Probabilistic Graphical Models

Lecture notes and exercises

Hao Zhu
Joschka Boedecker

Department of Computer Science
University of Freiburg

universität freiburg

Revised: 2024.04; **Run:** June 7, 2024

Preface

These are lecture notes and exercises for the course ‘Probabilistic Graphical Models’ at the University of Freiburg, starting from Summer 2024. There are many people who we should like to thank for comments on and corrections to the notes, but for the moment we would simply like to thank, as a collective, the students at the University of Freiburg who have made this course a joy to teach, as a result of their interest and enthusiasm.

Contents

Preface iii

Contents v

1 Introduction 1
1.1 Mathematical background 1
1.1.1 Probability theory 1
1.1.2 Graphs 4
1.2 Notation 5
Bibliography 6
Exercises 7

I Probabilistic models 9

2 Bayesian classifiers 11
2.1 Probabilistic classification and Bayesian classifiers 11
2.1.1 Naive Bayesian classifiers 11
2.1.2 Augmented Bayesian classifiers 13
2.1.3 Semi-naive Bayesian classifiers 14
2.2 Multi-dimensional classification 14
2.2.1 Basic approaches 15
2.2.2 Chain classifiers 15
Bibliography 17
Exercises 18

3 Markov models 19
3.1 Markov chains 19
3.1.1 Inference and parameter learning 20
3.1.2 Convergence 21
3.2 Hidden Markov models 22
3.2.1 Inference 23
3.2.2 Decoding 24
3.2.3 Parameter learning 26
3.2.4 Continuous observation space 28
Bibliography 29
Exercises 30

4 Bayesian networks 31
4.1 Representation 31
4.1.1 Structure 31
4.1.2 Parameters 33
4.2 Inference 35
4.2.1 Belief propagation 35
4.2.2 Variable elimination 39
4.2.3 Conditioning 40
4.2.4 Junction tree algorithm 42

4.2.5 Sampling based methods . . . 46
4.3 Parameter learning . . . 46
4.3.1 Smoothing . . . 47
4.3.2 Missing data . . . 48
4.3.3 Discretization . . . 49
4.4 Structure learning . . . 49
4.4.1 Tree learning . . . 49
4.4.2 Score-based methods . . . 51
4.4.3 PC algorithm . . . 52
Bibliography . . . 53
Exercises . . . 55

II Decision models 57

5 Markov decision problems 59
5.1 Markov decision processes . . . 59
5.1.1 Episodes and returns . . . 59
5.1.2 Value functions . . . 60
5.1.3 Optimal value functions and policies . . . 61
5.1.4 Example . . . 62
5.2 Dynamic programming . . . 63
5.2.1 Policy iteration . . . 63
5.2.2 Value iteration . . . 65
Bibliography . . . 66
Exercises . . . 67

6 Control as probabilistic inference 69
6.1 Exact inference . . . 69
6.1.1 The graphical model and policy search . . . 69
6.1.2 Connection to Bellman equations . . . 71
6.2 Approximate inference . . . 71
6.2.1 Maximum entropy control . . . 71
6.2.2 Connection to variational inference . . . 72
6.2.3 Obtaining the optimal policy . . . 73
Bibliography . . . 75

Appendices 77

A Inference with Monte Carlo methods 79
A.1 Monte Carlo methods . . . 79
A.1.1 Monte Carlo integration . . . 79
A.1.2 Sampling from simple distributions . . . 80
A.1.3 Rejection sampling . . . 81
A.1.4 Importance sampling . . . 82
A.2 Markov chain Monte Carlo . . . 83
A.2.1 Metropolis-Hastings algorithm . . . 84
A.2.2 Gibbs sampling . . . 87
A.2.3 Hamiltonian Monte Carlo . . . 89
Bibliography . . . 92
Exercises . . . 94

References 95

Notation 101

Chapter 1

Introduction

1.1 Mathematical background

1.1.1 Probability theory

Basic concepts

In this course we will adhere to the Bayesian interpretation of probability, according to which probabilities encode degrees of belief about events in the world and data are used to strengthen, update, or weaken those degrees of belief. For example, if A stands for an event, then $\mathbf{P}(A \mid K)$ stands for a person's subjective belief in event A given a body of knowledge K. In defining probability expressions, we often simply write $\mathbf{P}(A)$, leaving out the symbol K. However, when the background information undergoes changes, we need to identify specifically the assumptions that account for our beliefs and explicitly articulate K (or some of its elements).

In the Bayesian formalism, belief measures obey the three basic axioms of probability calculus:

- $0 \leq \mathbf{P}(A) \leq 1$,
- $\mathbf{P}(\text{sure proposition}) = 1$,
- $\mathbf{P}(A \text{ or } B) = \mathbf{P}(A) + \mathbf{P}(B)$ if A and B are mutually exclusive.

The third axiom states that the belief assigned to any set of events is the sum of the beliefs assigned to its nonintersecting components. Because any event A can be written as the union of the joint events $(A \wedge B)$ and $(A \wedge \neg B)$, their associated probabilities are given by

$$\mathbf{P}(A) = \mathbf{P}(A, B) + \mathbf{P}(A, \neg B), \tag{1.1}$$

where $\mathbf{P}(A, B)$ is short for $\mathbf{P}(A \wedge B)$. More generally, if $\{B_1, \ldots, B_n\}$ is a set of exhaustive and mutually exclusive propositions, then $\mathbf{P}(A)$ can be computed from $\mathbf{P}(A, B_i), i = 1, \ldots, n$, by using the sum

$$\mathbf{P}(A) = \sum_{i=1}^{n} \mathbf{P}(A, B_i), \tag{1.2}$$

which has come to be known as the *law of total probability.* The operation of summing up probabilities over all B_i is also called *marginalizing* over B; and the resulting probability, $\mathbf{P}(A)$, is called the *marginal probability* of A.

The basic expressions in the Bayesian formalism are statements about *conditional probabilities* — for example, $\mathbf{P}(A \mid B)$, which specify the belief in A under the assumption that B is known with absolute certainty. If $\mathbf{P}(A \mid B) = \mathbf{P}(A)$, we say that A and B are *independent*, since our belief in A remains unchanged upon learning the truth of B. If $\mathbf{P}(A \mid B, C) = \mathbf{P}(A \mid C)$, we say that A and B are *conditionally independent* given C; that is, once we know C, learning B would not change our belief in A. Bayesian philosophers see the conditional relationship as more basic than that of joint events — that is, more compatible with the organization of human knowledge. In this view, B serves as a pointer to a context or frame of knowledge, and $A \mid B$ stands for an event A in the context specified by B. Consequently, empirical knowledge invariably will be encoded in conditional probability statements, whereas belief in joint events will be computed from those statements via the product

$$\mathbf{P}(A, B) = \mathbf{P}(A \mid B)\mathbf{P}(B). \tag{1.3}$$

A useful generalization of the product rule (1.3) is the *chain rule* formula. It states that if we have a set of n events, $E_1, \ldots, E_n$, then the probability of the joint event $(E_1, \ldots, E_n)$ can be written as a product of n conditional probabilities:

$$\mathbf{P}(E_1, \ldots, E_n) = \mathbf{P}(E_1)\mathbf{P}(E_2 \mid E_1) \cdots \mathbf{P}(E_n \mid E_{n-1}, \ldots, E_1). \tag{1.4}$$

The heart of Bayesian inference lies in the celebrated inversion formula — the *Bayes' theorem*

$$\mathbf{P}(H \mid e) = \frac{\mathbf{P}(e \mid H)\mathbf{P}(H)}{\mathbf{P}(e)}, \tag{1.5}$$

which states that the belief we accord a hypothesis H upon obtaining evidence e can be computed by multiplying our previous belief $\mathbf{P}(H)$ by the *likelihood* $\mathbf{P}(e \mid H)$ that e will materialize if H is true. This $\mathbf{P}(H \mid e)$ is sometimes called the *posterior probability*, and $\mathbf{P}(H)$ is called the *prior probability*. The denominator $\mathbf{P}(e)$ of (1.5) hardly enters into consideration because it is merely a normalizing constant.

Random variables and expectations

By a *variable* we will mean an attribute, measurement or inquiry that may take on one of several possible values from a specified domain. If we have probabilities attached to the possible values that a variable may attain, we will call that variable a *random variable.* Most of our analysis will concern a finite set V of random variables (also called *partitions*) where each variable $X \in V$ may take on values from a finite *domain* $\mathbf{dom}(X)$. We will use capital letters X, Y, Z for variable names and lowercase letters x, y, z as generic symbols for specific values taken by the corresponding variables. Clearly, the statement $X = x$ defines a set of exhaustive and mutually exclusive events, one for each value of x.

In most of our discussions, we will not make notational distinction between variables and sets of variables, because a set of variables essentially defines a compound variable whose domain is the Cartesian product of the domains of the individual constituents in the set. Thus, if Z stands for the set $\{X, Y\}$, then z stands for pairs (x, y) such that $x \in \mathbf{dom}(X)$ and $y \in \mathbf{dom}(Y)$. When the distinction between variables and sets of variables requires special emphasis, indexed letters $X_1, \ldots, X_n$ will be used to represent individual variables. Besides, we shall consistently use the abbreviation $\mathbf{P}(x)$ for the probabilities $\mathbf{P}(X = x)$, $x \in \mathbf{dom}(X)$.

When the values of a random variable X are real numbers, i.e., $x \in \mathbf{R}$, X is called a *real* random variable; one can then define the *mean* or *expected value* of X as

$$\mathbf{E}[X] = \sum_x x\mathbf{P}(x) \tag{1.6}$$

and the conditional mean of X, given event $Y = y$, as

$$\mathbf{E}[X \mid y] = \sum_x x\mathbf{P}(x \mid y). \tag{1.7}$$

The expectation of any function g of X is defined as

$$\mathbf{E}[g(X)] = \sum_x g(x)\mathbf{P}(x). \tag{1.8}$$

In particular, the function $g(X) = (X - \mathbf{E}[X])^2$ has received much attention; its expectation is called the *variance* of X, denoted $\mathbf{var}(X)$;

$$\mathbf{var}(X) = \mathbf{E}\left[(X - \mathbf{E}[X])^2\right]. \tag{1.9}$$

We are often interested in the square root of the variance $\mathbf{var}(X)$, which is called the *standard deviation* of the random variable X, denoted as

$$\sigma(X) = \sqrt{\mathbf{var}(X)}. \tag{1.10}$$

When function g is a two-variable function with $g(X, Y) = (X - \mathbf{E}[X])(Y - \mathbf{E}[Y])$, its expectation is known as the *covariance* of X and Y,

$$\mathbf{cov}(X, Y) = \mathbf{E}[(X - \mathbf{E}[X])(Y - \mathbf{E}[Y])], \tag{1.11}$$

and which is often normalized to yield the *correlation coefficient*

$$\rho(X, Y) = \frac{\mathbf{cov}(X, Y)}{\sigma(X)\sigma(Y)} \tag{1.12}$$

and the *regression coefficient* (of X on Y)

$$r(X, Y) = \rho(X, Y)\frac{\sigma(X)}{\sigma(Y)} = \frac{\mathbf{cov}(X, Y)}{\mathbf{var}(Y)}. \tag{1.13}$$

The foregoing definitions apply to discrete random variables — that is, variables that take on finite or denumerable sets of values on $\mathbf{R}$. The treatment of expectation and correlation is more often applied to continuous random variables, which are characterized by a *density function* $p(x)$ defined as follows:

$$\mathbf{P}(a \leq X \leq b) = \int_a^b p(x)\ dx \tag{1.14}$$

for any $a, b \in \mathbf{R}, a < b$. If X is discrete, then $p(x)$ coincides with the probability function $\mathbf{P}(x)$, once we interpret the integral through the translation

$$\int_{-\infty}^{\infty} p(x)\ dx \Longleftrightarrow \sum_x \mathbf{P}(x). \tag{1.15}$$

This translation should be kept in mind whenever summation is used through the course. For example, the expected value of a continuous random variable X can be transformed from (1.6) to

$$\mathbf{E}[X] = \int_{-\infty}^{\infty} xp(x)\ dx, \tag{1.16}$$

with analogous translations for the variance, correlation, and so forth.

Example 1.1 *Gaussian distribution.* A random variable X has a Gaussian distribution with mean μ and variance σ^2, denoted $\mathcal{N}(\mu, \sigma^2)$, if it has the density function

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{1.17}$$

A Gaussian distribution has a bell-like curve, where the mean parameter μ controls the location of the peak, that is, the value for which the Gaussian gets its maximum value. The variance parameter σ^2 determines how peaked the Gaussian is: the smaller the variance, the more peaked the Gaussian. A standard Gaussian is one with mean 0 and variance 1. Figure 1.1 shows the density function of a few different Gaussian distributions.

More technically, the density function of Gaussian distribution is specified as an exponential, where the expression in the exponent corresponds to the square of the number of standard deviations σ that x is away from the mean μ. The probability of x decreases exponentially with the square of its deviation from the mean, as measured in units of its standard deviation.

Set conditional independence and graphoids

Let $V = \{V_1, V_2, \ldots\}$ be a finite set of variables and let X, Y, Z stand for any three subsets of variables in V. The sets X and Y are said to be *conditionally independent* given Z if and only if $\mathbf{P}(x \mid y, z) = \mathbf{P}(x \mid z)$ for all y, z that $\mathbf{P}(y, z) > 0$ holds. In words, learning the value of Y does not provide additional information about X, once we know Z. We will use the notation $(X \perp\!\!\!\perp Y \mid Z)$ to denote the conditional independence of X and Y given Z. Unconditional independence (also called *marginal independence*) will be denoted by $(X \perp\!\!\!\perp Y \mid \emptyset)$, which says $\mathbf{P}(x \mid y) = \mathbf{P}(x)$ for all y that $\mathbf{P}(y) > 0$ holds. Note that $(X \perp\!\!\!\perp Y \mid Z)$ implies the conditional

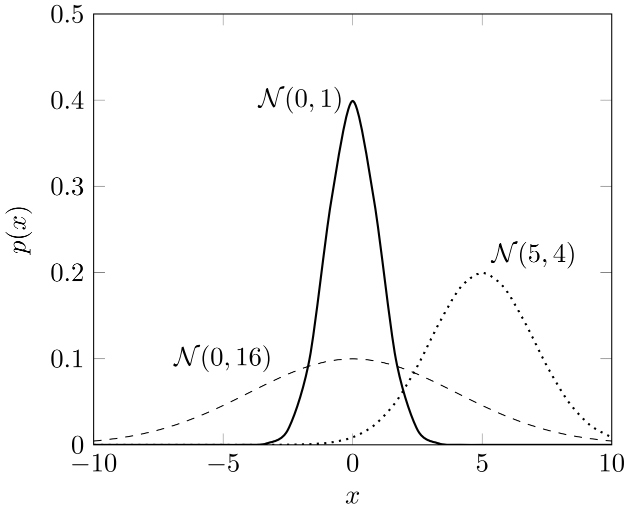


Figure 1.1 Density function of three example Gaussian distributions.

independence of all pairs of variables $V_i \in X$ and $V_j \in Y$, but the converse is not necessarily true.

In the following we list some properties satisfied by the conditional independence relation $(X \perp\!\!\!\perp Y \mid Z)$:

- *Symmetry*: $(X \perp\!\!\!\perp Y \mid Z) \implies (Y \perp\!\!\!\perp X \mid Z)$.
- *Decomposition*: $(X \perp\!\!\!\perp YW \mid Z) \implies (X \perp\!\!\!\perp Y \mid Z)$.
- *Weak union*: $(X \perp\!\!\!\perp YW \mid Z) \implies (X \perp\!\!\!\perp Y \mid ZW)$.
- *Contraction*: $(X \perp\!\!\!\perp Y \mid Z)$ & $(X \perp\!\!\!\perp W \mid ZY) \implies (X \perp\!\!\!\perp YW \mid Z)$.
- *Intersection*[1]: $(X \perp\!\!\!\perp W \mid ZY)$ & $(X \perp\!\!\!\perp Y \mid ZW) \implies (X \perp\!\!\!\perp YW \mid Z)$.

These properties are called *graphoid axioms* and the proof of them can be derived from the definition of conditional independence and the basic axioms of probability theory.

1.1.2 Graphs

A graph G consists of a set V of *vertices* (or *nodes*) and a set E of *edges* (or *links*) that connect some pairs of vertices. The vertices in our graphs will correspond to variables, and the edges will denote a certain relationship that holds in pairs of variables, the interpretation of which will vary with the application. Two vertices connected by an edge are called *adjacent*.

Each edge in a graph can be either directed (marked by a single arrowhead on the edge), or undirected (unmarked links). In some applications we will also use 'bidirected' edges to denote the existence of unobserved common causes (sometimes called *confounders*). These edges will be marked as dotted curved arcs with two arrowheads as shown in Figure 1.2(a). If all edges are directed (as in Figure 1.2(b)), we then have a *directed graph*. If we strip away all arrowheads from the edges in a graph G, the resultant undirected graph is called the *skeleton* of G. A *path* in a graph is a sequence of edges (e.g., $((W,Z),(Z,Y),(Y,X),(X,Z))$ in Figure 1.2(a)) such that each edge starts with the vertex ending the preceding edge. In other words, a path is any unbroken, nonintersecting route traced out along the edges in a graph, which may go either along or against the arrows. If every edge in a path is an arrow that points from the first to the second vertex of the pair, we have a *directed path*. In Figure 1.2(a), for example, the path $((W,Z),(Z,Y))$ is directed, but the paths $((W,Z),(Z,Y),(Y,X))$ and $((W,Z),(Z,X))$ are not. If there exists a path between two vertices in a graph, then the two vertices are said to be *connected*; else they are *disconnected*.

