.4.3 Rao-Blackwellised Particle Filtering (RBPF)

The Rao-Blackwell theorem shows how to improve upon any given estimator under every convex loss function. At its core is the following well-known identity:

$$\text{Var}[\tau(X, R)] = \text{Var}[E(\tau(X, R)|R)] + E[\text{Var}(\tau(X, R)|R)]$$

where $\tau(X, R)$ is some estimator of X and R . Hence $\text{Var}[E(\tau(X, R)|R)] \leq \text{Var}[\tau(X, R)]$, so $\tau'(X, R) = E(\tau(X, R)|R)$ is a lower variance estimator. So if we can sample R and compute the expectation of X given R analytically, we will need less samples (for a given accuracy). Of course, less samples does not necessarily mean less time: it depends on whether we can compute the conditional expectation efficiently or not.

We now give a simple but useful example of this idea. Consider a switching SSM. If we knew $S_{1:t}$, we could compute $P(X_t|y_{1:t}, s_{1:t})$ exactly using a Kalman filter. Since $S_{1:t}$ is unknown, we can sample it, and then, for each such sample, integrate out X_t analytically using the Kalman filter. Now we are only sampling in a small discrete space, instead of a large hybrid space, so the performance is much better, both in theory and practice.

¹³Specifically, we use the information form, since the old belief state is a delta function: $x_{t-1|t-1} = x_{t-1}^i$, $P_{t-1|t-1} = 0$. Hence $x_{t|t-1} = f(x_{t-1}^i)$ and $P_{t|t-1} = Q_t$. By Equation 15.43, $P_{t|t}^{-1} = P_{t|t-1}^{-1} + C_t^T R_t^{-1} C_t$, and by Equation 15.54, $P_{t|t}^{-1} \hat{x}_{t|t} = P_{t|t-1}^{-1} \hat{x}_{t|t-1} + C_t^T R_t^{-1} y_t$.

function $\{s_t^i, \mu_t^i, \Sigma_t^i, w_t^i\} = \text{RBPF-SSSM-prior}(\{s_{t-1}^i, \mu_{t-1}^i, \Sigma_{t-1}^i, w_{t-1}^i\}, y_t)$
for $i = 1 : N_s$

- Sample $s_t^i \sim P(S_t | s_{t-1}^i)$
- $(\mu_t^i, \Sigma_t^i, \hat{w}_t^i) = \text{KF}(\mu_{t-1}^i, \Sigma_{t-1}^i, y_t, \theta_{s_t^i})$
- $w_t^i = \hat{w}_t^i \times w_{t-1}^i$
- Compute $w_t = \sum_{i=1}^{N_s} w_t^i$
- Normalize $w_t^i := w_t^i / w_t$
- Compute N_{eff} from Equation 2
- if $N_{eff} < \text{threshold}$
- $\pi = \text{resample}(\{w_t^i\}_{i=1}^{N_s})$
- $s_t^\pi = s_t^{\pi}, \mu_t^\pi = \mu_t^\pi, \Sigma_t^\pi = \Sigma_t^\pi$
- $w_t^\pi = 1/N_s$

Figure 50: Pseudo-code for Rao-Blackwellised particle filtering applied to a switching SSM, where we sample from the prior.

Algorithmically, what this means is that each particle contains a sampled value for S_t , which implicitly represents a whole trajectory, $S_{1:t}$, plus the sufficient statistics for the Kalman filter conditioned on this trajectory, $\mu_t^i = E[X_t | y_{1:t}, s_{1:t}^i]$ and $\Sigma_t^i = \text{Cov}[X_t | y_{1:t}, s_{1:t}^i]$. If we propose from the transition prior for S_t , we can compute the weight using the marginal likelihood $P(y_t | s_{1:t}^i)$:

$$\begin{aligned} \hat{w}_t^i &= \frac{P(y_t | s_t, s_{1:t-1}^i) P(s_t | s_{t-1}^i)}{P(s_t | s_{t-1}^i)} \\ &= \int_{x_t} P(y_t | x_t, s_t) P(x_t | s_t, s_{1:t-1}^i) \\ &= \mathcal{N}(y_t; C_{st} A_{st} \mu_{t-1}^i, C_{st} (A_{st} \Sigma_{t-1}^i A_{st}^T + Q_{st}) C_{st}^T + R_{st}) \end{aligned}$$

This term is just a byproduct of applying the Kalman filtering equations to the sufficient statistics $\mu_{t-1}^i, \Sigma_{t-1}^i$ with the parameter set $\theta_{s_t} = (A_{st}, C_{st}, Q_{st}, R_{st})$. The overall algorithm is shown in Figure 50. Note that we can use this algorithm even if S_t is not discrete.

If the number of values of S_t is sufficiently small, and the transition model for S_t is not very informative, it might be beneficial to use the optimal proposal distribution, which is given by

$$P(S_t = s | s_{1:t-1}^i, y_{1:t}) \propto P(y_t | S_t = s, s_{1:t-1}^i, y_{1:t-1}) P(S_t = s | s_{t-1}^i)$$

As usual, the incremental weight is just the normalizing constant for the optimal proposal:

$$\hat{w}_t^i = \sum_s P(y_t | S_t = s, s_{1:t-1}^i, y_{1:t-1}) P(S_t = s | s_{t-1}^i)$$

The modified algorithm is shown in Figure 51. This is more expensive than sampling from the transition prior, since for each particle i , we must loop over all states s in order to compute the proposal distribution. However, this may require fewer particles, making it faster overall.

5.4.4 Particle filtering for DBNs

So far, we have assumed it is easy to sample from the transition model, $P(X_t | X_{t-1}^i)$, and to compute the weight $P(y_t | x_t^i)$. But what if the transition and observation models are represented by a complex DBN, instead of a simple parametric function? To apply particle filtering to a DBN, we can use the likelihood weighting (LW) routine in Figure 52 to sample x_t^i and compute \hat{w}_t^i , given x_{t-1}^{i-1} and y_t . The proposal distribution that LW corresponds to depends on which nodes of the DBN are observed. In the simplest case of an HMM, where the observation is at a leaf node,

```

function  $\{s_t^i, \mu_t^i, \Sigma_t^i, w_t^i\} = \text{RBPF-SSSM-opt}(\{s_{t-1}^i, \mu_{t-1}^i, \Sigma_{t-1}^i, w_{t-1}^i\}, y_t)$ 
for  $i = 1 : N_s$ 
  for each  $s$ 
     $(\mu^s, \Sigma^s, L(s)) = \text{KF}(\mu_{t-1}^i, \Sigma_{t-1}^i, y_t, \theta_s)$ 
     $q(s) = L(s) \times P(S_t = s | s_{t-1}^i)$ 
     $\hat{w}^s = \sum_s q(s)$ 
    Normalize  $q(s) := q(s) / \hat{w}^s$ 
    Sample  $s \sim q(\cdot)$ 
     $s_t^i = s, \mu_t^i = \mu^s, \Sigma_t^i = \Sigma^s, \hat{w}_t^i = \hat{w}^s$ 
     $w_t^i = \hat{w}_t^i \times w_{t-1}^i$ 
  ...