Directed graphs may include directed cycles (e.g., $X \to Y, Y \to X$), representing mutual causation or feedback processed, but not self-loops (e.g., $X \to X$). A graph (like the two in

[1] Intersection is valid in strictly positive probability distributions.

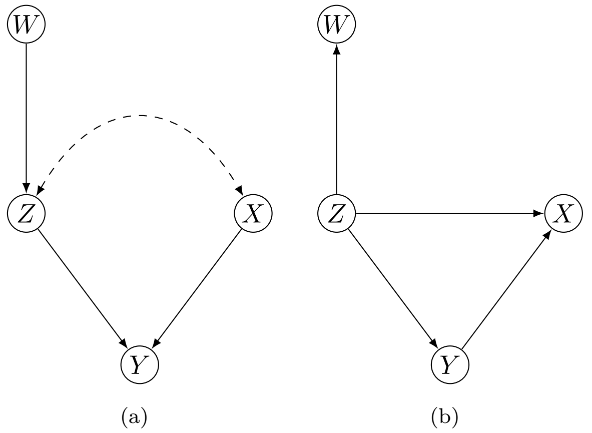


Figure 1.2 (a) A graph containing both directed and bidirected edges. (b) A directed acyclic graph with the same skeleton as (a).

Figure 1.2) that contains no directed cycles in called *acyclic*. A graph that is both directed and acyclic (Figure 1.2(b)) is called a *directed acyclic graph* (DAG), and such graphs will occupy much of our discussion through the course. We make free use of the terminology of kinship (e.g., *parents*, *children*, *descendants*, *ancestors*, *spouses*) to denote various relationships in a graph. These kinship relations are defined along the full arrows in the graph, including arrows that form directed cycles but ignoring bidirected and undirected edges. In Figure 1.2(a), for example, Y has two parents (X and Z), three ancestors (X, Z, and W), and no children, while X has no parents (hence, no ancestors), one spouse (Z), and one child (Y). A family in a graph is a set of nodes containing a node and all its parents. For example, $\{W\}$, $\{Z, W\}$, $\{X\}$, and $\{Y, Z, X\}$ are the families in the graph of Figure 1.2(a).

A node in a directed graph is called a *root* if it has no parents and a *sink* if it has no children. Every DAG has at least one root and at least one sink. A connected DAG in which every node has at most one parent is called a *tree*, and a tree in which every node has at most one child is called a *chain*. A graph in which every pair of nodes is connected by an edge is called *complete*. The graph in Figure 1.2(a), for instance, is connected but not complete, because the pairs (W, X) and (W, Y) are note adjacent.

1.2 Notation

Our notation is more or less standard, with a few exceptions. In this section we describe some additional basic notation except those introduced in the previous sections; a more complete list appears on page 101.

We use $\mathbf{R}$ to denote the set of real numbers, $\mathbf{R}_+$ to denote the set of nonnegative real numbers, and $\mathbf{R}_{++}$ to denote the set of positive real numbers. the set of real n-vectors is denoted $\mathbf{R}^n$, and the set of real $m \times n$ matrices is denoted $\mathbf{R}^{m \times n}$. The symbol $\mathbf{1}$ denotes a vector all of whose components are one (with dimension determined from context).

We use the notation $f\colon \mathbf{R}^p \to \mathbf{R}^q$ to mean that f is an $\mathbf{R}^q$ valued function on some subset of $\mathbf{R}^p$, specifically, its domain, which we denote $\mathbf{dom}(f)$. We can think of our use of the notation $f\colon \mathbf{R}^p \to \mathbf{R}^q$ as a declaration of the function *type*, as in a computer language: $f\colon \mathbf{R}^p \to \mathbf{R}^q$ means that the function f takes as argument a real p-vector, and returns a real q-vector. The set $\mathbf{dom}(f)$, the domain of the function f, specifies the subset of $\mathbf{R}^p$ of points x for which $f(x)$ is defined. As an example, we describe the logarithm function as $\log\colon \mathbf{R} \to \mathbf{R}$, with $\mathbf{dom}(\log) = \mathbf{R}_{++}$. The notation $\log\colon \mathbf{R} \to \mathbf{R}$ means that the logarithm function accepts and returns a real number; $\mathbf{dom}(\log) = \mathbf{R}_{++}$ means that the logarithm is defined only for positive numbers.

Bibliography

Our introduction about probability theory is mostly based on [Pea09, §1.1], with some additional information from [KF09, §2.1]. Many excellent textbooks on the subject, e.g., [Fel50, HPS71], or the appendix to [Sup70], can be referred to for additional mathematical machinery in probability.

The basic probability axioms introduced at the beginning of this chapter deviate a bit from the standard statement, which can be found in the textbook [KBR18].

The graphoid axioms listed in this chapter were first introduced in [Daw79] and [Spo80] in a slightly different form, and were independently proposed by Pearl and Paz [PP87] to characterize the relationships between graphs and informational relevance. Geiger and Pearl [GP93] present an in-depth analysis. The intuitive interpretation of the graphoid axioms is discussed in [Pea88, p. 85].

Our brief discussion about the terminology of graphs is adapted from [Pea09, §1.2.1]. A more detailed introduction on graphs related to the topic of probabilistic graphical models can be found in [KF09]. The excellent textbook [Wes01] specifically for graph theory can be referred to for additional mathematical machinery.

The notation introduced in this chapter and will be used in the following chapters are mostly based on [BV04]. Some notation related to probability are taken from those used in [Pea09] and [KF09].

Exercises

1.1 The table below shows the joint distribution $\mathbf{P}(X = x, Y = y)$ of random variables X and Y.

	y_1	y_2	y_3
x_1	0.1	0.2	0.1
x_2	0.3	0.1	0.2

(a) Calculate $\mathbf{P}(X = x_1)$, $\mathbf{P}(Y = y_2)$, and $\mathbf{P}(X = x_1 \mid Y = y_1)$.

(b) Are random variables X and Y independent? If not, modify the probabilities of the joint distribution so that X and Y are independent.

1.2 *Markov inequality.* Let X be a non-negative continuous random variable ($\mathbf{dom}(X) = \mathbf{R}_+$), show that for any $t > 0$ the following inequality holds:

$$\mathbf{P}(X \geq t) \leq \frac{\mathbf{E}[X]}{t}.$$

Part I

Probabilistic models

Chapter 2

Bayesian classifiers

2.1 Probabilistic classification and Bayesian classifiers

Given a set of samples X and a set of class labels Y whose entries are normally considered as nonnegative integers, i.e., $\mathbf{dom}(Y) \subseteq \mathbf{Z}_+$. An 'ordinary' classifier is some function $f\colon X \to Y$ that assigns each sample x a class label $\hat{y}$. Probabilistic classifiers, instead of providing a deterministic prediction on class label, predict the posterior probability $\mathbf{P}(y \mid x)$ for all $y \in Y$ given $x \in X$, and these posterior probabilities satisfy

$$\sum_y \mathbf{P}(y \mid x) = 1,$$

for all $x \in X$. The similar 'hard' classification as in ordinary classifiers can then be calculated by

$$\hat{y} = \operatorname*{argmax}_y \mathbf{P}(y \mid x).$$

The formulation of the *Bayesian classifier* is based on the application of the Bayes rule to estimate the posterior probability of each class given the sample $x \in X$:

$$\mathbf{P}(y \mid x) = \frac{\mathbf{P}(x \mid y)\mathbf{P}(y)}{\mathbf{P}(x)}. \tag{2.1}$$

Note that the denominator $\mathbf{P}(x)$ is just a normalizing constant and is barely taken into account in practice. The probability $\mathbf{P}(y)$ is known as the prior probability of the class labels. Therefore, to estimate the posterior probability we only need to calculate the likelihood $\mathbf{P}(x \mid y)$ according to the given data. Assuming that x is represented as a n-dimensional vector $(x_1, \ldots, x_n)$, with each entry being a random variable X_i denoting the ith feature of sample x. Then according to the chain rule (1.4), the likelihood term can be calculated as

$$\begin{aligned} \mathbf{P}(x \mid y) &= \mathbf{P}(x_1, \ldots, x_n \mid y) \\ &= \mathbf{P}(x_1 \mid y)\mathbf{P}(x_2 \mid x_1, y) \cdots \mathbf{P}(x_n \mid x_{n-1}, \ldots, x_1, y). \end{aligned} \tag{2.2}$$

Calculating (2.2) can be computationally expensive when n is large since the number of parameters in the likelihood term increase exponentially with the dimension of x. This will not only imply a huge amount of memory to store all the parameters, but it will also be very difficult to estimate all the probabilities from the data. Thus, the Bayesian classifier can only be of practical use for relatively small problems in terms of the dimension of the feature vector of x. An alternative is to consider some independence properties as in graphical models, in particular that all features are independent given the class, resulting in the *naive Bayesian classifier*.

2.1.1 Naive Bayesian classifiers

The *naive Bayesian classifier* is based on the assumption that all the features of sample x are independent given the class variable; that is, $\mathbf{P}(x_i \mid x_j, y) = \mathbf{P}(x_i \mid y)$ for all $j \neq i$. Under this assumption, (2.2) can be written as:

$$\mathbf{P}(x \mid y) = \mathbf{P}(x_1, \ldots, x_n \mid y) = \prod_{i=1}^{n} \mathbf{P}(x_i \mid y). \tag{2.3}$$

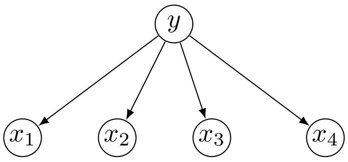


Figure 2.1 Example of a naive Bayesian classifier.

Substituting the likelihood term (2.3) under the naive Bayesian classifier assumption into (2.1), the posterior probability of class label y given sample x is

$$\mathbf{P}(y \mid x) = \frac{1}{Z}\mathbf{P}(y)\prod_{i=1}^{n}\mathbf{P}(x_i \mid y), \tag{2.4}$$

where Z is the normalization factor with

$$Z = \mathbf{P}(x) = \sum_{y}\mathbf{P}(y)\mathbf{P}(x \mid y) \tag{2.5}$$

so that the estimated posterior probabilities satisfy $\sum_y \mathbf{P}(y \mid x) = 1$.

The naive Bayes formulation drastically reduces the complexity of the Bayesian classifier, since in this case as the parameters for the model, we only require the prior probability of the class,

$$(\mathbf{P}(y_1), \mathbf{P}(y_2), \ldots), \tag{2.6}$$

which is a one dimensional vector with the dimension equals to the cardinality of Y; and the n conditional probabilities of each feature of sample x given the class y,

$$\begin{bmatrix} \begin{bmatrix} \mathbf{P}(x_1 \mid y_1) \\ \vdots \\ \mathbf{P}(x_n \mid y_1) \end{bmatrix} & \begin{bmatrix} \mathbf{P}(x_1 \mid y_2) \\ \vdots \\ \mathbf{P}(x_n \mid y_2) \end{bmatrix} & \cdots \end{bmatrix}, \tag{2.7}$$

which is a two dimensional matrix with the dimension of $n \times \mathbf{card}(Y)$. That is, the space requirement is reduced from exponential to linear in the number of features. A graphical representation of the naive Bayesian classifier is shown in Figure 2.1. This tree-like structure depicts the property of conditional independence between all the features given the class — as there are no arcs between the feature nodes.

To learn the parameters of a naive Bayesian classifier, the prior probabilities (2.6) of the class variable Y can either be considered as uniformly distributed, i.e.,

$$\mathbf{P}(y_i) = \frac{1}{\mathbf{card}(Y)}, \quad \text{for all } y_i \in Y,$$

or be determined by estimating the class probability from the training data:

$$\mathbf{P}(y_i) = \frac{\#\text{ samples in class } y_i}{\#\text{ samples in total}}, \quad \text{for all } y_i \in Y.$$

If all features of each sample $x \in X$ are represented with discrete random variables, each entry of the likelihood matrix (2.7) can be directly learnt from the given data by calculating:

$$\mathbf{P}(x_k \mid y_i) = \frac{\#\text{ samples in class } y_i \text{ with feature } x_k}{\#\text{ samples in class } y_i},$$

for all $k = 1, \ldots, n$, $y_i \in Y$. As for handling continuous features, one trivial way would be using binning to discretize the feature values, so that the likelihood term can still be learnt using the above equations. However, the discretization may throw away discriminative information from the data. Another common technique for handling continuous values is to assume a parameterized distribution for the features from the training dataset, such as Gaussian distribution, multinomial distribution, or Bernoulli distribution, so that the learning of the likelihood matrix (2.7) can be transformed to learning the parameters of the assumed distribution.

Example 2.1 *Gaussian naive Bayes.* When dealing with continuous featured samples, a typical assumption is that within each class, the values of each continuous feature are Gaussian distributed, i.e.,

$$p(x_k \mid y_i) = \frac{1}{\sqrt{2\pi}\sigma_{k|y_i}} \exp\left(-\frac{(x_k - \mu_{k|y_i})^2}{2\sigma^2_{k|y_i}}\right), \quad k = 1, \dots, n,$$

for all $y_i \in Y$, where $\mu_{k|y_i}$ and $\sigma^2_{k|y_i}$ are the mean and variance of the distribution of feature x_k within class y_i, respectively. Then to estimate the likelihood term required for naive Bayesian classifiers, we only need to learn $\mu_{k|y_i}$ and $\sigma_{k|y_i}$ from the training data according to

$$\mu_{k|y_i} = \mathbf{E}[X_k \mid y_i], \quad k = 1, \dots, n,$$

$$\sigma_{k|y_i} = \sqrt{\mathbf{var}(X_k \mid y_i)} = \sqrt{\mathbf{E}\left[(X_k - \mathbf{E}[X_k \mid y_i])^2 \,\middle|\, y_i\right]}, \quad k = 1, \dots, n,$$

for all $y_i \in Y$.

In the inference stage, the density function of the assumed distribution evaluated at feature x_k for class y_i, i.e., $p(x_k \mid y_i)$, is used to provide a relative estimation of each likelihood in matrix (2.7).

2.1.2 Augmented Bayesian classifiers

The general Bayesian classifier and the naive Bayesian classifier are the two extremes of possible dependency structures for Bayesian classifiers; the former represents the most complex structure with no independence assumptions, while the latter is the simplest structure that assumes that all the features are independent given the class. Between these two extremes there is a wide variety of possible models of varying complexities.

Example 2.2 *Tree augmented Bayesian classifiers.* The tree augmented Bayesian classifier incorporates some dependencies between the features of $x \in X$ by building a directed tree among the feature variables. That is, the n features form a graph which is restricted to a directed tree that represents the dependency relations between the features. Additionally there is an arc between the class labels and each feature. The structure of a tree augmented Bayesian classifier is depicted in Figure 2.2(a).

Example 2.3 *Bayesian network augmented Bayesian classifiers.* The Bayesian network augmented Bayesian classifier can be considered as an generalization of tree augmented Bayesian classifier by relaxing the constrain that the graph structure between features has to be a tree. In this case, it considers that the dependency structure among the features constitutes a directed acyclic graph (DAG). As with all aforementioned classifiers, there is a directed arc between the class node and each feature. The structure of a Bayesian network augmented Bayesian classifier is depicted in Figure 2.2(b).

The likelihood term (2.2) for the sample features x given the class label y can be obtained in a similar way as with the naive Bayesian classifiers; however, now each feature not only depends on the class but also on some other features according to the structure of the graph. Thus, we need to consider the conditional probability of each feature given the class and the feature of its parent nodes:

$$\mathbf{P}(x \mid y) = \mathbf{P}(x_1, \dots, x_n \mid y) = \prod_{i=1}^{n} \mathbf{P}(x_i \mid \mathbf{pa}(x_i), y). \tag{2.8}$$

where $\mathbf{pa}(x_i)$ is the set of parent nodes of feature x_i according to the feature dependency structure of the classifier.

The tree and Bayesian network augmented Bayesian classifiers can be considered as particular cases of a more general model, that is, *Bayesian networks*, which will be covered in more detail in the subsequent chapters.

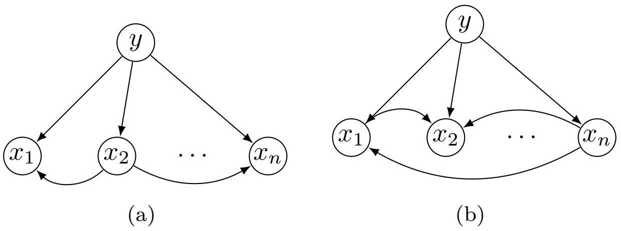


Figure 2.2 (a) Example of a tree augmented Bayesian classifier. (b) Example of a Bayesian network augmented Bayesian classifier.