```

Figure 51: Pseudo-code for RBPF applied to a switching SSM, where we sample from the optimal proposal. The third return argument from the KF routine is $L(s) = P(y_t | S_t = s, s_{1:t-1}^i, y_{1:t-1})$ means the code continues as in Figure 50.

```

function  $[x_t^i, \hat{w}_t^i] = \text{LW}(x_{t-1}^i, y_t)$ 
 $\hat{w}_t^i = 1$ 
 $x_t^i = \text{empty vector of length } N$ 
for each node  $i$  in topological order
  Let  $u$  be the value of  $\text{Pa}(X_t^i)$  in  $(x_{t-1}^i, x_t^i)$ 
  If  $X_t^i$  not in  $y_t$ 
    Sample  $x_t^i \sim P(X_t^i | \text{Pa}(X_t^i) = u)$ 
  else
     $x_t^i = \text{the value of } X_t^i \text{ in } y_t$ 
     $\hat{w}_t^i = \hat{w}_t^i \times P(X_t^i = x_t^i | \text{Pa}(X_t^i) = u)$ 

```

Figure 52: Pseudo-code for likelihood weighting.

LW samples from the transition prior, $P(X_t^i|x_{t-1}^i)$, and then computes the weight as $w = P(y_t|x_t^i)$. (We discuss how to improve this below.)

In general, some of the evidence might occur at arbitrary locations within the DBN slice. In this case, the proposal is $q(x_t, y_t) = \prod_j P(x_{t,j}|\text{Pa}(X_{t,j}))$, and the weight is $w(x_t, y_t) = \prod_j P(y_{t,j}|\text{Pa}(Y_{t,j}))$, where $x_{t,j}$ is the (value of the) j 'th hidden node at time t , and $y_{t,j}$ is the (value of the) j 'th observed node at time t , and the parents of both $X_{t,j}$ and $Y_{t,j}$ may contain evidence. This is consistent, since

$$P(x_t, y_t) = \prod_j P(x_{t,j}|\text{Pa}(X_{t,j})) \times \prod_j P(y_{t,j}|\text{Pa}(Y_{t,j})) = q(x_t, y_t)w(x_t, y_t)$$

Improving the proposal distribution Since the evidence usually occurs at the leaves, likelihood weighting effectively samples from the transition prior without looking at the evidence y_t . A general way to take the evidence into account while sampling in a Bayes net is called “evidence reversal”. This means applying the rules of “arc reversal” until all the evidence nodes become parents instead of leaves.

To reverse an arc from $X \rightarrow Y$, we must add Y 's unique parents, Y_p , to X , and add X 's unique parents, X_p , to Y (both nodes may also share common parents, C): see Figure 53. The CPDs in the new network are given by

$$\begin{aligned} P(Y|Y_p, X_p, C) &= \sum_x P(Y, x|Y_p, X_p, C) \\ &= \sum_x P(Y|x, Y_p, C)P(x|X_p, C) \\ P(X|Y, Y_p, X_p, C) &= \frac{P(Y|X, Y_p, X_p, C)P(X|Y_p, X_p, C)}{P(Y|Y_p, X_p, C)} \\ &= \frac{P(Y|X, Y_p, C)P(X|X_p, C)}{P(Y|Y_p, X_p, C)} \end{aligned}$$

Note that X_p , Y_p and C could represent sets of variables. Hence the new CPDs could be much larger than before the arc reversal.

In the case of an HMM, the arc reversal operation is shown in Figure 54. Applying the above rules with $X = X_t$, $Y = Y_t$, $X_p = X_{t-1}$, $Y_p = C = \emptyset$ yields the new CPDs:

$$\begin{aligned} P(Y_t|X_{t-1}) &= \sum_{x_t} P(Y_t|x_t)P(x_t|X_{t-1}) \\ P(X_t|X_{t-1}, Y_t) &= \frac{P(Y_t|X_t)P(X_t|X_{t-1})}{P(Y_t|X_{t-1})} \end{aligned}$$

For a general DBN, we may apply arc reversal to one or more of the leaf nodes, in order to improve the efficiency of the proposal distribution.

An alternative to using likelihood weighting combined with arc reversal is to create a junction tree from the original 2TBN. The evidence, E , consists of all of the nodes in slice $t - 1$ (the values are stored in the old particle) plus a subset of the nodes in slice t (the new observations). One can sample values for the remaining nodes, H , from the optimal distribution $P(H|E)$ using the following two pass algorithm. First collect to the root, as usual, but then, in the distribute phase, draw a random sample from $P(X_{C_i \setminus S_i}|x_{S_i}, E)$ for each clique C_i , where S_i is the separator nearer to the root. Although this samples from the optimal distribution, it is not usually possible to implement the necessary operations for manipulating the potentials if some of the variables are continuous and non-Gaussian. (Of course, arc reversal cannot be applied in this case either.)

Combining particle filtering with a factored belief state It is possible to combine particle filtering with the Boyen-Koller algorithm (see Section 5.1.1), i.e., to approximate the belief state by

$$\hat{P}(X_t|y_{1:t}) \approx \prod_{c=1}^C \frac{1}{N_c} \sum_{i=1}^{N_c} \delta(X_{t,c}, x_{t,c}^i)$$

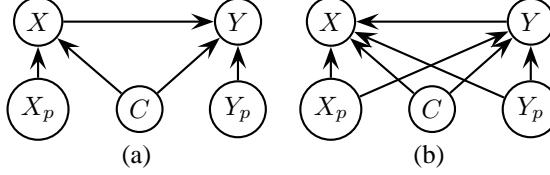


Figure 53: Arc reversal. (a) The original network. (b) The network after reversing the $X \rightarrow Y$ arc. The modified network encodes the same probability distribution as the original network.

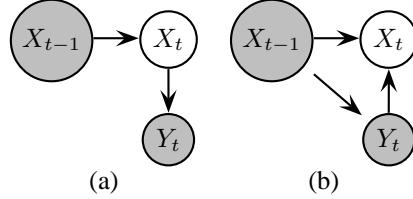


Figure 54: A DBN (a) before and (b) after evidence reversal.

where C is the number of clusters, and N_c is the number of particles in each cluster (assumed uniformly weighted). By applying particle filtering to a set of smaller state-spaces (each cluster), one can reduce the variance, at a cost of increasing the bias by using a factored representation. However, it now becomes much more complicated to propagate each particle.

5.5 Belief state = variable sized

In some applications, the size of the state-space is not fixed in advance. For instance, when we are tracking multiple objects, each measurement could have been caused by any of them, the background, or perhaps a new object that has entered the “scene”. This problem arises in visual tracking, tracking missiles with radar, and also in mobile robotics, in particular in the SLAM (simultaneous localization and mapping) problem.