2.1.3 Semi-naive Bayesian classifiers

Another alternative to deal with dependent features is to transform the basic structure of a naive Bayesian classifier, while maintaining a tree-structured network. This has as an advantage that the efficiency and simplicity of the naive Bayesian classifier is maintained, and at the same time the performance is improved for cases where the features are not independent. These types of Bayesian classifiers are known as *semi-naive Bayesian classifiers.*

The basic idea of semi-naive Bayesian classifiers is to eliminate or join features which are not independent given the class label, such that the performance of the classifier improves. This is analogous to feature selection in machine learning, and there are two types of approaches:

- *Filter.* The features are selected according to a local measure, for instance the mutual information between the feature and the class.
- *Wrapper.* The features are selected based on a global measure, usually by comparing the performance of the classifier with and without the feature.

Additionally, the learning algorithm can start from an empty structure and add features; or from a full structure with all the features, and eliminate (or combine) features. Figure 2.3 illustrates the two alternative operations to modify the structure of a naive Bayesian classifier starting from a full graph. Node elimination consists in simply eliminating a feature x_i from the graph, this could be because it is not relevant for the class (x_i and y are independent); or because the feature x_i and another feature x_j are not independent given the class. The rationale for eliminating one of the dependent features is that if the features are not independent given the class, one of them is redundant and could be eliminated. Node combination consists in merging two features x_i and x_j into a new feature, e.g., $x_i x_j$, which is an alternative when two features are not independent given the class. By merging them into a single feature, the independence condition is not longer relevant. In principle we should select from the two alternatives that implies a higher improvement in the performance of the classifier and the feature selection process can be performed multiple iterations until there are no more superfluous or dependent features. Then the same parameter learning and inference process as in naive Bayesian classifiers can be directly applied.

2.2 Multi-dimensional classification

All previous classifiers consider that there is a single class variable; that is, each sample belongs to one and only one class. Several important problems need to predict several classes simultaneously, in which more than one class can be assigned to a sample $x \in X$. Such problems are called multi-dimensional classification problems. Similarly, an 'ordinary' multi-dimensional classifier consists in finding a vector-valued function $f\colon X \to Y$, which directly assigns to each sample $x \in X$ an m-dimensional vector of class values $\hat{y} \in Y$, with $\mathbf{dom}(Y) \subseteq \mathbf{Z}_+^m$. From a probabilistic point of view, the classifier first predicts the posterior probability $\mathbf{P}(y \mid x)$ for all $y \in Y$ given sample $x \in X$, then the 'hard' class labels $\hat{y}$ will be assigned according to $\hat{y} = \operatorname{argmax}_y \mathbf{P}(y \mid x)$.

In this chapter we only consider a particular case of multi-dimensional classification problems — *multi-label classification*, where all class labels are binary, i.e., $\mathbf{dom}(Y_i) = \{0, 1\}$, $i = 1, \ldots, m$. We will introduce the Bayesian network methods in subsequent chapters, which can be applied to solve the general multi-dimensional classification problems.

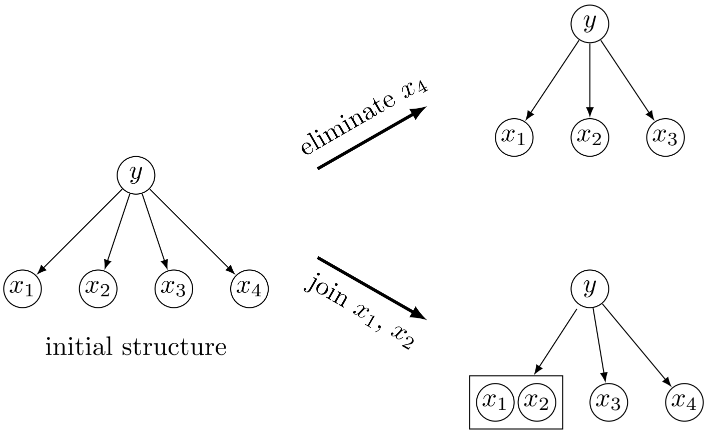


Figure 2.3 Example of structure improving process of a semi-naive Bayesian classifier.

2.2.1 Basic approaches

There are two basic approaches for multi-label classification problems: *binary relevance* and *label power-set*. Binary relevance approaches transform the multi-label classification problem into m independent binary classification problems, one for each class $Y_1, \dots, Y_m$. A classifier is independently learned for each class and the results are combined to determine the predicted class set; the dependencies between classes are not considered. The label power-set approach transforms the multi-label classification problem into a single-class scenario by defining a new compound class variable whose possible values are all the possible combinations of values of the original classes. In this case the interactions between classes are implicitly considered. Essentially, binary relevance can be effective when the classes are relatively independent, and label power-set when there are few class variables.

2.2.2 Chain classifiers

Under the framework of Bayesian classifiers, we can consider an alternative to the binary relevance approach — the *chain classifiers*, which implicitly incorporate the dependencies between classes by adding additional features to each independent classifier. A chain classifier consists of m base binary classifiers $(f_1, \dots, f_m)$ which are linked in a chain, such that each classifier incorporates the predicted classes $\hat{y}_i \in \{0, 1\}$, $i = 1, \dots, m$, by the previous classifiers as additional features. Thus, each classifier f_i in the chain is trained to learn the association of label $\hat{y}_i$ given the features augmented with all previous predicted class labels $(\hat{y}_1, \dots, \hat{y}_{i-1})$ in the chain. Formally, this process can be denoted as

$$\begin{gathered}\hat{y}_1 = \operatorname{argmax}_{y_1} \mathbf{P}(y_1 \mid x), \\ \hat{y}_i = \operatorname{argmax}_{y_i} \mathbf{P}(y_i \mid x, \hat{y}_1, \dots, \hat{y}_{i-1}), \quad i = 2, \dots, m.\end{gathered} \tag{2.9}$$

As in the binary relevance approach, the final prediction of the class vector is determined by concatenating the outputs of all the binary classifiers in the chain,

$$\hat{y} = (\hat{y}_1, \dots, \hat{y}_m).$$

A challenge for chain classifiers is to select the order of the classes in the chain, since the order can affect the performance of the classifier. We describe two approaches to address this challenge as follows.

Circular chain classifiers

In a *circular chain classifier*, the propagation of the predicted classes from the previous binary classifiers is done iteratively in a circular way. In the first cycle, as in the standard chain classifiers (2.9), the predictions of the previous classifiers are additional features for each classifier in the

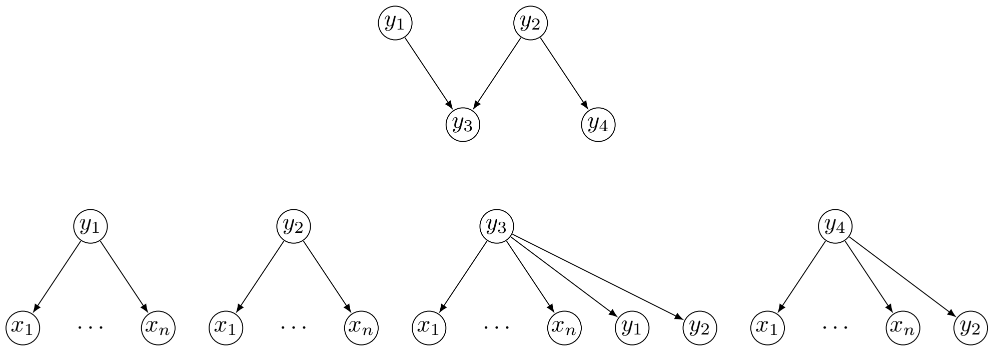


Figure 2.4 Example of a Bayesian chain classifier. Top: the graph representing the class dependency structure. Bottom: naive Bayesian classifiers, one for each class.

chain. After the first cycle, each binary classifier in the chain receives the predictions of all other classifiers as additional feature, and update its prediction according to

$$\hat{y}_i = \operatorname*{argmax}_{y_i} \mathbf{P}(y_i \mid x, \hat{y}_{-i}), \quad i = 1, \ldots, m, \tag{2.10}$$

where $\hat{y}_{-i}$ denotes the predicted class vector $\hat{y}$ with the ith entry removed. This process is repeated for a prefixed number of cycles or until convergence.

Bayesian chain classifiers

Bayesian chain classifiers consider the following two assumptions. Firstly, the dependency of different classes given the features can be represented as a DAG. Then the posterior probability of class vector y given sample $x \in X$ reduces to

$$\mathbf{P}(y \mid x) = \mathbf{P}(y_1, \ldots, y_m \mid x) = \prod_{i=1}^{m} \mathbf{P}(y_i \mid \mathbf{pa}(y_i), x). \tag{2.11}$$

Secondly, since the calculation of the final prediction $\hat{y}$ involves solving a hard combinatorial optimization problem on variable y,

$$\text{maximize} \quad \textstyle\prod_{i=1}^{m} \mathbf{P}(y_i \mid \mathbf{pa}(y_i), x), \tag{2.12}$$

Bayesian chain classifiers assume an approximation on this problem that it can be solved approximately via a sequence of independent optimization problems, each focusing on maximizing the conditional probability of a single variable y_i:

$$\text{maximize} \quad \mathbf{P}(y_i \mid \mathbf{pa}(y_i), x), \tag{2.13}$$

for all $i = 1, \ldots, m$. That is, the most probable joint combination of class assignments is approximated by the concatenation of the most probable individual classes. The final output class vector $\hat{y}$ of the Bayesian chain classifier will be a simple concatenation of all outputs $\hat{y}_i$ from the ith run of (2.13).

The first assumption is reasonable if we have enough data to obtain a good approximation of the class dependency structure, and assuming that this is obtained conditioned on the features. Regarding the second assumption, the total abduction or most probable explanation is not always equivalent to the maximization of the individual classes. However, the assumption is less strong than that assumed by the binary relevance approach. Bayesian chain classifiers provide an attractive alternative to multi-dimensional classification, as they incorporate in certain ways the dependencies between class variables, and they keep the efficiency of the binary relevance approach. For the base classifier that belongs to each class we can use any of the Bayesian classifiers presented in the previous sections, for instance a naive Bayesian classifier. The general idea for building a Bayesian chain classifier is illustrated in Figure 2.4.

Bibliography

This chapter is adapted from [Suc21, §4]. §4.7 of this book also provides some introduction on solving hierarchical classification problems with Bayesian classifiers, which is not included in our discussion.

The book by Michie et al. [MSTC95] can be referred to for a general introduction and comparison between different classification approaches.

Some discussion about the lost of discriminative information when using binning to discretize the continuous feature values for naive Bayesian classifiers are provided in [HY01].

In Gaussian naive Bayes, sometimes the distribution of class-conditional marginal densities for continuous features is far from normal. In these cases, kernel density estimation can be used for a more realistic estimate of the marginal densities of each class. This method, which was introduced by John and Langley [JL13] can boost the accuracy of the classifier considerably.

Except the Gaussian naive Bayes introduced in this chapter, there are many other useful extensions of naive Bayesian classifiers to continuous feature spaces, such as Multinomial naive Bayes [RSTK03] and Bernoulli naive Bayes [MN$^+$98].

A more detailed description about variants of augmented Bayesian classifiers can be found in [FGG97].

The semi-naive Bayesian classifier was initially introduced in [MAS06], and later extended by Pazzani [Paz95]. The paper [MAS06] can be referred to for more detail about the feature selection process of a semi-naive Bayesian classifier.

The paper [TK07] provides an overview on multi-label classification problems, where some additional discussions about the label power-set approach can be found.

Chain classifiers for multi-label classification problems were initially introduced in [RPHF11], and were extended to incorporate a Bayesian network architecture by Sucar et al. [SBM$^+$14]. Rivas et al. [ROES18] did some convergence analysis on circular chain classifiers and it has been shown empirically that circular chain classifiers tend to converge in a few iterations and that the order of the classes in the chain does not affect the performance according to several metrics.

Exercises

2.1 *Naive Bayesian classifier.* Displayed in the table below is a dataset focusing on golf with solely discrete features. The initial four columns denote weather forecasts for various dates, while the final column indicates whether a golf game was played on that particular day. Our aim is to employ a naive Bayesian classifier to predict whether a game will be played in the future based on the weather forecast.

Outlook	Temperature	Humidity	Windy	Play
sunny	high	high	false	no
sunny	high	high	true	no
overcast	high	high	false	yes
rainy	medium	high	false	yes
rainy	low	normal	false	yes
rainy	low	normal	true	no
overcast	low	normal	true	yes
sunny	medium	high	false	no
sunny	low	normal	false	yes
rainy	medium	normal	false	yes
sunny	medium	normal	true	yes
overcast	medium	high	true	yes
overcast	high	normal	false	yes
rainy	medium	high	true	no

(a) Illustrate the graphical representation of the naive Bayesian classifier.

(b) Before examining the samples for prediction, contemplate which parameters might be necessary. Deduce these parameters manually using the provided dataset.

(c) Offer a prediction after examining the subsequent weather forecasts with the learnt parameters.

- $(\text{overcast}, \text{low}, \text{high}, \text{false})$.
- $(\text{rainy}, \text{high}, \text{high}, \text{false})$.
- $(\text{sunny}, \text{low}, \text{normal}, \text{true})$.

2.2 *Gaussian naive Bayes.* In this exercise, our objective is to address a binary classification problem involving continuous features using a naive Bayesian classifier. The dataset provided in `data.csv` is structured as follows: the first column indicates whether the sample belongs to the training or testing dataset, followed by the 20 feature vector entries, and finally, the last column record the binary sample labels. Please complete the Gaussian naive Bayes algorithm in `src.py`, and check your implementation using the notebook `2-2.ipynb`.

Chapter 3

Markov models

3.1 Markov chains

We consider a stochastic process

$$\{X_0, X_1, \ldots, X_t, \ldots\}$$

with random variables $X_0, X_1, \ldots, X_t, \ldots$ sharing the same domain $\mathbf{dom}(X)$. Suppose the conditional probability of any future *state* $x_{t+1} \in X$ given the past states $x_0, \ldots, x_{t-1}$ and present state x_t, is independent of the past states and depends only on the present state,

$$\mathbf{P}(x_{t+1} \mid x_0, \ldots, x_t) = \mathbf{P}(x_{t+1} \mid x_t), \tag{3.1}$$

then this stochastic process is called a *Markov chain.* The equation (3.1) is called the *Markov property.* The random variable X is called the *state space* of the Markov chain.[1] Specifically, if the conditional probability $\mathbf{P}(X_{t+1} = x_j \mid X_t = x_i)$ is independent of time t given the same x_i and x_j, the Markov chain is *time-homogeneous.* Note that in the following discussion we will use subscripts to denote both the state instance at time t, denoted as x_t, and the ith instance of random variable X (suppose we assign each entry in X a unique index), denoted as x_i. The intended meaning of the notation should be inferred from the context of its usage. We hope it will cause no confusion.

For a time-homogeneous Markov chain, let P be a square matrix with each entry

$$P_{ij} = \mathbf{P}(x_j \mid x_i), \tag{3.2}$$

for all $x_i, x_j \in X$, representing the probability that the next state is x_j given the present state x_i. Considering an m-dimensional state space X, i.e., $\mathbf{card}(X) = m$, clearly the matrix P has the following property:

- $P \in \mathbf{R}^{m \times m}$.
- $P_{ij} \geq 0$ for all $i = 1, \ldots, m$, $j = 1, \ldots, m$.
- $P_{i:}^T \mathbf{1} = 1$ for all $i = 1, \ldots, m$.

Here we use $P_{i:}$ to denote the ith row of matrix P, represented as a column vector. The matrix P is called the *transition matrix* of the Markov chain, which can be represented graphically with a *state transition diagram.* The state transition diagram is a directed graph where each note is a state and the arcs represent possible transitions between states, weighted by corresponding transition probabilities. If an arc between state x_i and x_j does not appear in the diagram, it means that the corresponding transition probability P_{ij} is zero. Figure 3.1 shows an example of a Markov chain with a 3-dimensional state space. Note that although the state transition diagram and the graphical model diagram are both represented with graphs, they have completely different interpretations. The former represents the transition probability between states, i.e., different instances of the random variable X, while the latter represents the probabilistic dependencies between different random variables.

[1] In this course we will only consider the case where the state space is finite and discrete.

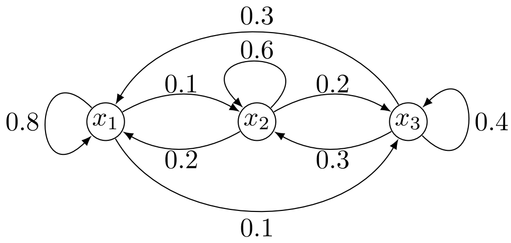


Figure 3.1 Example of state transition diagram of a Markov chain.

Except for the transition matrix P, to uniquely determine an m-dimensional Markov chain[2], we will also need the vector of initial state distribution:

$$\rho = (\ldots, \mathbf{P}(X_0 = x_i), \ldots), \quad i = 1, \ldots, m. \tag{3.3}$$

Obviously, the initial state distribution $\rho \in \mathbf{R}^m$ has to satisfy $\rho \succeq 0$ and $\rho^T \mathbf{1} = 1$, where the generalized inequality symbol '$\succeq$' denotes the componentwise inequality between two vectors. Thus the parameter set of a Markov chain can be denoted as a 2-dimensional set

$$\Theta = \{\rho, P\}.$$

3.1.1 Inference and parameter learning

If the parameter set Θ of a Markov chain is known, we can calculate the probability of observing any sequence of states $\zeta = \{X_0 = x_i, X_1 = x_j, X_2 = x_k, \ldots\}$ generated by that Markov chain, which is basically the product of the transition probabilities of the sequence of states:

$$\mathbf{P}(\zeta \mid \rho, P) = \rho_i P_{ij} P_{jk} \cdots . \tag{3.4}$$

On the other hand, if the parameters are unknown, but we are given a set of observed state sequences $\mathcal{D} = \{\zeta_1, \zeta_2, \ldots\}$ from that Markov chain, where $\zeta = \{x_0, \ldots, x_N\}$ for all $\zeta \in \mathcal{D}$, we can estimate the ρ and P for the Markov chain according to:

$$\rho_i = \mathbf{E}_{\zeta \sim \mathcal{D}}[\mathbf{P}(X_0 = x_i \mid \zeta)], \quad i = 1, \ldots, m, \tag{3.5}$$

and

$$P_{ij} = \frac{\mathbf{E}_{\zeta \sim \mathcal{D}, t}\left[\mathbf{P}(X_t = x_i, X_{t+1} = x_j \mid \zeta)\right]}{\mathbf{E}_{\zeta \sim \mathcal{D}, t}\left[\mathbf{P}(X_t = x_i \mid \zeta)\right]}, \quad i = 1, \ldots, m, \quad j = 1, \ldots, m. \tag{3.6}$$

Note that for the last observed state x_N in each sequence we do not observe the next state, so the above expectations are estimated across $t = 0, \ldots, N-1$.

Remark 3.1 The equations (3.5) and (3.6), which are used to learn parameters of a Markov chain based on observations, intuitively align with our understanding. It's validity can also be shown analytically as follows.

The problem of estimating the initial state distribution ρ and the transition matrix P of a Markov chain according to the set of observations $\mathcal{D}$ can be formally defined as a *maximum likelihood estimation* (MLE) problem,

$$\begin{array}{ll} \text{maximize} & l_{\mathcal{D}}(\Theta) = \mathbf{E}_{\zeta \sim \mathcal{D}}\left[\log \mathbf{P}(\zeta \mid \rho, P)\right] \\ \text{subject to} & \rho \succeq 0, \ \rho^T \mathbf{1} = 1 \\ & P_{ij} \geq 0, \quad i = 1, \ldots, m, \quad j = 1, \ldots, m \\ & P_{i:}^T \mathbf{1} = 1, \quad i = 1, \ldots, m, \end{array} \tag{3.7}$$

where $\Theta = \{\rho, P\}$ is the optimization variable and $\mathcal{D}$ is the problem data. The function $l_{\mathcal{D}}(\Theta)$ is called the *log-likelihood function* of model parameter Θ given the observation $\mathcal{D}$.

[2] We will always assume the Markov chain to be time-homogeneous except mentioned specifically.

Note that the objective function of (3.7) can be written as

$$\begin{aligned}
l_{\mathcal{D}}(\Theta) &= \mathbf{E}_{\zeta\sim\mathcal{D}}\left[\log \mathbf{P}(\zeta \mid \rho, P)\right] \\
&= \mathbf{E}_{\zeta\sim\mathcal{D}}\left[\log \mathbf{P}(x_0 \mid \rho) \prod_{t=0}^{N-1} \mathbf{P}(x_{t+1} \mid x_t, P)\right] \\
&= \mathbf{E}_{\zeta\sim\mathcal{D}}\left[\log \mathbf{P}(x_0 \mid \rho)\right] + \mathbf{E}_{\zeta\sim\mathcal{D}}\left[\sum_{t=0}^{N-1} \log \mathbf{P}(x_{t+1} \mid x_t, P)\right] \\
&= \mathbf{E}_{\zeta\sim\mathcal{D}}\left[\sum_{i=1}^{m} I_{x_i}(x_0)\log \rho_i\right] + \mathbf{E}_{\zeta\sim\mathcal{D}}\left[\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{t=0}^{N-1} I_{x_i}(x_t) I_{x_j}(x_{t+1}) \log P_{ij}\right], \qquad (3.8)
\end{aligned}$$

where $I_{x_i}(x)$ is an indicator function with $I_{x_i}(x) = 1$ if $x = x_i$, and 0 otherwise. Thus problem (3.7) can be transformed into the following two optimization problems

$$\begin{array}{ll}
\text{maximize} & \mathbf{E}_{\zeta\sim\mathcal{D}}\left[\sum_{i=1}^{m} I_{x_i}(x_0)\log \rho_i\right] \\
\text{subject to} & \rho \succeq 0, \;\; \rho^T \mathbf{1} = 1,
\end{array} \qquad (3.9)$$

with optimization variable ρ, and

$$\begin{array}{ll}
\text{maximize} & \mathbf{E}_{\zeta\sim\mathcal{D}}\left[\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{t=0}^{N-1} I_{x_i}(x_t) I_{x_j}(x_{t+1}) \log P_{ij}\right] \\
\text{subject to} & P_{ij} \geq 0, \quad i = 1,\ldots,m, \quad j = 1,\ldots,m \\
& P_{i:}^T \mathbf{1} = 1, \quad i = 1,\ldots,m,
\end{array} \qquad (3.10)$$

with optimization variable P, which are maximized by equation (3.5) and (3.6) (according to the Gibbs' inequality), respectively.

Example 3.1 Consider that we have the following observation sequences generated from the Markov chain described in Figure 3.1:

- $(x_2, x_2, x_3, x_3, x_3, x_3, x_1)$.
- $(x_1, x_3, x_2, x_3, x_3, x_3, x_3)$.
- (x_3, x_3, x_2, x_2).
- $(x_2, x_1, x_2, x_2, x_1, x_3, x_1)$.

According to these observations, the initial state distribution can be estimated as:

$$\rho = \left(\frac{1}{4}, \frac{2}{4}, \frac{1}{4}\right) = (0.25, 0.5, 0.25)\,,$$

and the transition matrix of the Markov chain is

$$P = \begin{bmatrix} \frac{0}{3} & \frac{1}{3} & \frac{2}{3} \\ \frac{2}{7} & \frac{3}{7} & \frac{2}{7} \\ \frac{2}{11} & \frac{2}{11} & \frac{7}{11} \end{bmatrix}.$$

3.1.2 Convergence

Convergence is another useful property of Markov chains. That says, if a Markov chain with $\mathbf{card}(X) = m$ satisfies the following two requirements:

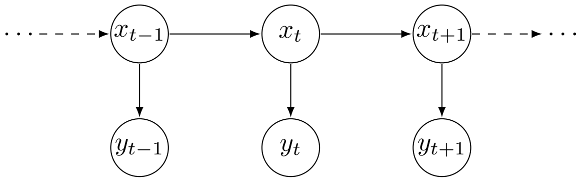


Figure 3.2 Graphical model representing a hidden Markov model. The top variables represent the hidden states and the nodes on the bottom are the observations.

- *Irreducible.* From every state $x_i \in X$ there is a probability $P_{ij} > 0$ of transiting to any state $x_j \in X$.
- *Aperiodic.* For any given state, there isn't a fixed interval after which the chain will return to the same state.

Then this Markov chain will reduce to a unique stationary state distribution $\pi \in \mathbf{R}_+^m$ with $\pi^T \mathbf{1} = 1$ when $t \to \infty$, such that

$$\pi P = \pi. \tag{3.11}$$

In this case, the matrix P^t (the transition matrix P to the tth power) converges to a rank-one matrix in which each row is the stationary distribution π:

$$\lim_{t\to\infty} P^t = \mathbf{1}\pi^T. \tag{3.12}$$

The rate of convergence of a Markov chain is determined by the second largest eigen-value of the transition matrix P.

3.2 Hidden Markov models

A *hidden Markov model* (HMM) consists of a Markov chain $\{X_0, X_1, \dots, X_t, \dots\}$ with domain $\mathbf{dom}(X)$ whose states are not directly observable, and an observable stochastic process $\{Y_0, Y_1, \dots, Y_t, \dots\}$ with domain $\mathbf{dom}(Y)$ whose outcomes depend only on the present instance of X in a known way,

$$\mathbf{P}(y_t \mid x_0, \dots, x_t) = \mathbf{P}(y_t \mid x_t). \tag{3.13}$$

The set Y is called the *observation space* of the hidden Markov model. For example, in weather forecasting, the weather cannot be directly measured; in reality, the weather is estimated based on the results from a series of sensors — temperature, pressure, wind velocity, etc. Figure 3.2 shows a hidden Markov model represented in a graph diagram.

We first consider a standard hidden Markov model where both the state space X and observation space Y are discrete and finite with $\mathbf{card}(X) = m$ and $\mathbf{card}(Y) = n$, respectively. To uniquely determine a hidden Markov model, the latent Markov chain can be well defined with the initial state distribution ρ and the transition matrix P. Besides, to describe the relationship between each latent state $x \in X$ and observation $y \in Y$, let B be a matrix with each entry

$$B_{ij} = \mathbf{P}(y_j \mid x_i), \tag{3.14}$$

for all $x_i \in X$, $y_i \in Y$, representing the probability of observing y_j under state x_i. The matrix B is called the *emission matrix* of the hidden Markov model. Similar to the transition matrix P, the emission matrix B has the following property:

- $B \in \mathbf{R}^{m\times n}$.
- $B_{ij} \geq 0$ for all $i = 1, \dots, m$, $j = 1, \dots, n$.
- $B_{i:}^T \mathbf{1} = 1$ for all $i = 1, \dots, m$.

Thus the parameter set of a hidden Markov model can be denoted as a 3-dimensional set

$$\Theta = \{\rho, P, B\}.$$

Given a hidden Markov model representation of a certain domain, there are three basic questions that are of interest in most applications.

1. *Inference.* Given a model, estimate the probability of a sequence of observations.
2. *Decoding.* Given a model and a particular observation sequence, estimate the most probable state sequence that produced the observations.
3. *Parameter learning.* Given some sequences of observations, estimate the parameters of the model.

3.2.1 Inference

The inference problem of a hidden Markov model consists in determining the posterior probability of observing some sequence $\varphi = \{y_0, \ldots, y_N\}$, given the model parameters $\Theta = \{\rho, P, B\}$, that is, estimating the conditional probability $\mathbf{P}(\varphi \mid \Theta)$.

Direct method

Note that a sequence of observations $\varphi = \{y_0, \ldots, y_N\}$ can be generated by different state sequences $\zeta = \{x_0, \ldots, x_N\}$. Thus, to calculate the posterior probability of a given observation sequence, we can estimate the probability for a certain state sequence, and then add together the estimations for all the possible state sequences, resulting in

$$\mathbf{P}(\varphi \mid \Theta) = \sum_{\zeta} \mathbf{P}(\varphi, \zeta \mid \Theta). \tag{3.15}$$

Given a possible state sequence $\zeta = \{x_0, \ldots, x_N\}$, the probability $\mathbf{P}(\varphi, \zeta \mid \Theta)$ can be obtained by first calculating the probability of generating the specific state sequence, and then multiplying to the probability of observing corresponding observations from the state sequence, i.e.,

$$\mathbf{P}(\varphi, \zeta \mid \Theta) = \mathbf{P}(x_0 \mid \rho)\mathbf{P}(y_0 \mid x_0, B) \prod_{t=0}^{N-1} \mathbf{P}(x_{t+1} \mid x_t, P)\mathbf{P}(y_{t+1} \mid x_{t+1}, B), \tag{3.16}$$

where the probabilities on the right hand side are respective entries of the vector ρ and matrices P and B. Put together, the direct method estimates the posterior probability of observing some sequence $\varphi = \{y_0, \ldots, y_N\}$ according to

$$\mathbf{P}(\varphi \mid \Theta) = \sum_{x_0, \ldots, x_N} \mathbf{P}(x_0 \mid \rho)\mathbf{P}(y_0 \mid x_0, B) \prod_{t=0}^{N-1} \mathbf{P}(x_{t+1} \mid x_t, P)\mathbf{P}(y_{t+1} \mid x_{t+1}, B). \tag{3.17}$$

For a model with m states and an observation length of N, the direct method requires a number of operations in the order of $2Nm^N$ (or simply m^N) to solve the inference problem. This can be less practical when the length of observations is relatively large.

Iterative method

The basic idea of the iterative method, also known as the *forward algorithm*, is to estimate the probabilities of the observations per time step. That is, starting from $t = 0$, calculate the posterior probability of a partial sequence of observations until time t,

$$\mathbf{P}(\varphi_{0:t} \mid \Theta) = \mathbf{P}(y_0, \ldots, y_t \mid \Theta),$$

and based on this partial result, calculate it for time $t + 1$, until the end of the observation sequence where $t = N$.

First we introduce an auxiliary variable $\alpha_t \in \mathbf{R}_+^m$, and each of its entries $\alpha_t(i)$ is defined as

$$\alpha_t(i) = \mathbf{P}(y_0, \ldots, y_t, X_t = x_i \mid \Theta), \quad i = 1, \ldots, m. \tag{3.18}$$

The auxiliary variable α_t is called the *forward probability*, representing the posterior probability of observing the partial observation sequence up until time t, and the state under which the tth observation was generated is x_i. The forward probability can be calculated recursively with

$$\alpha_t(i) = \begin{cases} \rho_i \mathbf{P}(y_0 \mid X_0 = x_i, B) & t = 0 \\ \alpha_{t-1}^T P_{:i} \mathbf{P}(y_t \mid X_t = x_i, B) & t > 0, \end{cases} \tag{3.19}$$

for all $i = 1, \dots, m$. Then the posterior probability of observing the whole sequence $\varphi = \{y_0, \dots, y_N\}$ given model parameters Θ can be obtained by summing up all the entries of the forward probability evaluated at time $t = N$,

$$\mathbf{P}(\varphi \mid \Theta) = \alpha_N^T \mathbf{1}. \tag{3.20}$$

The pseudocode of the forward algorithm is shown in Algorithm 3.1.

Algorithm 3.1 *Forward algorithm.*

```
given hidden Markov model parameters $\Theta$, observation sequence $\varphi$.
for $i = 1, \dots, m$ do
    $\alpha_0(i) \coloneqq \rho_i \mathbf{P}(y_0 \mid X_0 = x_i, B)$.
end for
for $t = 1, \dots, N$ do
    for $i = 1, \dots, m$ do
        $\alpha_t(i) \coloneqq \alpha_{t-1}^T P_{:i} \mathbf{P}(y_t \mid X_t = x_i, B)$.
    end for
end for
return $\mathbf{P}(\varphi \mid \Theta) = \alpha_N^T \mathbf{1}$.
```

We now analyze the time complexity of the iterative method. Each iteration requires approximately m multiplications and m additions, so for the N iterations, the number of floating point operations is in the order of Nm^2, or simply m^2. Thus, the time complexity is reduced from exponential in N for the direct method to quadratic in m for the iterative method.

3.2.2 Decoding

Given a sequence of observations $\varphi = \{y_0, \dots, y_N\}$ and model parameters Θ, the decoding problem of hidden Markov models can be interpreted in two ways:

- *Optimal state prediction.* Finding the most probable state x_t^* at time t.
- *Optimal sequence prediction.* Finding the most probable state sequence $\zeta^* = \{x_0^*, \dots, x_N^*\}$ that generated observation sequence.

Optimal state prediction

Similar to the forward probability α_t, let $\beta_t \in \mathbf{R}_+^m$ be the *backward probability*, where each of its entry $\beta_t(i)$ is defined as

$$\beta_t(i) = \mathbf{P}(y_{t+1}, \dots, y_N \mid X_t = x_i, \Theta), \quad i = 1, \dots, m. \tag{3.21}$$

The backward probability β_t represents the posterior probability of observing the partial observation sequence after time t until the end of the sequence given that the state at that time is x_i. By defining the backward probability of the last observation ($t = N$) to be equal to 1, the others can be calculated recursively as

$$\beta_t(i) = \begin{cases} \displaystyle\sum_{j=1}^{m} \beta_{t+1}(j) P_{ij} \mathbf{P}(y_{t+1} \mid X_{t+1} = x_j, B) & t < N \\ 1 & t = N, \end{cases} \tag{3.22}$$

for all $i = 1, \dots, m$. Figure 3.3 illustrates the computation of forward probability α_t and backward probability β_t across the graph diagram of a hidden Markov model. Note that by combining

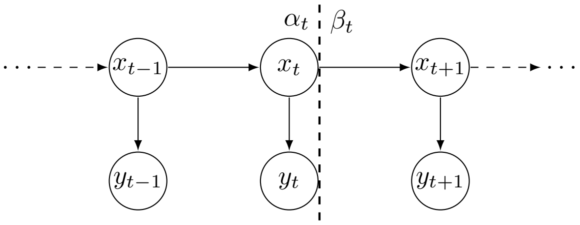


Figure 3.3 Computation of forward probability α_t and backward probability β_t of a hidden Markov model. Vertical dashed line indicates the separation point of the two probabilities at time t.

the forward probability and backward probability, we can obtain the objective of the inference problem from §3.2.1, $\mathbf{P}(\varphi \mid \Theta)$, at any time t along the observation sequence, according to

$$\mathbf{P}(\varphi \mid \Theta) = \alpha_t^T \beta_t, \tag{3.23}$$

for all $t = 0, \ldots, N$.