A standard heuristic way to detect the presence of new objects is if an observation arises in a location that was not expected. If we have a prior over each object’s position, we can compute a distribution over the expected measurement position:

$$P(Y_t|y_{1:t-1}) = \int_x P(Y_t|X_t = x)P(X_t = x|y_{1:t-1})$$

If these densities are Gaussians, this is often represented as a confidence ellipse or validation gate. If an observation falls inside this ellipse, we assume it was generated by the object. If there is more than one observation inside the ellipse, or if the observation falls inside more than one ellipse, we can either assign the observation to the nearest target (using Mahalanobis distance), or compute the likelihood of all possible joint assignments of observations to targets, and pick the most likely one (Note that the nearest neighbor rule might assign the same measurement to multiple objects, which leads to inaccuracies.)

If an observation does not fall inside the validation gate of any existing object, it could either be due to a new object, or due to background clutter. Hence we consider both hypotheses, and add the new object to a provisional list. Once the object on the provisional list receives a minimum number of measurements inside its validation gate, it is added to the state space.

It is also possible for an object to be removed from the state space if it has not been updated recently (e.g., it left the scene). Hence in general we must allow the state space to grow and shrink dynamically.

A more rigorous approach to inference in such variable-sized state-spaces models is to use particle filtering. This is easy since each particle can have a different dimensionality. We give a detailed example below, in the context of online model selection.

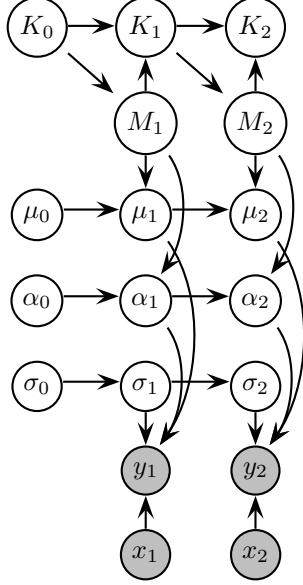


Figure 55: A DBN for sequential Bayesian model selection.

5.5.1 Non-linear regression with online model selection

We discussed a model for online parameter estimation for non-linear regression in Section 2.11. Now we will let the number of basis functions (and hence the size of the state space) vary over time. Let K_t be the number of bases at time t , and let M_t be a random variable with five possible values: B=birth, D=death, S=split, M=merge and N=no-change. These specify the ways in which K_t can increase/decrease by 1 at each step. Birth means we create a new basis function; its position can depend on the previous basis function centers, μ_{t-1} , as well as the current data point, x_t ; death means we remove a basis function at random, split means we split one center into two, and merge means we combine two centers into one. We enforce that $0 \leq K_t \leq K_{max}$ at all times. The DBN is shown in Figure 55, and the CPDs are as follows.

$$\begin{aligned}
P(M_t = (B, D, S, M, N)|K_{t-1} = k) &= \begin{cases} \left(\frac{1}{2}, 0, 0, 0, \frac{1}{2}\right) & \text{if } k = 0 \\ \left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, 0, \frac{1}{4}\right) & \text{if } k = 1 \\ \left(0, \frac{1}{3}, 0, \frac{1}{3}, \frac{1}{3}\right) & \text{if } k = K_{max} \\ \left(\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right) & \text{otherwise} \end{cases} \\
P(K_t = k'|K_{t-1} = k, M_t = m) &= \begin{cases} \delta(k', k+1) & \text{if } m = B \text{ or } m = S \\ \delta(k', k-1) & \text{if } m = D \text{ or } m = M \\ \delta(k', k) & \text{if } m = N \end{cases} \\
P(\mu_t|\mu_{t-1}, M_t = m) &= \begin{cases} \mathcal{N}(\mu_t; \mu_{t-1}, \delta_\mu I) & \text{if } m = N \\ \mathcal{N}(\mu_t; \text{birth}(\mu_{t-1}), \cdot) & \text{if } m = B \\ \mathcal{N}(\mu_t; \text{death}(\mu_{t-1}), \cdot) & \text{if } m = D \\ \mathcal{N}(\mu_t; \text{split}(\mu_{t-1}), \cdot) & \text{if } m = S \\ \mathcal{N}(\mu_t; \text{merge}(\mu_{t-1}), \cdot) & \text{if } m = M \end{cases} \\
P(\alpha_t|\alpha_{t-1}, M_t = m) &= \begin{cases} \mathcal{N}(\alpha_t; \alpha_{t-1}, \delta_\alpha I) & \text{if } m = N \\ \mathcal{N}(\alpha_t; \text{grow}(\alpha_{t-1}), \cdot) & \text{if } m = B \text{ or } m = S \\ \mathcal{N}(\alpha_t; \text{shrink}(\alpha_{t-1}), \cdot) & \text{if } m = D \text{ or } m = M \end{cases} \\
P(\log \sigma_t | \log \sigma_{t-1}) &= \mathcal{N}(\log \sigma_t; \log \sigma_{t-1}, \delta_\sigma I) \\
P(y_t|x_t, \alpha_t, \mu_t) &= \mathcal{N}(y_t; D(\mu_t, x_t)\alpha_t, \sigma_t)
\end{aligned}$$

The function $\text{birth}(\mu_{t-1})$ takes in the vector of RBF centers and adds a new one to the end, according to some heuristic. In principle, this heuristic could depend on x_t as well (not shown). Call the new center μ_{birth} . The confidence

associated with μ_{birth} is yet another free parameter; suppose it is Q_{birth} ; then the full CPD would be

$$P(\mu_t | \mu_{t-1}, M_t = B) = \mathcal{N} \left(\mu_t; (\mu'_{t-1} \quad \mu'_{\text{birth}})', \begin{pmatrix} 0 & 0 \\ 0 & Q_{\text{birth}} \end{pmatrix} \right)$$

The 0 terms in the covariance matrix do not mean we are certain about the locations of the other centers, only that we have not introduced any extra noise, i.e.,

$$\text{Cov}(\mu_t | y_{1:t}, M_t = B) = \begin{pmatrix} \text{Cov}(\mu_{t-1} | y_{1:t-1}) & 0 \\ 0 & Q_{\text{birth}} \end{pmatrix}$$

Of course, we could relax this assumption. If we were full-blooded Bayesians, we would now add priors to all of our parameters (such as Q_{birth} , δ_α , etc.); these priors would in turn have their own (hyper-)parameters; we would continue in this way until the resulting model is insensitive to the parameters we choose. This is called hierarchical Bayesian modelling.

Despite the apparent complexity, it is actually quite easy to apply particle filtering to this model. [AdFD00] suggest a more complex scheme, which combines particle filtering with reversible jump MCMC [Gre98]. (Regular MCMC can only be applied if the state-space has a fixed size.) We can use the simpler method of particle filtering because we included M_t in the state-space.

6 Approximate smoothing

There are two main kinds of approximate smoothing algorithms: those that make a single forwards-backwards pass over the data (two-filter smoothers), and those that make multiple passes. Exact inference algorithms only need to make a single forwards-backwards pass, but in general, approximate inference can achieve better results if it is allowed to make multiple passes. We will briefly discuss a variety of algorithms below.