Now we define another auxiliary variable $\gamma_t \in \mathbf{R}_+^m$, which represents the conditional probability of being in state x_i given the whole observation sequence:

$$\gamma_t(i) = \mathbf{P}(X_t = x_i \mid \varphi, \Theta), \quad i = 1, \ldots, m. \tag{3.24}$$

According to Bayes' rule, the vector γ_t can be written in terms of α_t and β_t as:

$$\gamma_t(i) = \frac{\mathbf{P}(X_t = x_i, \varphi \mid \Theta)}{\mathbf{P}(\varphi \mid \Theta)} = \frac{\alpha_t(i)\beta_t(i)}{\alpha_t^T \beta_t}, \tag{3.25}$$

for all $i = 1, \ldots, m$. Then the most probable state x_t^* at time t given the observation sequence φ and parameters Θ of a hidden Markov model can be obtained by

$$x_t^* = \operatorname*{argmax}_{x_i} \gamma_t(i). \tag{3.26}$$

Optimal sequence prediction

The optimal sequence prediction problem of a hidden Markov model consists in finding a sequence of states $\zeta^* = \{x_0^*, \ldots, x_N^*\}$ such that

$$x_0^*, \ldots, x_N^* = \operatorname*{argmax}_{x_0, \ldots, x_N} \mathbf{P}(x_0, \ldots, x_N \mid \varphi, \Theta), \tag{3.27}$$

given the observation sequence $\varphi = \{y_0, \ldots, y_N\}$ and model parameters Θ. This problem can be solved approximately by a simple concatenation of the optimal states at each time t, i.e.,

$$\tilde{\zeta}^* = \left\{ \operatorname*{argmax}_{x_i} \gamma_t(i) \;\middle|\; t = 0, \ldots, N \right\}. \tag{3.28}$$

However, (3.28) is not guaranteed to provide a global optimal result of the state sequence since it does not consider the transition between states.

To obtain the exact solution of the optimal state sequence prediction problem, note that

$$\mathbf{P}(\zeta \mid \varphi, \Theta) = \frac{\mathbf{P}(\zeta, \varphi \mid \Theta)}{\mathbf{P}(\varphi \mid \Theta)} \propto \mathbf{P}(\zeta, \varphi \mid \Theta),$$

thus maximizing the posterior probability $\mathbf{P}(\zeta \mid \varphi, \Theta)$ over ζ is equivalent to maximizing the joint probability $\mathbf{P}(\zeta, \varphi \mid \Theta)$ of the state and observation sequence over ζ. This problem can be solved with the *Viterbi algorithm*. Let $\delta_t \in \mathbf{R}_+^m$ be the maximum value of the joint probability of a subsequence of states until time $t-1$, and observations until time t, with the state at time t is x_i,

$$\delta_t(i) = \max_{x_0, \ldots, x_{t-1}} \mathbf{P}(x_0, \ldots, x_{t-1}, X_t = x_i, y_0, \ldots, y_t \mid \Theta), \quad i = 1, \ldots, m. \tag{3.29}$$

This probability δ_t can be interpreted as the probability of being in state x_i at time t given that the state subsequence up until $t-1$ is optimal w.r.t. the partial observation sequence $\{y_0, \dots, y_{t-1}\}$. The probability δ_t can also be obtained recursively as

$$\delta_t(i) = \max_{j=1,\dots,m} (\delta_{t-1}(j)P_{ji})\,\mathbf{P}(y_t \mid X_t = x_i, B), \tag{3.30}$$

for all $i = 1, \dots, m$. Let $\psi_t \in \mathbf{Z}^m_{++}$ store the index j of the previous state x_j at time $t-1$ that gives the maximum probability $\delta_{t-1}(j)P_{ji}$, for each state i at time t, i.e.,

$$\psi_t(i) = \operatorname*{argmax}_{j=1,\dots,m} \delta_{t-1}(j)P_{ji}, \quad i = 1, \dots, m, \tag{3.31}$$

which is used to reconstruct the state sequence by backtracking from the last state. Let p^* be the probability of obtaining the given observation sequence under the optimal state sequence $\zeta^* = \{x_0^*, \dots, x_N^*\}$, the complete procedure of the Viterbi algorithm is shown in Algorithm 3.2.

Algorithm 3.2 *Viterbi algorithm.*

```
given hidden Markov model parameters $\Theta$, observation sequence $\varphi$.
1. Initialization.
for $i = 1, \dots, m$ do
    $\delta_0(i) := \rho_i \mathbf{P}(y_0 \mid X_0 = x_i, B)$.
end for
2. Recursion.
for $t = 1, \dots, N$ do
    for $i = 1, \dots, m$ do
        $\delta_t(i) := \max_{j=1,\dots,m} (\delta_{t-1}(j)P_{ji})\,\mathbf{P}(y_t \mid X_t = x_i, B)$.
        $\psi_t(i) := \operatorname{argmax}_{j=1,\dots,m} \delta_{t-1}(j)P_{ji}$.
    end for
end for
3. Termination.
$p^* := \max_{i=1,\dots,m} \delta_N(i)$.
$i_N^* := \operatorname{argmax}_{i=1,\dots,m} \delta_N(i)$.
$x_N^* := x_{i_N^*}$.
4. Backtracking.
for $t = N, \dots, 1$ do
    $i_{t-1}^* := \psi_t(i_t^*)$.
    $x_{t-1}^* := x_{i_{t-1}^*}$.
end for
```

3.2.3 Parameter learning

The *expectation-maximization algorithm* (EM) is commonly used in estimating the parameters of models involving latent variables. The idea is to start with some initial parameters for the model, which can be initialized randomly or based on some domain knowledge. In the E-step, some likelihood function w.r.t. the current model parameters is calculated. Then in the M-step, these parameters are optimized to maximizing an MLE objective. This cycle is repeated until convergence; e.g., until the difference between the parameters for the model from one step to the next is below a certain threshold.

To learn the parameters of a hidden Markov model, suppose we are given a set of observation sequences $\mathcal{D} = \{\varphi_1, \varphi_2, \dots\}$ from that hidden Markov model, where $\varphi = \{y_0, \dots, y_N\}$ for all $\varphi \in \mathcal{D}$. First we should note that the cardinalities of the state space and observation space of the model have to be known or previously defined. Let $\Theta = \{\rho, P, B\}$ be the set of estimated parameters of the model from the previous EM-iteration, and $\Theta^+ = \{\rho^+, P^+, B^+\}$ be the new set of parameters to be updated. For each EM-iteration, we first calculate the current estimation of hidden state sequence $\zeta = \{x_0, \dots, x_N\}$ with parameters Θ. Then similar to learning the parameters of a Markov chain (§3.1.1), the parameters ρ^+, P^+, and B^+ of a hidden Markov model can be updated according to

$$\rho_i^+ = \mathbf{E}_{\varphi\sim\mathcal{D}} \left[\mathbf{P}(X_0 = x_i \mid \varphi, \Theta)\right], \quad i = 1, \dots, m, \tag{3.32}$$

$$P_{ij}^{+} = \frac{\mathbf{E}_{\varphi\sim\mathcal{D},t}\left[\mathbf{P}(X_t = x_i, X_{t+1} = x_j \mid \varphi, \Theta)\right]}{\mathbf{E}_{\varphi\sim\mathcal{D},t}\left[\mathbf{P}(X_t = x_i \mid \varphi, \Theta)\right]}, \quad i = 1,\dots,m, \quad j = 1,\dots,m, \tag{3.33}$$

and

$$B_{ij}^{+} = \frac{\mathbf{E}_{\varphi\sim\mathcal{D},t}\left[\mathbf{P}(X_t = x_i, Y_t = y_j \mid \varphi, \Theta)\right]}{\mathbf{E}_{\varphi\sim\mathcal{D},t}\left[\mathbf{P}(X_t = x_i \mid \varphi, \Theta)\right]}, \quad i = 1,\dots,m, \quad j = 1,\dots,n. \tag{3.34}$$

Again, these updating equations align with our intuition, but can also be derived analytically. (See exercise 3.1.) To obtain the probability term on the numerator of (3.33), let $\xi_t \in \mathbf{R}_{+}^{m\times m}$ represent the probability of transitioning from state x_i at time t to state x_j at time $t+1$ given an observation sequence φ:

$$\begin{aligned}\xi_t(i,j) &= \mathbf{P}(X_t = x_i, X_{t+1} = x_j \mid \varphi, \Theta)\\ &= \frac{\mathbf{P}(X_t = x_i, X_{t+1} = x_j, \varphi \mid \Theta)}{\mathbf{P}(\varphi \mid \Theta)}, \quad i = 1,\dots,m, \quad j = 1,\dots,m.\end{aligned} \tag{3.35}$$

The denominator $\mathbf{P}(\varphi \mid \Theta)$ is just a normalization factor, which can be calculated from:

$$\mathbf{P}(\varphi \mid \Theta) = \sum_{i=1}^{m}\sum_{j=1}^{m} \mathbf{P}(X_t = x_i, X_{t+1} = x_j, \varphi \mid \Theta).$$

The auxiliary variable ξ_t can be written in terms of the forward probability α_t and the backward probability β_t:

$$\xi_t(i,j) = \frac{\alpha_t(i)P(i,j)\mathbf{P}(y_{t+1} \mid X_{t+1} = x_j, B)\beta_{t+1}(j)}{\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_t(i)P(i,j)\mathbf{P}(y_{t+1} \mid X_{t+1} = x_j, B)\beta_{t+1}(j)}, \tag{3.36}$$

for all $i = 1,\dots,m$, $j = 1,\dots,m$. Clearly, the probability γ_t can also be written in terms of ξ_t:

$$\gamma_t(i) = \sum_{j=1}^{m} \xi_t(i,j), \tag{3.37}$$

for all $i = 1,\dots,m$. As a result, the equations (3.32) to (3.34) can be represented compactly with the previously defined auxiliary variables as

$$\rho_i^{+} = \mathbf{E}_{\varphi\sim\mathcal{D}}[\gamma_0(i)], \quad i = 1,\dots,m, \tag{3.38}$$

$$P_{ij}^{+} = \frac{\mathbf{E}_{\varphi\sim\mathcal{D},t}[\xi_t(i,j)]}{\mathbf{E}_{\varphi\sim\mathcal{D},t}[\gamma_t(i)]}, \quad i = 1,\dots,m, \quad j = 1,\dots,m, \tag{3.39}$$

and

$$B_{ij}^{+} = \frac{\mathbf{E}_{\varphi\sim\mathcal{D},t}[\gamma_t(i)I_j(y_t)]}{\mathbf{E}_{\varphi\sim\mathcal{D},t}[\gamma_t(i)]}, \quad i = 1,\dots,m, \quad j = 1,\dots,n, \tag{3.40}$$

where $I_j(y)$ is an indicator function with $I_j(y) = 1$ if the index of y in observation space Y is equal to j, and 0 otherwise. Note that similar to (3.6), the expectations in (3.33) and (3.39) over time t are estimated across $t = 0,\dots,N-1$. The update equations (3.38) to (3.40) are called the *Baum-Welch algorithm*. The pseudocode for the whole procedure is shown in Algorithm 3.3.

Algorithm 3.3 *Baum-Welch algorithm.*

```
given the estimated hidden Markov model parameters $\Theta$ from previous EM-iteration, the set of observation sequences $\mathcal{D}$.
1. Estimate the initial state distribution $\rho^{+}$ according to (3.38).
2. Estimate the transition matrix $P^{+}$ according to (3.39).
3. Estimate the emission matrix $B^{+}$ according to (3.40).
return the updated set of parameters $\Theta^{+} = \{\rho^{+}, P^{+}, B^{+}\}$.
```

As the last point, it should be noted that while the EM algorithm is guaranteed to converge to a local optimum, it does not ensure a global optimal solution. The convergence point of EM depends on its initial conditions.

3.2.4 Continuous observation space

In many applications the observation space is continuous. In this case, an alternative to discretization is to work directly with the continuous features, assuming them to be Gaussian distributed. This assumption leads to the *Gaussian hidden Markov models*, where the initial state ditribution vector and the transition matrix are as those in standard hidden Markov models, but the map from each state to the observation space are modeled as a Gaussian distribution. Suppose the domain of the observation space $\mathbf{dom}(Y) = \mathbf{R}$, then the emission map of the hidden Markov model is given by the Gaussian density function $\mathcal{N}(\mu_i, \sigma_i^2)$:

$$p(y \mid X = x_i) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(y-\mu_i)^2}{2\sigma_i^2}\right), \quad i = 1, \dots, m. \tag{3.41}$$

This idea can be further extended to vector observations, for example $y \in \mathbf{R}^n$ for all $y \in Y$, by considering the following joint Gaussian density function $\mathcal{N}(\mu_i, \Sigma_i)$ as an alternative of (3.41):

$$p(y \mid X = x_i) = \frac{1}{\sqrt{(2\pi)^n \det \Sigma_i}} \exp\left(-\frac{1}{2}(y-\mu_i)^T \Sigma_i^{-1} (y-\mu_i)\right), \quad i = 1, \dots, m, \tag{3.42}$$

with the vector of distribution mean $\mu_i \in \mathbf{R}^n$ and the covariance matrix $\Sigma_i \in \mathbf{S}_{++}^n$. Moreover, sometimes the observation space can not be described by a single Gaussian distribution, in this case we can use a *Gaussian mixture model*, which consists of multiple Gaussian distributions that are combined to represent the desired distribution.

The algorithms for solving the three basic problems (inference, decoding, and parameter learning) of a Gaussian hidden Markov model are essentially the same as those for standard hidden Markov models, just considering that the observations are modeled as a Gaussian distribution or a Gaussian mixture model.

Bibliography

A general introduction to Markov chains is provided in the article [KS$^+$69]. The convergence property of Markov chains is a corollary of the *Perron-Frobenius theorem*, which is discussed in detail in [HJ12].

The Viterbi algorithm was initially introduced by Andrew Viterbi in decoding convolutional codes, which is widely used in communications. The original paper [Vit67] contains some details about the underlying idea of dynamical programming and related mathematical derivations.

The convergence analysis about the expectation-maximization algorithm can be found in [LR19].

About some other variants of hidden Markov models and their applications, one can refer to [AASSMDPC11].

Exercises

3.1 *Parameter learning of hidden Markov models.* Suppose we are given a set of observation sequences $\mathcal{D} = \{\varphi_1, \varphi_2, \dots\}$ from a hidden Markov model with unknown parameters, where $\varphi = \{y_0, \dots, y_N\}$ for all $\varphi \in \mathcal{D}$. Let $\zeta = \{x_0, \dots, x_N\}$ be the latent state sequence that generated φ. For each iteration of EM, finding the optimal parameters ρ^+, P^+, and B^+ of the hidden Markov model can be formulated as the following optimization problem:

$$\text{maximize} \quad J(\Theta^+ \mid \Theta) = \mathbf{E}_{\varphi \sim \mathcal{D}, \zeta} \left[\log \mathbf{P}(\varphi, \zeta \mid \Theta^+)\right],$$

where $J(\Theta^+ \mid \Theta)$ is the EM objective; Θ^+ is the optimization variable, and $\mathcal{D}, \Theta$ are the problem data. To solve the problem, note that

$$\begin{aligned} J(\Theta^+ \mid \Theta) &= \mathbf{E}_{\varphi \sim \mathcal{D}, \zeta} \left[\log \mathbf{P}(\varphi, \zeta \mid \Theta^+)\right] \\ &= \mathbf{E}_{\varphi \sim \mathcal{D}} \left[\sum_{\zeta} \mathbf{P}(\zeta \mid \varphi, \Theta) \log \mathbf{P}(\varphi, \zeta \mid \Theta^+)\right]. \end{aligned}$$

Using this information, show that the optimal value of ρ^+, P^+, and B^+ for each EM-iteration are given by (3.32) to (3.34).
Hint.

- $\sum_{\zeta} \mathbf{P}(\zeta \mid \varphi, \Theta) I_i(x_t) = \mathbf{P}(X_t = x_i \mid \varphi, \Theta)$.
- Similar derivation can be done here on $J(\Theta^+ \mid \Theta)$ as was in (3.8).

3.2 In this exercise, our objective is to learn the parameters of a hidden Markov model given a set of observation sequences generated by it. The provided dataset `data.csv` is a 100×50 matrix with each row representing one sequence of observations. We know that the cardinalities of the state and observation space are 3 and 2, respectively. Please complete the expectation-maximization algorithm for parameter learning in `src.py`, and check your implementation using the notebook `3-2.ipynb`. You should see an increase of log-likelihood as the number of EM-iterations increases.