6.1 Two-filter smoothing

As for filtering, we can classify two-filter smoothers based on which representation they choose for the forwards and backwards messages. By forwards message we mean $\alpha_t = P(X_t | y_{1:t})$, the filtered estimate; by backwards message we mean either $\gamma_t = P(X_t | y_{1:T})$, the smoothed estimate, or $\beta_t = P(y_{t+1:T} | X_t)$, the conditional likelihood. (If the dynamics are invertible, we can define a more natural backwards filtered estimate, $P(X_t | y_{t+1:T})$. However, in general we wish to avoid the invertibility assumption. We discuss the relative merits of using β vs γ in Section 4.4.3.) The backwards messages can be computed by modifying the forward recursive update equations.

6.1.1 Belief state = discrete distribution

We have already discussed how to perform an exact backwards step, either in β or γ form, for discrete-state DBNs in Section 4. In Section 5.1, we discussed some approximate filtering methods if the exact ones are too slow. We now discuss how to extend these to the smoothing case.

ADF/BK algorithm It is fairly easy to modify the interface algorithm to use a factored representation, as we saw in Section 5.1.1. The same modifications can be used in the backwards pass in the obvious way.

Beam search The standard approximation is to compute β only for those states i which were considered at time t of the forwards pass, i.e., for which $\hat{\alpha}_t(i) > 0$, where $\hat{\alpha}_t$ is the pruned version of α_t .

6.1.2 Belief state = Gaussian distribution

Kalman smoother We saw the β and γ versions of the Kalman smoother in Chapter ???. For completeness, we restate the equations here. First we compute the following predicted quantities (or we could pass them in from the filtering

stage):

$$\begin{aligned} x_{t+1|t} &= A_{t+1}x_{t|t} \\ P_{t+1|t} &= A_{t+1}P_{t|t}A_{t+1}^T + Q_{t+1} \end{aligned}$$

Then we compute the smoother gain matrix.

$$J_t = P_{t|t}A_{t+1}^TP_{t+1|t}^{-1}$$

Finally, we compute our estimates of the mean, variance, and cross variance $P_{t,t-1|T} = \text{Cov}[X_{t-1}, X_t | y_{1:T}]$.

$$\begin{aligned} x_{t|T} &= x_{t|t} + J_t(x_{t+1|T} - x_{t+1|t}) \\ P_{t|T} &= P_{t|t} + J_t(P_{t+1|T} - P_{t+1|t})J_t' \\ P_{t-1,t|T} &= J_{t-1}P_{t|T} \end{aligned}$$

As for Kalman filtering, We can generalize the Kalman smoother equations to apply to any linear-Gaussian DBN by using the junction tree algorithm discussed in Section 4, but modifying the definition of summation, multiplication and division, so that these operations can be applied to Gaussian potentials instead of discrete ones.

Extended Kalman smoother As we saw in Section 5.2.2, the EKF linearizes the system and then applies the standard KF equations. Similarly, the extended Kalman smoother just applies the above smoothing equations to the same linearized system. Note that the system is linearized about the filtered estimate $\hat{x}_{t|t}$. Although it is possible to relinearize about the backwards estimates, this requires that the dynamics be invertible, which will not generally be the case.

Unscented Kalman smoother It is possible to apply the unscented transform to either the β or γ version of the backwards pass, to get an unscented Kalman smoother. We leave the details as an exercise.

ADF/Moment matching smoother We will discuss the moment matching smoother in the context of switching SSMs in Section 6.1.3 and in the context of expectation propagation in Section 6.2.

6.1.3 Belief state = mixture of Gaussians

The GPB2 filter corresponds to a collect operation in the following junction tree, where we use weak marginalization to marginalize out discrete nodes from a mixture of Gaussians:

$$(Z_{t-1}, Z_t) - [Z_t] - (Z_t, Z_{t+1}) - [Z_{t+1}] - \dots$$

where $Z_t = (S_t, X_t)$. (We ignore observed nodes for simplicity.) The cliques are in round brackets and the separators are in the square brackets. (Smaller cliques are possible.)

Hence we can create a GPB2 smoother simply by running the backwards pass in the same tree. No additional assumptions are necessary.

6.1.4 Belief state = set of particles

One approach to particle smoothing is to sample complete trajectories, and then reweight them given all the evidence. Another is to sample in both the forwards and backwards direction; unfortunately, this takes $O(TN_s^2)$ time, and is therefore generally considered intractable. A much more popular sampling technique for the offline case is Gibbs sampling, discussed in Section 6.4.

6.2 Expectation propagation (EP)

Expectation propagation generalizes assumed density filtering (Section 5.2.4) to the batch context. It is less sensitive to the order in which the Bayesian updates are performed because it can go back and re-optimize each term of the overall likelihood in the context of the other terms. We explain this in more detail below.

Recall that in ADF, the exact posterior is given by

$$\hat{p}(\theta) = \frac{l(\theta)q^{old}(\theta)}{Z}$$

where $q^{old}(\theta) \in \mathcal{F}$ is the prior (a member of the tractable family of distributions \mathcal{F}), $l(\theta)$ is the likelihood, and $Z = \int_{\theta} l(\theta)q^{old}(\theta)$. We then pick the best tractable approximation to this by computing $q^{new}(\theta) = \arg_{q \in \mathcal{F}} \min D(\hat{p}(\theta)||q(\theta))$. This can be viewed as updating the prior and then projecting the posterior, as in Figure 42. Another way to view it is as computing an approximation to the likelihood, $\tilde{l}(\theta)$, that is conjugate to the prior, such that when $\tilde{l}(\theta)$ is combined with $q^{old}(\theta)$, the resulting posterior will be in \mathcal{F} . This approximate likelihood can be inferred from

$$\tilde{l}(\theta) = \frac{\dot{p}(\theta)Z}{q^{old}(\theta)}$$

This makes it simple to remove the old likelihood from the joint, and then use a newly approximated likelihood, to get a better approximation.

A simple example of this algorithm applied to a switching SSM is shown in Figure 56. Here the ADF operations correspond to moment-matching for a mixture of Gaussians, as discussed in Section 5.3.1. The first forwards-backwards pass corresponds exactly to the GPB2 smoother. In subsequent iterations, we recompute each clique potential $P(Z_{t-1}, Z_t|y_{1:T})$ by absorbing α messages from the left and β messages from the right, and combining with the local potentials, $\psi_t(z_{t-1}, z_t) = P(y_t|z_t)P(z_t|z_{t-1})$. From this, we can compute the separator potentials, $q(z_t) \approx P(X_t, S_t|y_{1:T})$, using weak marginalization. From this, we can infer the equivalent tractable messages by division: $\alpha(z_t) \propto q(z_t)/\beta(z_t)$ and $\beta(z_t) \propto q(z_t)/\alpha(z_t)$, which can be propagated to future/past slices. Typically 2-3 passes is enough to improve performance.

Another application of EP is to iterate the BK algorithm. This allows the algorithm to recover from incorrect independence assumptions it made in earlier passes. Typically 2-3 passes is enough to improve performance.

6.3 Variational methods

Variational methods are discussed in Chapter ??, and can be applied to DBNs in a straightforward way. Some of these methods, such as structured variational approximations and variational EM, use the exact inference routines in Section 4 as subroutines.

As an example, we discuss how to apply a structured variational approximation to the switching SSM in Section 2.12.2. In this model, the observation switches between one of M chains, each of which evolves linearly and independently. For simplicity, we assume the observation noise is tied, $R_i = R$, and that $\mu_i = 0$.

Since exact inference is intractable in this model, we choose a tractable approximation of the following form:

$$Q(\{S_t, X_t\}) = \frac{1}{Z_Q} \left[\psi(S_1) \prod_{t=2}^T \psi(S_{t-1}, S_t) \right] \prod_{m=1}^M \psi(X_1^{(m)}) \prod_{t=2}^T \psi(X_{t-1}^{(m)}, X_t^{(m)})$$

where the ψ are unnormalized potentials and Z_Q is the normalizing constant. This corresponds to the graphical model in Figure 57.

The potentials for the discrete switching process are defined as follows.

$$\begin{aligned} \psi(S_1 = m) &= P(S_1 = m)q_1^{(m)} \\ \psi(S_{t-1}, S_t = m) &= P(S_t = m|S_{t-1})q_t^{(m)} \end{aligned}$$

where the $q_t^{(m)}$ are variational parameters of the Q distribution (we will discuss how to compute them shortly). These parameters play the same role as the observation likelihood, $P(Y_t|S_t = m)$, does in a regular HMM.

```

for t = 1 : T
     $\alpha_t(z_t) = 1$ 
     $\beta_t(z_t) = 1$ 
    if  $t == 1$ 
         $\psi_1(z_1) = P(y_1|z_1)P(z_1)$ 
    else
         $\psi_t(z_{t-1}, z_t) = P(y_t|z_t)P(z_t|z_{t-1})$ 
    while not converged
        for t = 1 : T
            if  $t == 1$ 
                 $\hat{p}(z_1) = \psi_1(z_1)\beta_1(z_1)$ 
            else
                 $\hat{p}(z_{t-1}, z_t) = \alpha_{t-1}(z_{t-1})\psi_t(z_{t-1}, z_t)\beta_t(z_t)$ 
                 $q(z_t) = \text{Collapse}_{s_{t-1}} \int_{x_{t-1}} \hat{p}(z_{t-1}, z_t)$ 
                 $\alpha_t(z_t) = \frac{q(z_t)}{\beta_t(z_t)}$ 
            for t = T : 1
                 $\hat{p}(z_{t-1}, z_t) = \alpha_{t-1}(z_{t-1})\psi_t(z_{t-1}, z_t)\beta_t(z_t)$ 
                 $q(z_{t-1}) = \text{Collapse}_{s_t} \int_{x_t} \hat{p}(z_{t-1}, z_t)$ 
                 $\beta_{t-1}(z_{t-1}) = \frac{q(z_{t-1})}{\alpha_{t-1}(z_{t-1})}$ 

```

Figure 56: Pseudo-code for EP applied to a switching SSM, based on [HZ02]. $Z_t = (X_t, S_t)$. Collapse means weak marginalization (moment matching). Note that the messages are equivalent to the separator potentials,

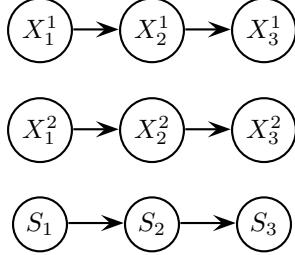


Figure 57: A structured variational approximation to the switching SSM in Figure 26. We remove all the vertical links, replacing them with variational parameters, but leave the horizontal (temporal) links intact.

The potentials for each chain are defined as follows.

$$\begin{aligned}\psi(X_1 = m) &= P(X_1) \left[P(Y_1 | X_1^{(m)}, S_1 = m) \right]^{h_1^{(m)}} \\ \psi(X_{t-1}, X_t) &= P(X_t^{(m)} | X_{t-1}^{(m)}) \left[P(Y_t | X_t^{(m)}, S_t = m) \right]^{h_t^{(m)}}\end{aligned}$$

where the $h_t^{(m)}$ are more variational parameters of Q , called the responsibilities. The vector $h_t^{(m)}$ is like a soft version of the switch variable S_t : if $h_t^{(m)} = 0$, it means $P(S_t = m) = 0$, so Y_t will not influence $X_t^{(m)}$; if $h_t^{(m)} = 1$, it means $P(S_t = m) = 1$, so Y_t will only influence $X_t^{(m)}$; for values in between, Y_t will have an intermediate effect on $X_t^{(m)}$.

We can compute locally optimal variational parameters by minimizing the KL divergence between Q and P :

$$D(Q||P) = \sum_{\{S_t\}} \int_{\{X_t\}} Q(\{S_t, X_t\}) \log \left[\frac{Q(\{S_t, X_t\})}{P(\{S_t, X_t\} | \{Y_t\})} \right] = H(Q) - \langle P \rangle$$

where $H(Q)$ is the entropy of Q and $\langle P \rangle$ is the expected value of P wrt Q . We would like to minimize $D(P||Q)$, but that is intractable to compute, because we would have to take expectations wrt P , which is a mixture of M^T Gaussians.

ADF/EP performs this optimization for a single time step. Hence the difference between EP and variational methods is that EP locally optimizes $D(P||Q)$, whereas variational methods (try to) globally optimize $D(Q||P)$.

Taking derivatives of $D(Q||P)$ wrt the variational parameters yields the following set of coupled equations:

$$\begin{aligned} h_t^{(m)} &= Q(S_t = m) \\ q_t^{(m)} &= \exp \left\{ -\frac{1}{2} \left\langle (Y_t - C^{(m)} X_t^{(m)'}) R^{-1} (Y_t - C^{(m)} X_t^{(m)}) \right\rangle \right\} \\ &= \exp \left\{ -\frac{1}{2} Y_t' R^{-1} Y_t + Y_t' R^{-1} C^{(m)} \langle X_t^{(m)} \rangle - \frac{1}{2} \text{tr} [C^{(m)'} R^{-1} C^{(m)} \langle X_t^{(m)} X_t^{(m)'} \rangle] \right\} \end{aligned}$$

The $Q(S_t = m)$ term can be computed using the forwards backwards algorithm on an HMM where the observation probabilities are given by $q_t^{(m)}$. The $\langle X_t^{(m)} \rangle$ and $\langle X_t^{(m)} X_t^{(m)'} \rangle$ terms, where the expectation is wrt Q , can be computed by running one Kalman smoother per chain, weighting the data by the responsibilities $h_t^{(m)}$. (Weighting the data by $h_t^{(m)}$ is equivalent to using unweighted data and a time-varying observation noise of $R_t^{(m)} = R/h_t^{(m)}$.) The above equations can be iterated until convergence.

In practice, it is essential to anneal the fixed point equations, to avoid getting stuck in bad local optima. This can be done by minimizing $\mathcal{T}H(Q) - \langle P \rangle$, where \mathcal{T} represents a temperature, and gradually reducing the temperature to $\mathcal{T} = 1$. This ensures a certain level of entropy in the system, and is called deterministic annealing.

Some simple experiments on this model suggest that the variational approximation with deterministic annealing performs about as well as the ADF algorithm (uniterated EP); without annealing, it performs much worse.