Chapter 4

Bayesian networks

4.1 Representation

A Bayesian network represents the joint distribution of a set of discrete random variables, $X_1, \ldots, X_n$, as a directed acyclic graph (DAG) and some sets of conditional probabilities. Each node, that corresponds to a variable, has an associated set of conditional probabilities that contains the probability of each instance of the variable given its parents in the graph, which is known as the *conditional probability table.* The structure of the network implies a set of conditional independence assertions, which give power to this representation. Figure 4.1 depicts an example of a simple Bayesian network. In this example, X_4 is conditionally independent of X_1, X_3, X_5, X_6 given X_2, that is:

$$\mathbf{P}(X_4 \mid X_1, X_2, X_3, X_5, X_6) = \mathbf{P}(X_4 \mid X_2).$$

4.1.1 Structure

Mappings

Given a probability distribution $\mathbf{P}$ of $X = (X_1, \ldots, X_n)$, and its graphical representation G, there must be a correspondence between the conditional independence in $\mathbf{P}$ and in G; this is called a *mapping.* There are three basic types of mappings:

- *D-map*: all the conditional independence relations in $\mathbf{P}$ are satisfied in G.
- *I-map*: all the conditional independence relations in G are true in $\mathbf{P}$.
- *P-map*: or perfect map, it is a D-map and an I-map.

Particularly, we say the graph G and probability distribution $\mathbf{P}$ are *compatible* if G is an I-map of $\mathbf{P}$. In general, it is not always possible to have a perfect mapping of the independence relations between the graph G and the distribution $\mathbf{P}$, so we settle for what is called a *minimal I-map*: all the conditional independence relations implied by G are true in P, and if any arc is deleted in G this condition is lost.

d-separation

Consider three disjoint sets of variables, X, Y, and Z, which are represented as nodes in a directed acyclic graph G. To test whether X is independent of Y given Z in any distribution compatible with G, we need to test whether the nodes corresponding to variables Z 'block' all paths from nodes in X to nodes in Y. By path we mean a sequence of consecutive edges (of any directionality) in the graph, and blocking is to be interpreted as stopping the flow of information (or of dependency) between the variables that are connected by such paths. A path p is said to be *d-separated* (or blocked) by a set of nodes Z if and only if

1. the path p contains a chain $i \to m \to j$ or a fork $i \leftarrow m \to j$ such that the middle node m is in Z, or

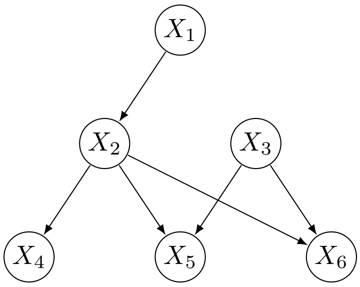


Figure 4.1 An example Bayesian network.

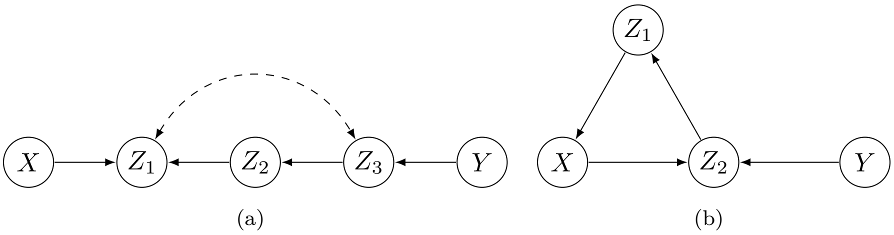


Figure 4.2 Graphs illustrating d-separation. In (a), X and Y are d-separated given Z_2 and d-connected given Z_1. In (b), X and Y cannot be d-separated by any set of nodes.

2. the path p contains an inverted fork (or collider) $i \rightarrow m \leftarrow j$ such that the middle node m is not in Z and such that no descendant of m is in Z.

A set Z is said to d-separate X from Y if and only if Z blocks every path from a node in X to a node in Y. In this case, random variable X is independent of Y conditional on Z in every distribution compatible with G. Conversely, if X and Y are not d-separated by Z in a directed acyclic graph G, then X and Y are dependent conditional on Z in at least one distribution compatible with G.

The intuition behind d-separation is simple and can best be recognized if we attribute causal meaning to the arrows in the graph. In causal chains $i \rightarrow m \rightarrow j$ and causal forks $i \leftarrow m \rightarrow j$, the two extreme variables are marginally dependent but become independent of each other (i.e., blocked) once we condition on (i.e., know the value of) the middle variable. Figuratively, conditioning on m appears to 'block' the flow of information along the path, since learning about i has no effect on the probability of j, given m. Inverted forks $i \rightarrow m \leftarrow j$, representing two causes having a common effect, act the opposite way; if the two extreme variables are (marginally) independent, they will become dependent (i.e., connected through unblocked path) once we condition on the middle variable (i.e., the common effect) or any of its descendants.

Example 4.1 *d-separation.* Figure 4.2(a) contains a bidirected arc $Z_1 \longleftrightarrow Z_3$, and Figure 4.2(b) involves a directed cycle $X \rightarrow Z_2 \rightarrow Z_1 \rightarrow X$. In Figure 4.2(a), the two paths between X and Y are blocked when none of $\{Z_1, Z_2, Z_3\}$ is measured. However, the path $X \rightarrow Z_1 \longleftrightarrow Z_3 \leftarrow Y$ becomes unblocked when Z_1 is measured. This is so because Z_1 unblocks the 'colliders' at both Z_1 and Z_3; the first because Z_1 is the collision node of the collider, the second because Z_1 is a descendant of the collision node Z_3 through the path $Z_1 \leftarrow Z_2 \leftarrow Z_3$. In Figure 4.2(b), X and Y cannot be d-separated by any set of nodes, including the empty set. If we condition on Z_2, we block the path $X \leftarrow Z_1 \leftarrow Z_2 \leftarrow Y$ yet unblock the path $X \rightarrow Z_2 \leftarrow Y$. If we condition on Z_1, we again block the path $X \leftarrow Z_1 \leftarrow Z_2 \leftarrow Y$ and unblock the path $X \rightarrow Z_2 \leftarrow Y$ because Z_1 is a descendant of the collision node Z_2.

According to the previous definition of d-separation, any node X is conditionally independent of all nodes in G that are not descendants of X given its parents in the graph, $\mathbf{pa}(X)$, which is known as the *Markov assumption*. The set of parents of a variable X is called the *contour* of X. Besides, we call the *Markov blanket* of a node X, denoted as $\mathbf{mb}(X)$, is a set of nodes that make X independent of all the other nodes in G:

$$\mathbf{P}(X \mid G_{-X}) = \mathbf{P}(X \mid \mathbf{mb}(X)). \tag{4.1}$$

For a Bayesian network, the Markov blanket of X consists of:

- the parents of X and,
- the children of X and,
- the other parents of the children of X.

For instance, in the Bayesian network of Figure 4.1, the Markov blanket of X_4 is X_2, and the Markov blanket of X_2 is X_1, X_3, X_4, X_5, X_6.

The structure of a Bayesian network can then be specified by the parents of each variable. For example the Bayesian network in Figure 4.1, its structure can be specified as:

$$\left\{\begin{array}{l}\mathbf{pa}(X_1) = \emptyset,\\ \mathbf{pa}(X_2) = \{X_1\},\\ \mathbf{pa}(X_3) = \emptyset,\\ \mathbf{pa}(X_4) = \{X_2\},\\ \mathbf{pa}(X_5) = \{X_2, X_3\},\\ \mathbf{pa}(X_6) = \{X_2, X_3\}\end{array}\right\}.$$

Then using the chain rule, we can specify the joint probability distribution of the set of variables in a Bayesian network as the product of the conditional probability of each variable given its parents:

$$\mathbf{P}(X_1, \ldots, X_n) = \prod_{i=1}^{n} \mathbf{P}(X_i \mid \mathbf{pa}(X_i)). \tag{4.2}$$

For the example in Figure 4.1:

$$\mathbf{P}(X_1, \ldots, X_6) = \mathbf{P}(X_1)\mathbf{P}(X_2 \mid X_1)\mathbf{P}(X_3)\mathbf{P}(X_4 \mid X_2)\mathbf{P}(X_5 \mid X_2, X_3)\mathbf{P}(X_6 \mid X_2, X_3).$$

4.1.2 Parameters

To complete the specification of a Bayesian network, we need to define its parameters, which are the conditional probabilities of each node given its parents in the graph: $\mathbf{P}(X_i \mid \mathbf{pa}(X_i))$. In the case of continuous variables, wee need to specify a function that relates the density function of each variable to the density of its parents; in the case of discrete variables, the number of parameters to specify $\mathbf{P}(X_i \mid \mathbf{pa}(X_i))$ can be combinatorially large as the number of parents of X_i increases. Two main alternatives have been proposed to overcome this issue, one is based on *canonical models* and the other on graphical representations of conditional probability tables.

Canonical models

Canonical models represent the relations between a set of random variables for particular interactions using few parameters. It can be applied when the probabilities of a random variable in a Bayesian network conform to certain canonical relations with respect to the configurations of its parents. There are several classes of canonical models, the most common are the *noisy-OR* and *noisy-AND* for binary variables, and their extensions for multivalued variables, *noisy-max* and *noisy-min*, respectively. Canonical models can provide a considerable reduction in the number of parameters when a variable has many parents; and also some inference techniques take advantage of this compact representation.

Example 4.2 *Noisy-OR.* Consider an OR logic gate, in which the output is true if any of its inputs are true. The noisy-OR model is based on the concept of the logic OR; the difference is that there is a certain (small) probability that the variable is not true even if one or more of its parents are true. The noisy-OR model is applied when several variables or causes can produce an effect if any one of them is true, and as more of the causes are true, the probability of the effect increases. For instance, the effect could be a certain symptom or effect, E, and the causes are a number of possible diseases, $C_1, \dots, C_n$, that can produce the symptom, such that if none of the diseases is present (all false), the symptom does not appear; and when any disease is present (true) the symptom is present with high probability and it increases as the number of $C_i = \text{true}$ increases. A graphical representation of a noisy-OR relation in a Bayesian network is depicted in Figure 4.3.

Formally, the following two conditions must be satisfied for a noisy-OR canonical model to be applicable:

- *Independence of exceptions*: if an effect is the manifestation of several causes, the mechanisms that inhibit the occurrence of the effect, $E = \text{false}$, under one cause are independent of the mechanisms that inhibit it under the other causes[a], i.e.,

$$\mathbf{P}(E = \text{false} \mid C_1, \dots, C_n) = \prod_{i=1}^{n} \mathbf{P}(E = \text{false} \mid C_i). \tag{4.3}$$

- *Responsibility*: the effect E is false if all the possible causes are false. Together with (4.3), this implies

$$\mathbf{P}(E = \text{false} \mid C_i = \text{false}) = 1, \quad i = 1, \dots, n. \tag{4.4}$$

The probability that the effect E is inhibited (it does not occur) under cause C_i is defined as:

$$q_i = \mathbf{P}(E = \text{false} \mid C_i = \text{true}). \tag{4.5}$$

Given this definition and the previous conditions, the parameters in the conditional probability table for a noisy-OR model can be obtained using the following expressions when all the n causes are true:

$$\mathbf{P}(E = \text{false} \mid C_1 = \text{true}, \dots, C_n = \text{true}) = \prod_{i=1}^{n} q_i, \tag{4.6}$$

and

$$\mathbf{P}(E = \text{true} \mid C_1 = \text{true}, \dots, C_n = \text{true}) = 1 - \prod_{i=1}^{n} q_i. \tag{4.7}$$

In general, if k out of n causes are true, then (informally):

$$\mathbf{P}(E = \text{false} \mid C_1, \dots, C_n) = \prod_{i=1}^{k} q_i, \tag{4.8}$$

so that if all the causes are false then the effect is false with probability one. Thus, only one parameter is required per parent variable to construct the conditional probability table — the inhibition probability q_i. In this case the number of independent parameters $(q_1, q_2, \dots, q_n)$ increases linearly with the number of parents, instead of exponentially. As an example, consider a noisy-OR model with 3 causes, C_1, C_2, and C_3, where the inhibition probabilities are the same for the three, $q_1 = q_2 = q_3 = 0.1$. Given these parameters we can obtain the conditional probability table for the effect variable E, as shown in Table 4.1.

[a] This independence does not necessarily hold for $E = \text{true}$.

Graphical representations

Canonical models apply in certain situations but do not provide a general solution for compact representations of conditional probability tables. An alternative representation is based on the observation that within each conditional probability table, the same probability values tend to be repeated several times. Thus, it is not necessary to represent these repeated values many times. A representation that takes advantage of this condition is *decision tree*, which could be used for representing a conditional probability table in a compact way. In a decision tree, each internal node corresponds to a variable in the conditional probability table, and the branches

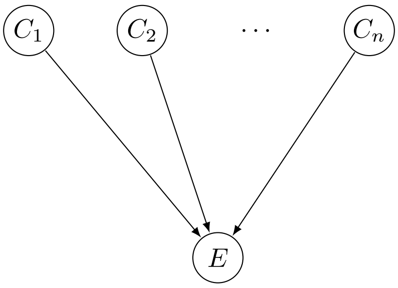


Figure 4.3 Graphical representation of a noisy-OR structure. The cause variables $C_1, \ldots, C_n$ are the parents of the effect variable E.

Table 4.1 Conditional probability table for a noisy-OR variable with three parents and parameters $q_1 = q_2 = q_3 = 0.1$.

C_1	0	0	0	0	1	1	1	1
C_2	0	0	1	1	0	0	1	1
C_3	0	1	0	1	0	1	0	1
$\mathbf{P}(E = 0)$	1	0.1	0.1	0.01	0.1	0.01	0.01	0.001
$\mathbf{P}(E = 1)$	0	0.9	0.9	0.99	0.9	0.99	0.99	0.999

from a node correspond to the different values that a variable can take. The leaf nodes in the tree represent the different probability values. A trajectory from the root to a leaf, specifies a probability value for the corresponding variables. If a variable is omitted in a trajectory, it means that the conditional probability table has the same probability for all values of this variable. Another graphical representation of conditional probability tables is *decision diagram*, which extends decision tree by considering a directed acyclic graph structure, such that it is not restricted to a tree. This avoids the need to duplicate repeated probability values in the leaf nodes, and in some cases provides an even more compact representation. Examples about these two graphical representations of conditional probability tables, $\mathbf{P}(X \mid A, B, C, D, E, F, G)$, is shown in Figure 4.4, assuming variables A, B, C, D, E, F, G are all binary.

4.2 Inference

4.2.1 Belief propagation

The *belief propagation* algorithm only applies to *singly connected graphs* (trees and polytrees). Although there is an extension of belief propagation for general Bayesian networks, but the convergence is not guaranteed.

Given certain evidence E, the posterior probability for any variable $X = x_i$ can be obtained by applying the Bayes rule:

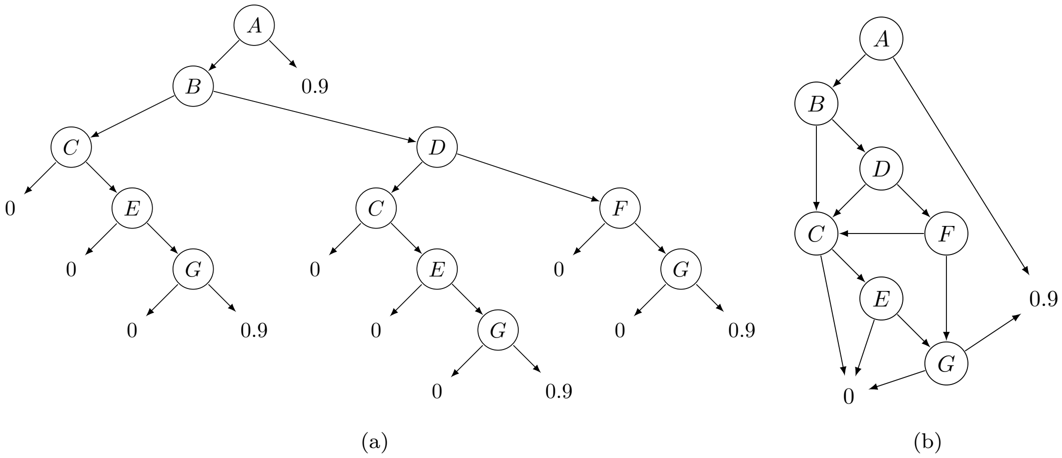
$$\mathbf{P}(x_i \mid E) = \frac{\mathbf{P}(E \mid x_i)\mathbf{P}(x_i)}{\mathbf{P}(E)}. \tag{4.9}$$

Given that the Bayesian network has a tree structure, the network can then be divided by any node into two independent subtrees. Thus, we can separate the evidence into (Figure 4.5):

- E_-: evidence of the rooted tree in X, and
- E_+: all other evidence.

Since E_+ and E_- are conditionally independent given X, thus

$$\begin{aligned}\mathbf{P}(x_i \mid E) &= \frac{\mathbf{P}(E \mid x_i)\mathbf{P}(x_i)}{\mathbf{P}(E)} && \text{(from (4.9))}\\ &= \frac{\mathbf{P}(E_-, E_+ \mid x_i)\mathbf{P}(x_i)}{\mathbf{P}(E)}\end{aligned}$$



(a)

(b)

Figure 4.4 Examples about (a) decision tree and (b) decision diagram representation of a conditional probability table.

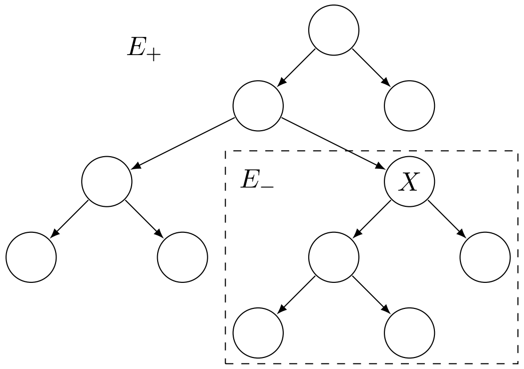


Figure 5.5 Divide the evidence E into two independent components E_+ and E_- with node X.

$$\begin{aligned}
&= \frac{\mathbf{P}(E_- \mid x_i)\mathbf{P}(E_+ \mid x_i)\mathbf{P}(x_i)}{\mathbf{P}(E)} \\
&= \frac{\mathbf{P}(E_- \mid x_i)\mathbf{P}(x_i \mid E_+)\mathbf{P}(E_+)\cancel{\mathbf{P}(x_i)}}{\mathbf{P}(E)\cancel{\mathbf{P}(x_i)}} \\
&= \frac{1}{Z}\mathbf{P}(x_i \mid E_+)\mathbf{P}(E_- \mid x_i), \qquad (4.10)
\end{aligned}$$

where $\frac{1}{Z} = \frac{\mathbf{P}(E_+)}{\mathbf{P}(E)}$ is the normalization constant. Let us introduce two auxiliary variables:

$$\mu(x_i) = \mathbf{P}(x_i \mid E_+), \qquad (4.11)$$

and

$$\lambda(x_i) = \mathbf{P}(E_- \mid x_i), \qquad (4.12)$$

then (4.10) can be written as

$$\mathbf{P}(x_i \mid E) = \frac{1}{Z}\mu(x_i)\lambda(x_i). \qquad (4.13)$$

Equation (4.13) is the basis of the belief propagation algorithm to obtain the posterior probability of all non-instantiated nodes. The computation of the posterior probability of any node X is decomposed into two parts: (1) the evidence λ coming from the children of X in the tree, and the evidence μ coming from the parent of X. We can think of each node X in the tree as a simple processor that stores its vectors

$$\mu(X) = (\ldots, \mathbf{P}(x_i \mid E_+), \ldots), \quad i = 1, 2, \ldots$$

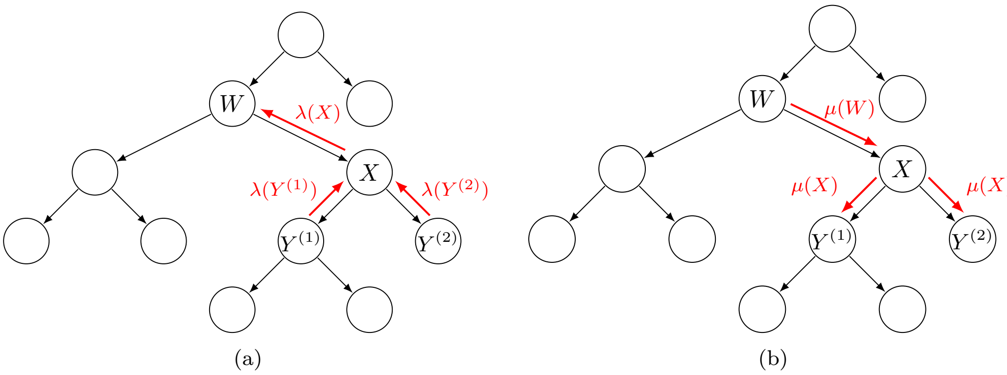


Figure 4.6 (a) Bottom-up propagation of λ evidence. (b) Top-down propagation of μ evidence.

and

$$\lambda(X) = (\ldots, \mathbf{P}(E_- \mid x_i), \ldots), \quad i = 1, 2, \ldots,$$

and its conditional probability table $\mathbf{P}(X \mid \mathbf{pa}(X))$. The evidence is propagated via a message passing mechanism, in which each node sends the corresponding messages to its parent and children in the tree.

Next we derive the equations for the messages. For the λ messages, given that the children of X are conditionally independent given x_i, we have

$$\lambda(x_i) = \mathbf{P}(E_- \mid x_i) = \prod_k \mathbf{P}(E_-^{(k)} \mid x_i), \tag{4.14}$$

where $E_-^{(k)}$ is the evidence coming from the tree rooted in the kth child $Y^{(k)}$ of X. Applying the rule of total probability conditioning on $Y^{(k)}$, we obtain

$$\begin{aligned}\mathbf{P}(E_-^{(k)} \mid x_i) &= \sum_{y^{(k)} \in Y^{(k)}} \mathbf{P}(E_-^{(k)} \mid x_i, y^{(k)}) \mathbf{P}(y^{(k)} \mid x_i) \\ &= \sum_{y^{(k)} \in Y^{(k)}} \mathbf{P}(E_-^{(k)} \mid y^{(k)}) \mathbf{P}(y^{(k)} \mid x_i) \\ &= \sum_{y^{(k)} \in Y^{(k)}} \lambda(y^{(k)}) \mathbf{P}(y^{(k)} \mid x_i), \end{aligned} \tag{4.15}$$

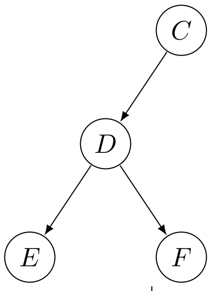
where the second equivalence comes from the fact that the evidence coming from the tree rooted in node $Y^{(k)}$ is conditionally independent of X given $Y^{(k)}$. Hence the λ evidence for each note can be calculated recursively according to the λ evidence of its children by performing a bottom-up propagation, which is shown in Figure 4.6(a).

Similarly, for the μ messages, we apply the rule of total probability conditioning on the parent node $W = \mathbf{pa}(X)$ of X:

$$\begin{aligned}\mu(x_i) &= \mathbf{P}(x_i \mid E_+) \\ &= \sum_{w \in W} \mathbf{P}(x_i \mid E_+, w) \mathbf{P}(w \mid E_+) \\ &= \sum_{w \in W} \mathbf{P}(x_i \mid w) \mathbf{P}(w \mid E_+), \end{aligned} \tag{4.16}$$

where the third equivalence is, again, because of X is conditionally independent of the evidence E_+ given W. $\mathbf{P}(w \mid E_+)$ corresponding to the probability of w given the evidence coming from all the tree except the subtree rooted on X. According to (4.13) to (4.15), it can be written as

$$\mathbf{P}(w \mid E_+) = \frac{1}{Z} \mu(w) \prod_{E_{W-}^{(k)} \neq E_-} \mathbf{P}(E_{W-}^{(k)} \mid w)$$



c_1	c_2
0.8	0.2

	c_1	c_2
d_1	0.9	0.7
d_2	0.1	0.3

	d_1	d_2
e_1	0.9	0.5
e_2	0.1	0.5

	d_1	d_2
f_1	0.7	0.4
f_2	0.3	0.6

Figure 4.7 A simple Bayesian network with corresponding conditional probability table.

$$= \frac{1}{Z}\mu(w) \prod_{X^{(k)} \neq X} \sum_{x^{(k)} \in X^{(k)}} \lambda(x^{(k)})\mathbf{P}(x^{(k)} \mid w), \tag{4.17}$$

where $X^{(k)}$ is the kth child of W, and $E^{(k)}_{W-}$ is the evidence coming from the tree rooted in $X^{(k)}$. Hence, the μ evidence of each node can be recursively expressed as a function of the μ evidence of its parent W, and the λ evidence of all other children of W. To calculate the μ evidence for all nodes, we can first perform a bottom-up propagation from all leaves of the tree (Figure 4.6(a)) to obtain the λ evidence of all nodes, then perform a top-down propagation from the root of the tree (Figure 4.6(b)).

Before the propagation starts, we should assign the evidence to those instantiated variables, and define the prior of the root and leaf nodes of the tree. For example, we can consider a uniform distribution $\lambda = \mathbf{1}$ if a leaf node is unknown, and a one-hot coding for a known leaf node (one for the assigned value and zero for all other values). After obtaining the λ and μ evidence vector of all nodes in the tree from the propagation, the posterior probability of any variable X is obtained by combining these vectors using (4.13) and normalizing.

Example 4.3 *Belief propagation.* We now illustrate the belief propagation algorithm with a simple example. Consider the Bayesian network in Figure 4.7 with 4 binary variables C, D, E, F, and the only evidence is $E = e_1$. Then the initial conditions for the leaf nodes are: $\lambda(E) = (1, 0)$ and $\lambda(F) = (1, 1)$. Propagating to the parent node D is basically multiplying the λ vectors by the corresponding conditional probability tables:

$$\begin{aligned}
\lambda(D) &= \begin{bmatrix} 1 \\ 0 \end{bmatrix}^T \begin{bmatrix} 0.9 & 0.5 \\ 0.1 & 0.5 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 1 \end{bmatrix}^T \begin{bmatrix} 0.7 & 0.4 \\ 0.3 & 0.6 \end{bmatrix} \\
&= \begin{bmatrix} 0.9 \\ 0.5 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 0.9 \\ 0.5 \end{bmatrix}.
\end{aligned}$$

Now propagating it to its parent C:

$$\lambda(C) = \begin{bmatrix} 0.9 \\ 0.5 \end{bmatrix}^T \begin{bmatrix} 0.9 & 0.7 \\ 0.1 & 0.3 \end{bmatrix} = \begin{bmatrix} 0.86 \\ 0.78 \end{bmatrix}.$$

In this way, we complete the bottom-up propagation.

We now perform the top-down propagation. Given that C is not instantiated with prior $(0.8, 0.2)$, we define $\mu(C) = (0.8, 0.2)$ and propagate to its child D:

$$\mu(D) = \begin{bmatrix} 0.8 \\ 0.2 \end{bmatrix}^T \begin{bmatrix} 0.9 & 0.1 \\ 0.7 & 0.3 \end{bmatrix} = \begin{bmatrix} 0.86 \\ 0.14 \end{bmatrix}.$$

We now propagate to its child F; however, given that D has another child, E,we also need to consider the λ message from this other child, thus:

$$\mu(F) = \left(\begin{bmatrix} 0.86 \\ 0.14 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 0 \end{bmatrix}^T \begin{bmatrix} 0.9 & 0.5 \\ 0.1 & 0.5 \end{bmatrix} \right)^T \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} = \begin{bmatrix} 0.57 \\ 0.27 \end{bmatrix}.$$

This completes the top-down propagation. Given the λ and μ vectors for each unknown variable, we just multiply them term by term and then normalize to obtain the posterior probabilities:

$$\mathbf{P}(C) = \frac{1}{Z} \begin{bmatrix} 0.8 \\ 0.2 \end{bmatrix} \odot \begin{bmatrix} 0.86 \\ 0.78 \end{bmatrix} = \frac{1}{Z} \begin{bmatrix} 0.69 \\ 0.16 \end{bmatrix} = \begin{bmatrix} 0.81 \\ 0.19 \end{bmatrix},$$

$$\mathbf{P}(D) = \frac{1}{Z} \begin{bmatrix} 0.86 \\ 0.14 \end{bmatrix} \odot \begin{bmatrix} 0.9 \\ 0.5 \end{bmatrix} = \frac{1}{Z} \begin{bmatrix} 0.77 \\ 0.07 \end{bmatrix} = \begin{bmatrix} 0.92 \\ 0.08 \end{bmatrix},$$

$$\mathbf{P}(F) = \frac{1}{Z} \begin{bmatrix} 0.57 \\ 0.27 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{Z} \begin{bmatrix} 0.57 \\ 0.27 \end{bmatrix} = \begin{bmatrix} 0.68 \\ 0.32 \end{bmatrix}.$$

4.2.2 Variable elimination

Assume a Bayesian network representing the joint probability distribution of $X = \{X_1, \ldots, X_n\}$. We want to calculate the posterior probability of a certain variable of subset of variables X_H, given a subset of evidence variables X_E; the remaining variables are X_R, such that $X = \{X_H \cup X_E \cup X_R\}$. The posterior probability of X_H given the evidence is:

$$\mathbf{P}(X_H \mid X_E) = \frac{\mathbf{P}(X_H, X_E)}{\mathbf{P}(X_E)}. \tag{4.18}$$

We can obtain both terms via marginalization of the joint distribution:

$$\mathbf{P}(X_H, X_E) = \sum_{X_R} \mathbf{P}(X), \tag{4.19}$$

and

$$\mathbf{P}(X_E) = \sum_{X_H} \mathbf{P}(X_H, X_E). \tag{4.20}$$

The objective of the variable elimination method is to perform these calculations efficiently. To achieve this, we can first represent the joint distribution as a product of local probabilities according to the network structure. Then, summations can be carried out only on the subset of terms which are a function of the variables being normalized. This approach takes advantage of the properties of summation and multiplication, resulting in the number of necessary operations being reduced.

Example 4.4 *Variable elimination.* Consider the Bayesian network in Figure 4.8, where we want to obtain $\mathbf{P}(A \mid D)$. In order to achieve this we need to obtain $\mathbf{P}(A, D)$ and $\mathbf{P}(D)$. To calculate the first term we should eliminate B, C, and E from the joint distribution, that is:

$$\begin{aligned} \mathbf{P}(A, D) &= \sum_B \sum_C \sum_E \mathbf{P}(A, B, C, D, E) \\ &= \sum_B \sum_C \sum_E \mathbf{P}(A)\mathbf{P}(B \mid A)\mathbf{P}(C \mid A)\mathbf{P}(D \mid B, C)\mathbf{P}(E \mid C) \\ &= \mathbf{P}(A) \sum_B \left[\mathbf{P}(B \mid A) \sum_C \left[\mathbf{P}(C \mid A)\mathbf{P}(D \mid B, C) \sum_E \mathbf{P}(E \mid C) \right] \right]. \end{aligned}$$

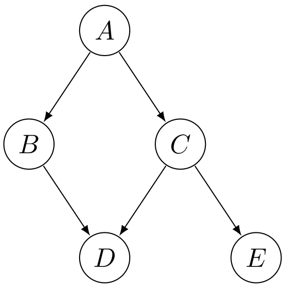


Figure 4.8 A simple Bayesian network.

If we consider that all variables are binary, this implies a reduction from 32 operations to 15 operations.

The critical aspect of the variable elimination algorithm is to select the appropriate order for eliminating each variable, as this has an important effect on the number of required operations. The different terms that are generated during the calculations are known as *factors* which are functions over a subset of variables, that map each instantiation of these variables to a non-negative number (not necessarily probabilities). In general a factor can be represented as $f(X_1, \ldots, X_m)$. For instance, in the previous example, one of the factors is $f(C, E) = \mathbf{P}(E \mid C)$, which is a function of two variables. The computational complexity in terms of space and time of the variable elimination algorithm is determined by the size of the factors, i.e., the number of variables on which the factor is defined. Basically, the complexity for eliminating any number of variables is exponential on the number of variables in the factor. Thus, the order in which the variables are eliminated should be selected so that the largest factor is kept to a minimum.

In general, finding the best order of variable elimination is NP-hard, but there are several heuristics that help to determine a good ordering for variable elimination. These heuristics can be explained based on the *interaction graph*, which is an undirected graph that is built during the process of variable elimination. The variables of each factor form a clique in the interaction graph. The initial interaction graph is obtained from the original Bayesian network structure by eliminating the direction of the arcs, and adding additional arcs between each pair of non-connected variables that have a common child. Then, each time a variable X_j is eliminated, the interaction graph is modified by adding an arc between each pair of neighbors of X_j that are not connected, and deleting variable X_j from the graph. For example, the interaction graphs that result from the Bayesian network in Figure 4.8 by the following elimination ordering: E, D, C, B, is depicted in Figure 4.9. Two popular heuristics for determining the elimination ordering, which can be obtained from the interaction graph, are the following:

- *Min-degree*: eliminate the variable that leads to the smallest possible factor, which is equivalent to eliminating the variable with the smallest number of neighbors in the current interaction graph.

- *Min-fill*: eliminate the variable that leads to adding the minimum number of edges to the interaction graph.

A disadvantage of variable elimination is that it only obtains the posterior probability of one variable (or subset of variables). To obtain the posterior probability of each non-instantiated variable in a Bayesian network, the calculations have to be repeated for each variable.