6.4 Gibbs sampling

MCMC methods in general, and Gibbs sampling particular, are discussed in Chapter ??, and can be applied to DBNs in a straightforward way. Unfortunately, Gibbs sampling mixes very slowly for time series, because the data are correlated. It is therefore crucial to use Rao-Blackwellisation.

An example of where this is possible is in switching SSMs. (The advantage of using Gibbs sampling in this context, as opposed to EP or variational methods, is that the algorithm will provably eventually converge to the right answer.) The basic algorithm is as follows:

1. Randomly initialise $s_{1:T}^0$.
2. For $i = 1, 2, \dots$ until convergence
 - (a) For $t = 1, \dots, T$, sample $s_t^i \sim P(S_t|y_{1:T}, s_{-t}^i)$
 - (b) Compute $x_{0:T}^i = \mathbb{E}[x_{0:T}|y_{1:t}^i, s_{1:T}^i]$

where $s_{-t}^i \stackrel{\text{def}}{=} (s_1^i, \dots, s_{t-1}^i, s_{t+1}^{i-1}, \dots, s_T^{i-1})$ contains new values of S_t to the left of t , and old values to the right. The final estimate is then obtained by averaging the samples $s_{1:T}^i$ and $x_{1:T}^i$, possibly discarding an initial segment corresponding to the burn-in period of the Markov chain.

The main issue is how to efficiently compute the sampling distribution,

$$P(s_t|y_{1:T}, s_{-t}) \propto P(s_t|s_{-t})P(y_{1:T}|s_{1:T})$$

The first term is just

$$P(s_t|s_{-t}) = P(s_t|s_{t-1})P(s_{t+1}|s_t)$$

The second term is given by

$$\begin{aligned} P(y_{1:T}|s_{1:T}) &= \sum_i P(y_{t+1:T}|y_{1:t}, s_{1:T}, X_t = i)P(y_{1:t}, X_t = i|s_{1:T}) \\ &= \sum_i P(y_{t+1:T}|s_{t+1:T}, X_t = i)P(y_{1:t}, X_t = i|s_{1:t}) \\ &= P(y_{1:t}|s_{1:t}) \sum_i P(y_{t+1:T}|s_{t+1:T}, X_t = i)P(X_t = i|y_{1:t}, s_{1:t}) \\ &= P(y_{1:t-1}|s_{1:t-1})P(y_t|y_{1:t-1}, s_{1:t}) \sum_i P(y_{t+1:T}|s_{t+1:T}, X_t = i)P(X_t = i|y_{1:t}, s_{1:t}) \end{aligned}$$

Hence, dropping terms that are independent of s_t ,

$$P(s_t|y_{1:T}, s_{-t}) \propto P(s_t|s_{t-1})P(s_{t+1}|s_t)P(y_t|y_{1:t-1}, s_{1:t}) \sum_i P(y_{t+1:T}|s_{t+1:T}, X_t = i)P(X_t = i|y_{1:t}, s_{1:t})$$

The key insight is that $P(y_{t+1:T}|s_{t+1:T}, X_t = i)$ is independent of $s_{1:t}$. Hence we can fix $s_{1:t}$ and compute $P(y_{t+1:T}|s_{t+1:T}, X_t = i)$ in a backwards pass using the β form of the backwards Kalman filter (which does not assume invertible dynamics). In the forwards pass, we can sample $s_{1:t}$ in order, computing $P(X_t = i|y_{1:t}, s_{1:t})$ and $P(y_t|y_{1:t-1}, s_{1:t})$ terms as we go.

The relative performance (in terms of accuracy vs. time) of Rao-Blackwellised Gibbs sampling, Rao-Blackwellised particle smoothing, expectation propagation, and structured variational approximations, all of which can be applied to switching SSMs, is unknown at this time.

7 Historical remarks and bibliography

The term “DBN” was first coined in [DK89]. HMMs with semi-tied mixtures of Gaussians are discussed in [Gal99]. Buried Markov models are discussed in [Bil00]. Mixed-memory Markov models are discussed in [SJ99]. ngram models for language are described in [Jel97, Goo01]; variable length Markov models are described in [RST96, McC95]. For continuous data, ngram models correspond to standard regression models, either linear or nonlinear. For a recent tree-based approach to regression, see [MCH02]. Factorial HMMs are discussed in [GJ97]. Coupled HMMs are discussed in [SJ95, Bra96]; the AVSR application is described in [NLP⁺02] (see also [NY00]), and the traffic application in [KCB00]. Variable-duration HMMs are described in [Rab89]. Segment models are described in [ODK96]. Hierarchical HMMs were introduced in [FST98], where they proposed an $O(T^3)$ inference algorithm. The $O(T)$ inference algorithm was introduced in [MP01]. Applications of HHMMs to robot navigation appear in [TRM01]. HHMMs are very closely related to abstract HMMs [BVW01] and cascaded finite automata [Moh96, PR97].

State-space models date back to the ’60s. See e.g., [DK01] for a recent account of applications of SSMs to time-series analysis, and [WH97] for a more Bayesian approach. Dynamic chain graphs are described in [DE00]. The example of sequential nonlinear regression is from [AdFD00]. Switching SSMs date back to the ’60s, and are commonly used in econometrics and control. The fault diagnosis example is from [KL01]. Data association is described in many places, e.g., [BSF88].

Exact inference for DBNs using unrolled junction trees was first described in [Kja92]; see [Kja95] for a more detailed account. The complexity results are from [Dar01]. The frontier algorithm is from [Zwe96]; a special case is described in Appendix B of [GJ97]. The interface algorithm is from [Mur02]. The log-space smoothing algorithm was introduced in [BMR97] and applied to a speech recognition task in [ZP00]. The $O(1)$ -space, $O(T^2)$ -time smoothing algorithm is classical, but can also be found in [Dar01]. The discussion of constant-space, constant-time smoothing is based in part on [RN02]. The example of conditionally tractable substructure is from [TDW02]. The idea of sampling the root nodes to get a factored belief state was first proposed in [Mur00] in the context of map learning, and was extended in [MTKW02].

The idea of classifying filtering algorithms based on the type of belief state representation is from [Min02]. The Boyen-Koller algorithm is described in [BK98b]; the smoothing extension is in [BK98a]. A method for computing the top N most probable paths in an HMM is described in [NG01]; the generalization to any Bayes net is in [Nil98]. The KF and EKF algorithms date back to the ’60s: see e.g., [AM79]. The unscented Kalman filter was first described in [JU97]; the presentation in Section 5.2.3 is closely based on [WdM01]. ADF is a classical technique; the presentation in Section 5.2.4 is based on [Min01]. Various approaches to filtering in switching SSMs are discussed in [SM80, TSM85]. The GPB2 and IMM filters are described in [BSL93]; the smoothing step is discussed in [Kim94, Mur98]. Exact inference in conditionally Gaussian models (even tree-structured ones) is NP-hard, as proved in [LP01].

See [AMGC02, LC98, Dou98] for good tutorials on particle filtering, and [DdFG01] for a collection of recent papers. Rao-Blackwellisation in general is discussed in [CR96]. The application to switching SSMs is discussed in [AK77, CL00, DGK99]. The unscented particle filter is described in [vdMDdFW00]. Likelihood weighting is described in [FC89, SP89], arc reversal in [Sha86], and the idea of combining them in [FC89, KKR95, CB97]. Using a junction tree for sampling is described in [Daw92]. Combining particle filtering and BK is described in [NPP02].

Multi-object tracking is discussed in [BSF88, BS90]. Exact solutions to the data association problem are discussed in [CH96]. SLAM is discussed in e.g., [DNCDW01]. The online model selection example is based on [AdFD00].

Particle smoothing is discussed in [GDW00, IB98]. Expectation propagation was introduced in [Min01]. The application to switching SSMs is from [HZ02]; the iterated BK algorithm (another instance of expectation propagation) is discussed in [MW01]. The structured variational approximation for switching SSMs is from [GH98]. The Rao-Blackwellised Gibbs sampler for switching SSMs is from [CK96]. [KN98] is a whole book devoted to deterministic and MCMC methods for switching SSMs.

References

- [AdFD00] C. Andrieu, N. de Freitas, and A. Doucet. Sequential Bayesian estimation and model selection for dynamic kernel machines. Technical report, Cambridge Univ., 2000.
- [AK77] H. Akashi and H. Kumamoto. Random sampling approach to state estimation in switching environments. *Automatica*, 13:429–434, 1977.
- [AM79] B. Anderson and J. Moore. *Optimal Filtering*. Prentice-Hall, 1979.
- [AMGC02] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking. *IEEE Trans. on Signal Processing*, 50(2):174–189, February 2002.
- [Bil00] J. Bilmes. Dynamic Bayesian multinets. In *UAI*, 2000.
- [Bil01] J. A. Bilmes. Graphical models and automatic speech recognition. Technical Report UWEETR-2001-0005, Univ. Washington, Dept. of Elec. Eng., 2001.
- [BK98a] X. Boyen and D. Koller. Approximate learning of dynamic models. In *NIPS-11*, 1998.
- [BK98b] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *UAI*, 1998.
- [BMR97] J. Binder, K. Murphy, and S. Russell. Space-efficient inference in dynamic probabilistic networks. In *IJCAI*, 1997.
- [Bra96] M. Brand. Coupled hidden Markov models for modeling interacting processes. Technical Report 405, MIT Lab for Perceptual Computing, 1996.
- [BS90] Y. Bar-Shalom, editor. *Multitarget-multisensor tracking : advanced applications*. Artech House, 1990.
- [BSF88] Y. Bar-Shalom and T. Fortmann. *Tracking and data association*. Academic Press, 1988.
- [BSL93] Y. Bar-Shalom and X. Li. *Estimation and Tracking: Principles, Techniques and Software*. Artech House, 1993.
- [BVW01] H. Bui, S. Venkatesh, and G. West. Tracking and surveillance in wide-area spatial environments using the Abstract Hidden Markov Model. *Intl. J. of Pattern Rec. and AI*, 2001.
- [BZ02] J. Bilmes and G. Zweig. The graphical models toolkit: An open source software system for speech and time-series processing. In *Intl. Conf. on Acoustics, Speech and Signal Proc.*, 2002.
- [CB97] A. Y. W. Cheuk and C. Boutilier. Structured arc reversal and simulation of dynamic probabilistic networks. In *UAI*, 1997.
- [CH96] I.J. Cox and S.L. Hingorani. An Efficient Implementation of Reid’s Multiple Hypothesis Tracking Algorithm and its Evaluation for the Purpose of Visual Tracking. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 18(2):138–150, 1996.
- [CK96] C. Carter and R. Kohn. Markov chain Monte Carlo in conditionally Gaussian state space models. *Biometrika*, 83:589–601, 1996.

- [CL00] R. Chen and S. Liu. Mixture Kalman filters. *J. Royal Stat. Soc. B*, 2000.
- [CR96] G Casella and C P Robert. Rao-Blackwellisation of sampling schemes. *Biometrika*, 83(1):81–94, 1996.
- [Dar01] A. Darwiche. Constant Space Reasoning in Dynamic Bayesian Networks. *Intl. J. of Approximate Reasoning*, 26:161–178, 2001.
- [Daw92] A. P. Dawid. Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing*, 2:25–36, 1992.
- [DdFG01] A. Doucet, N. de Freitas, and N. J. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer Verlag, 2001.
- [DE00] R. Dahlhaus and M. Eichler. Causality and graphical models for time series. In P. Green, N. Hjort, and S. Richardson, editors, *Highly structured stochastic systems*. Oxford University Press, 2000.
- [DGK99] A. Doucet, N. Gordon, and V. Krishnamurthy. Particle Filters for State Estimation of Jump Markov Linear Systems. Technical report, Cambridge Univ. Engineering Dept., 1999.
- [DK89] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Artificial Intelligence*, 93(1–2):1–27, 1989.
- [DK01] J. Durbin and S. J. Koopman. *Time Series Analysis by State Space Methods*. Oxford University Press, 2001.
- [DNCDW01] M. G. Dissanayake, P. Newman, S. Clark, and H. F. Durrant-Whyte. A Solution to the Simultaneous Localization and Map Building (SLAM) Problem. *IEEE Trans. Robotics and Automation*, 17(3):229–241, 2001.
- [Dou98] A Doucet. On sequential simulation-based methods for Bayesian filtering. Technical report CUED/F-INFENG/TR 310, Department of Engineering, Cambridge University, 1998.
- [FC89] R. Fung and K. Chang. Weighting and integrating evidence for stochastic simulation in Bayesian networks. In *UAI*, 1989.
- [FST98] S. Fine, Y. Singer, and N. Tishby. The hierarchical Hidden Markov Model: Analysis and applications. *Machine Learning*, 32:41, 1998.
- [Gal99] M. J. F. Gales. Semi-tied covariance matrices for hidden Markov models. *IEEE Trans. on Speech and Audio Processing*, 7(3):272–281, 1999.
- [GDW00] S. Godsill, A. Doucet, and M. West. Methodology for Monte Carlo Smoothing with Application to Time-Varying Autoregressions. In *Proc. Intl. Symp. on Frontiers of Time Series Modelling*, 2000.
- [GH98] Z. Ghahramani and G. Hinton. Variational learning for switching state-space models. *Neural Computation*, 12(4):963–996, 1998.
- [GJ97] Z. Ghahramani and M. Jordan. Factorial hidden Markov models. *Machine Learning*, 29:245–273, 1997.
- [Goo01] J. Goodman. A bit of progress in language modelling. *Computer Speech and Language*, pages 403–434, October 2001.
- [Gre98] P. Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82:711–732, 1998.
- [HZ02] T. Heskes and O. Zoeter. Expectation propagation for approximate inference in dynamic Bayesian networks. In *UAI*, 2002.

- [IB98] M. Isard and A. Blake. A smoothing filter for condensation. In *Proc. European Conf. on Computer Vision*, volume 1, pages 767–781, 1998.
- [Jel97] F. Jelinek. *Statistical methods for speech recognition*. MIT Press, 1997.
- [JKOP89] F. V. Jensen, U. Kjaerulff, K. G. Olesen, and J. Pedersen. An expert system for control of waste water treatment — a pilot project. Technical report, Univ. Aalborg, Judex Datasystemer, 1989. In Danish.
- [JU97] S. Julier and J. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In *Proc. of AeroSense: The 11th Intl. Symp. on Aerospace/Defence Sensing, Simulation and Controls*, 1997.
- [KCB00] J. Kwon, B. Coifman, and P. Bickel. Day-to-day travel time trends and travel time prediction from loop detector data. *Transportation Research Record*, 1554, 2000.
- [Kim94] C-J. Kim. Dynamic linear models with Markov-switching. *J. of Econometrics*, 60:1–22, 1994.
- [Kja92] U. Kjaerulff. A computational scheme for reasoning in dynamic probabilistic networks. In *UAI-8*, 1992.
- [Kja95] U. Kjaerulff. dHugin: A computational system for dynamic time-sliced Bayesian networks. *Intl. J. of Forecasting*, 11:89–111, 1995.
- [KKR95] K. Kanazawa, D. Koller, and S. Russell. Stochastic simulation algorithms for dynamic probabilistic networks. In *UAI*, 1995.
- [KL01] D. Koller and U. Lerner. Sampling in Factored Dynamic Systems. In A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*. Springer, 2001.
- [KN98] C-J. Kim and C. Nelson. *State-Space Models with Regime-Switching: Classical and Gibbs-Sampling Approaches with Applications*. MIT Press, 1998.
- [LC98] J. Liu and R. Chen. Sequential Monte Carlo methods for dynamic systems. *JASA*, 93:1032–1044, 1998.
- [LP01] U. Lerner and R. Parr. Inference in hybrid networks: Theoretical limits and practical algorithms. In *UAI*, 2001.
- [McC95] A. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Univ. Rochester, 1995.
- [MCH02] Christopher Meek, David Maxwell Chickering, and David Heckerman. Autoregressive tree models for time-series analysis. In *Proceedings of the Second International SIAM Conference on Data Mining*, pages 229–244, Arlington, VA, April 2002. SIAM.
- [Min01] T. Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, MIT, 2001.
- [Min02] T. Minka. Bayesian inference in dynamic models: an overview. Technical report, CMU, 2002.
- [Moh96] M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 20(1):1–33, 1996.
- [MP01] K. Murphy and M. Paskin. Linear time inference in hierarchical HMMs. In *NIPS*, 2001.
- [MTKW02] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In *AAAI*, 2002.
- [Mur98] K. P. Murphy. Switching Kalman filters. Technical report, DEC/Compaq Cambridge Research Labs, 1998.
- [Mur00] K. P. Murphy. Bayesian map learning in dynamic environments. In *NIPS-12*, pages 1015–1021, 2000.

- [Mur02] K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, Dept. Computer Science, UC Berkeley, 2002.
- [MW01] K. Murphy and Y. Weiss. The Factored Frontier Algorithm for Approximate Inference in DBNs. In *UAI*, 2001.
- [NG01] D. Nilsson and J. Goldberger. Sequentially finding the N-Best List in Hidden Markov Models. In *IJCAI*, pages 1280–1285, 2001.
- [Nil98] D. Nilsson. An efficient algorithm for finding the M most probable configurations in a probabilistic expert system. *Statistics and Computing*, 8:159–173, 1998.
- [NLP⁺02] A. Nefian, L. Liang, X. Pi, X. Liu, and K. Murphy. Dynamic Bayesian Networks for Audio-Visual Speech Recognition. *J. Applied Signal Processing*, 2002.
- [NPP02] B. Ng, L. Peshkin, and A. Pfeffer. Factored particles for scalable monitoring. In *UAI*, 2002.
- [NY00] H. J. Nock and S. J. Young. Loosely coupled HMMs for ASR. In *Intl. Conf. on Acoustics, Speech and Signal Proc.*, 2000.
- [ODK96] M. Ostendorf, V. Digalakis, and O. Kimball. From HMMs to segment models: a unified view of stochastic modeling for speech recognition. *IEEE Trans. on Speech and Audio Processing*, 4(5):360–378, 1996.
- [PR97] F. Pereira and M. Riley. Speech recognition by composition of weighted finite automata. In *Finite-State Language Processing*, pages 431–453. MIT Press, 1997.
- [Rab89] L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proc. of the IEEE*, 77(2):257–286, 1989.
- [RN02] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002. 2nd edition, in preparation.
- [RST96] Dana Ron, Yoram Singer, and Naftali Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25, 1996.
- [RTL76] D. Rose, R. Tarjan, and G. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. on Computing*, 5:266–283, 1976.
- [Sha86] R. Shachter. Evaluating influence diagrams. *Operations Research*, 33(6):871–882, 1986.
- [SJ95] L. Saul and M. Jordan. Boltzmann chains and hidden Markov models. In *NIPS-7*, 1995.
- [SJ99] L. Saul and M. Jordan. Mixed memory markov models: Decomposing complex stochastic processes as mixture of simpler ones. *Machine Learning*, 37(1):75–87, 1999.
- [SM80] A. Smith and U. Makov. Bayesian detection and estimation of jumps in linear systems. In O. Jacobs, M. Davis, M. Dempster, C. Harris, and P. Parks, editors, *Analysis and optimization of stochastic systems*. 1980.
- [SP89] R. D. Shachter and M. A. Peot. Simulation approaches to general probabilistic inference on belief networks. In *UAI*, volume 5, 1989.
- [TDW02] M. Takikawa, B. D’Ambrosio, and E. Wright. Real-time inference with large-scale temporal bayes nets. In *UAI*, 2002.
- [TRM01] G. Theocharous, K. Rohanimanesh, and S. Mahadevan. Learning Hierarchical Partially Observed Markov Decision Process Models for Robot Navigation. In *ICRA*, 2001.
- [TSM85] D. M. Titterington, A. F. M. Smith, and U. E. Makov. *Statistical analysis of finite mixture distributions*. Wiley, 1985.

- [vdMDdFW00] R. van der Merwe, A. Doucet, N. de Freitas, and E. Wan. The unscented particle filter. In *NIPS-13*, 2000.
- [WdM01] E. A. Wan and R. Van der Merwe. The Unscented Kalman Filter. In S. Haykin, editor, *Kalman Filtering and Neural Networks*. Wiley, 2001.
- [WH97] Mike West and Jeff Harrison. *Bayesian forecasting and dynamic models*. Springer, 1997.
- [ZP00] G. Zweig and M. Padmanabhan. Exact alpha-beta computation in logarithmic space with application to map word graph construction. In *Proc. Intl. Conf. Spoken Lang. Proc.*, 2000.
- [Zwe96] G. Zweig. A forward-backward algorithm for inference in Bayesian networks and an empirical comparison with HMMs. Master's thesis, Dept. Comp. Sci., U.C. Berkeley, 1996.