4.2.3 Conditioning

The conditioning method is based on the fact that an instantiated variable blocks the propagation of the evidence in a Bayesian network. Thus, we can cut the graph at an instantiated variable, and this can transform a multi-connected graph into a polytree, for which we can apply the belief propagation algorithm introduced in §4.2.1. In general, a subset of variables can be instantiated to transform a multi-connected network into a singly connected graph. If these variables are not actually known, we can set them to each of their possible values, and then do probability propagation for each value. With each propagation we obtain the posterior probabilities for all

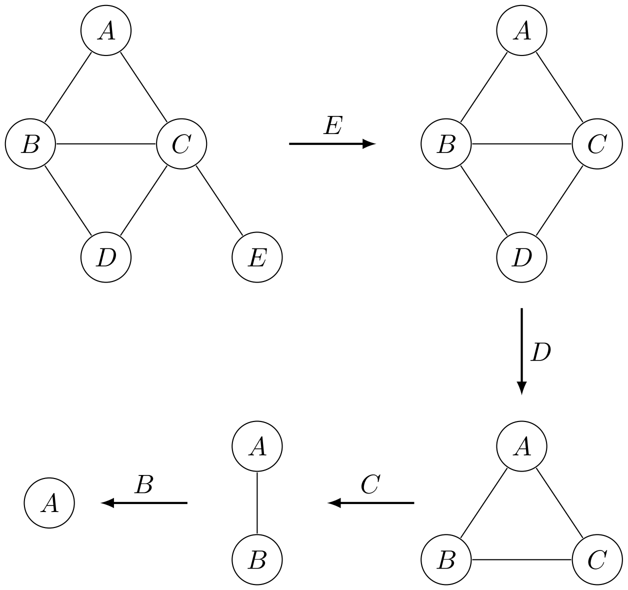


Figure 4.9 Interaction graphs resulting from the elimination of variables with ordering E, D, C, B from the Bayesian network in Figure 4.8.

unknown variables. Then, the final probability values are obtained as a weighted combination of these posterior probabilities.

First we will introduce the conditioning algorithm assuming we only need to partition a single variable and then we will extend it for multiple variables. Formally, we want to obtain the probability of any variable X, given the evidence E, conditioning on variable A. The variable A is selected such that by instantiating A, the multi-connected graph can be transformed to a singly connected graph. By the rule of total probability, we have

$$\mathbf{P}(X \mid E) = \sum_{a \in A} \mathbf{P}(X \mid E, a)\mathbf{P}(a \mid E), \tag{4.21}$$

where $\mathbf{P}(X \mid E, a)$ is the posterior probability of X which is obtained by probability propagation for each possible value of A, and the probability $\mathbf{P}(a \mid E)$ is the weight. By applying the Bayes rule, we obtain the following equation to estimate the weight:

$$\mathbf{P}(a \mid E) = \frac{1}{Z}\mathbf{P}(a)\mathbf{P}(E \mid a), \tag{4.22}$$

for all $a \in A$, where $\frac{1}{Z}$ is the normalization constant. The first term $\mathbf{P}(a)$ can be obtained by propagating without evidence. The second term $\mathbf{P}(E \mid a)$ can be calculated by propagation with $A = a$ to obtain the probability of the evidence variables.

Example 4.5 *Conditioning.* Consider the Bayesian network in Figure 4.8, this multi-connected network can be transformed into a polytree by assuming A is instantiated (Figure 4.10). If the evidence is D and E, then probabilities of the other variables A, B, and C can be obtained via conditioning following these steps:

1. Obtain the prior probability of A.
2. Obtain the probability of the evidence nodes D and E for each value of A by probability propagation in the polytree.
3. Calculate the weights $P(a \mid D, E)$ according to (4.22), with the probabilities obtained from the last two steps.
4. Estimate the probability of B and C for each value of A given the evidence by probability propagation in the polytree.
5. Obtain the posterior probabilities for B and C from the last two steps by applying (4.21).

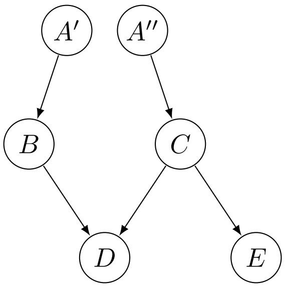


Figure 4.10 Transform the Bayesian network in Figure 4.8 into a singly connected network by instantiating A.

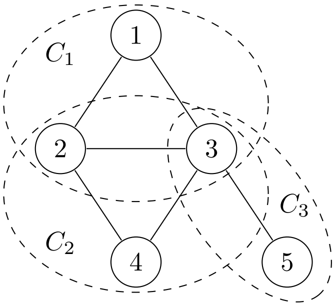


Figure 4.11 An example of an ordering of nodes and cliques in a graph. In this case, the nodes have a perfect ordering, and the ordering of the cliques satisfies the running intersection property.

In general, if we need to instantiate n variables to transform a multi-connected Bayesian network to a polytree, propagation must be performed for all the combinations of values of the instantiated variables. If each variable has k values, the number of propagations is k^n. The procedure is basically the same as described above for one variable with increased complexity.

4.2.4 Junction tree algorithm

The *junction tree* method is based on a transformation of the Bayesian network to a junction tree, where each node in this tree is a group or cluster of variables from the original network. Probabilistic inference is performed over this new representation. Before going into the details about the algorithm, we first introduce some background knowledge about graph theory.

Complete graph and cliques

A *complete graph* is a graph, in which each pair of nodes is adjacent, i.e., there is an edge between each pair of nodes. A *complete set* is a subset of some graph G that induces a complete subgraph of G. It is a subset of vertices of G so that each pair of nodes in this subgraph is adjacent. A *clique* C is a subset of graph G such that it is a complete set that is maximal, i.e., there is no other complete set in G that contains C.

Ordering and triangulation

An *ordering* of the nodes in a graph consists in assigning an integer to each node. Given a graph $G = (V, E)$, with n vertices, then $\alpha = (V_1, \dots, V_n)$ is an ordering of the graph, where V_i is before V_j according to this ordering if $i < j$. An ordering α of a graph $G = (V, E)$ is a *perfect ordering* if all the adjacent vertices of each vertex V_i that are before V_i are completely connected according to this ordering. That is, for every node V_i, $\mathbf{adj}(V_i) \cap \{V_1, \dots, V_{i-1}\}$ is a complete subgraph of G, where $\mathbf{adj}(V)$ denotes the adjacent nodes of V. Figure 4.11 depicts an example of a perfect ordering.

Consider the set of m cliques of an undirected connected graph G. In an analogous way as an ordering of the nodes, we can define an ordering of the cliques as $\beta = (C_1, \dots, C_m)$. An ordering β of the cliques has the *running intersection property*, if all the common nodes of each clique C_i with previous cliques according to this order are contained in some clique C_j with $j < i$. That is, for every clique C_i with $i > 1$, there exists a clique C_j with $j < i$ such that $C_i \cap \{C_1, \dots, C_{i-1}\} \subseteq C_j$. In this case, we call the clique C_j the parent of clique C_i. It is possible that a clique has more than one parent. The cliques C_1, C_2, and C_3 in Figure 4.11 satisfy the running intersection property. In this example, C_1 is the parent of C_2, and C_1 and C_2 are parents of C_3.

A graph G is *triangulated* if every simple circuit of length greater than three in G has a chord. A chord is an edge that connects two of the vertices in the circuit and that is not part of that circuit. For example, in the triangulated graph shown in Figure 4.11, the circuit formed by the vertices $1 \to 2 \to 4 \to 3 \to 1$ has a chord that connects nodes 2 and 3. Given a graph G, satisfying the triangulation property is a condition for achieving a perfect ordering of the vertices, and having an ordering of the cliques that satisfies the running intersection property.

Maximum cardinality search

Given that a graph is triangulated, the maximum cardinality search algorithm provides a perfect ordering of the nodes. Let $G = (V, E)$ be an undirected graph with n vertices, we first select any node from V and assign it index 1. Then from all the non-indexed vertices, select the one with higher number of adjacent indexed vertices and assign it the next number, until all nodes in G have been numbered. If there are multiple nodes with the same number of adjacent indexed vertices, we can choose from one of them randomly for the current index.

Given a perfect ordering of the vertices, it is easy to number the cliques so the order satisfies the running intersection property. For this, the cliques are numbered in inverse order. Given a set of m cliques, the clique that has the node with the highest index is assigned m, and the clique that includes the next highest indexed node is assigned $m - 1$, until all cliques are numbered. This method can be illustrated with the example in Figure 4.11. The node with the highest number is 5, so the clique that contains it is C_3. The next highest node is 4, so the clique that includes it is C_2. The remaining clique is C_1.

Junction tree algorithm

The intuition behind the junction tree method is based on transforming a Bayesian network (which is a directed graph) to an undirected graph, and then clustering the variables into cliques so that the resulting graph is singly connected. Finally the belief propagation can be performed on the resulting singly connected network.

The transformation proceeds as follows:

1. Eliminate the directionality of the arcs.
2. Moralize the graph by adding an arc between pairs of nodes with common children, and add additional arcs if necessary to make the graph triangulated.
3. Order the nodes in the graph with maximum cardinality search.
4. Obtain and order the cliques of the graph such that the order satisfies the running intersection property.
5. Build a junction tree according to the clique ordering.

Figure 4.12 shows an example of transforming a Bayesian network to a junction tree. Those common variables of neighbor cliques in the junction tree are called separators. Given the relevance of these separators, the junction tree is usually drawn including the separator nodes, depicted as rectangles.

Once the junction tree is built, inference is based on probability propagation over the junction tree, in an analogous way as for tree-structured Bayesian networks. In practice, the belief propagation process on a junction tree is divided into two stages: preprocessing and propagation. In the preprocessing phase, the potentials ψ of each clique are obtained through the following steps:

1. Determine the set of variables for each clique C_i.

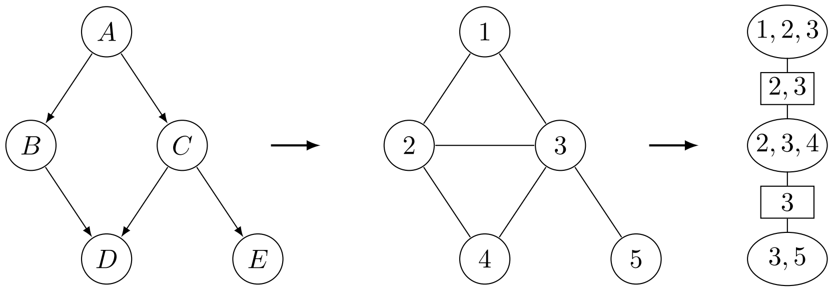


Figure 4.12 Transformation of the Bayesian network in Figure 4.8 to a junction tree. The ellipse nodes represent cliques and the rectangle nodes are the separators.

2. Determine the set of variables that are shared with the previous (parent) clique, i.e., the separators S_i.
3. Determine the set of other variables R_i that are in C_i but not in S_i.
4. Calculate the potential of each clique as $\psi(C_i) = \prod_{X \in R_i} \mathbf{P}(X \mid \mathbf{pa}(X))$.

The propagation phase proceeds in a similar way to belief propagation for trees, by propagating λ messages bottom-up and μ messages top-down:

- *Bottom-up propagation.*
  1. Start from the leaf clique, calculate the λ message to send to the parent clique: $\lambda(C_i) = \sum_{R_i} \psi(C_i)$.
  2. Update the potential of each clique with the λ messages from its children: $\psi'(C_j) = \lambda(C_i)\psi(C_j)$.
  3. Repeat the previous two steps until reaching the root clique, and obtain $\mathbf{P}(C_{\text{root}}) = \psi'(C_{\text{root}})$.
- *Top-down propagation.*
  1. Start from the root clique, calculate the μ message to send to each child node C_i by its parent C_j: $\mu(C_i) = \sum_{C_j - S_i} \mathbf{P}(C_j)$.
  2. Update the potential of each clique when receiving the μ message from its parent and obtain: $\mathbf{P}(C_i) = \psi'(C_i)\frac{\mu(C_i)}{\lambda(C_i)}$.
  3. Repeat the previous two steps until reaching the leaf nodes in the junction tree.

At the end of this propagation in both directions, each clique has the joint marginal probability of the variables that conform it. Hence, the marginal posterior probabilities of each variable can be obtained from the clique via marginalization.

Example 4.6 *Junction tree algorithm.* Consider the junction tree in Figure 4.12, the preprocessing phase is obtaining:

$$\begin{array}{lll}
C_1 = \{A, B, C\} & C_2 = \{B, C, D\} & C_3 = \{C, E\} \\
S_1 = \emptyset & S_2 = \{B, C\} & S_3 = \{C\} \\
R_1 = \{A, B, C\} & R_2 = \{D\} & R_3 = \{E\} \\
\psi(C_1) = \mathbf{P}(A)\mathbf{P}(B \mid A)\mathbf{P}(C \mid A) & \psi(C_2) = \mathbf{P}(D \mid B, C) & \psi(C_3) = \mathbf{P}(E \mid C).
\end{array}$$

Then in the propagation phase, first C_3 sends a λ message to C_2:

$$\lambda(C_3) = \sum_E \psi(C_3) = \sum_E \mathbf{P}(E \mid C),$$

and the potential of C_2 is updated as:

$$\psi'(C_2) = \psi(C_2)\lambda(C_3) = \mathbf{P}(D \mid B, C) \sum_E \mathbf{P}(E \mid C).$$

Next, the λ message from C_2

$$\lambda(C_2) = \sum_D \psi'(C_2) = \sum_D \mathbf{P}(D \mid B, C) \sum_E \mathbf{P}(E \mid C)$$

is sent to C_1, and the potential of C_1 is updated as:

$$\psi'(C_1) = \psi(C_1)\lambda(C_2) = \mathbf{P}(A)\mathbf{P}(B \mid A)\mathbf{P}(C \mid A) \sum_D \mathbf{P}(D \mid B, C) \sum_E \mathbf{P}(E \mid C).$$

Finally we assign $\mathbf{P}(C_1) = \psi'(C_1)$, which ends the bottom-up propagation. Note that the obtained probability $\mathbf{P}(C_1)$ is equivalent to the joint marginal probability of the variables in clique C_1 since

$$\begin{aligned}
\mathbf{P}(C_1) &= \psi'(C_1)\\
&= \mathbf{P}(A)\mathbf{P}(B \mid A)\mathbf{P}(C \mid A) \sum_D \mathbf{P}(D \mid B, C) \sum_E \mathbf{P}(E \mid C)\\
&= \sum_{D,E} \mathbf{P}(A)\mathbf{P}(B \mid A)\mathbf{P}(C \mid A)\mathbf{P}(D \mid B, C)\mathbf{P}(E \mid C)\\
&= \sum_{D,E} \mathbf{P}(A, B, C, D, E)\\
&= \mathbf{P}(A, B, C).
\end{aligned}$$

Now starts the top-down propagation. Clique C_1 sends μ message

$$\mu(C_2) = \sum_{C_1 - S_2} \mathbf{P}(C_1) = \sum_A \mathbf{P}(A, B, C)$$

to clique C_2, and we obtain

$$\begin{aligned}
\mathbf{P}(C_2) &= \psi'(C_2)\frac{\mu(C_2)}{\lambda(C_2)}\\
&= \frac{\mathbf{P}(D \mid B, C) \sum_E \mathbf{P}(E \mid C) \sum_A \mathbf{P}(A, B, C)}{\sum_D \mathbf{P}(D \mid B, C) \sum_E \mathbf{P}(E \mid C)}\\
&= \mathbf{P}(D \mid B, C)\mathbf{P}(B, C)\\
&= \mathbf{P}(B, C, D),
\end{aligned}$$

which is also equivalent to the joint marginal probability of the variables in clique C_2. Finally, clique C_2 sends μ message

$$\mu(C_3) = \sum_{C_2 - S_3} \mathbf{P}(C_2) = \sum_{B,D} \mathbf{P}(B, C, D)$$

to clique C_3, and we get

$$\begin{aligned}
\mathbf{P}(C_3) &= \psi'(C_3)\frac{\mu(C_3)}{\lambda(C_3)}\\
&= \frac{\mathbf{P}(E \mid C) \sum_{B,D} \mathbf{P}(B, C, D)}{\sum_E \mathbf{P}(E \mid C)}\\
&= \mathbf{P}(E \mid C)\mathbf{P}(C)\\
&= \mathbf{P}(C, E).
\end{aligned}$$

This ends the top-down propagation.

After obtaining the joint marginal probability of the variables for all cliques, the single variable's probabilities can be calculated via marginalization:

$$\begin{aligned}
&\mathbf{P}(A) = \textstyle\sum_{B,C} \mathbf{P}(C_1) \quad \mathbf{P}(B) = \sum_{A,C} \mathbf{P}(C_1) \quad \mathbf{P}(C) = \sum_{A,B} \mathbf{P}(C_1)\\
&\mathbf{P}(D) = \textstyle\sum_{B,C} \mathbf{P}(C_2)\\
&\mathbf{P}(E) = \textstyle\sum_{C} \mathbf{P}(C_3).
\end{aligned}$$

4.2.5 Sampling based methods

Stochastic simulation algorithms consist in simulating the Bayesian several times, where each simulation gives a sample value for all non-instantiated variables. Then the posterior probability of each variable is approximated in terms of the frequency of each value in the sample space. This process gives an estimate of the posterior probability which depends on the number of samples. However, the computational cost is not affected by the complexity of the network.

Logic sampling

Logic sampling is a basic stochastic simulation algorithm that generates samples according to the following procedure:

1. Generate sample values for all root nodes of the Bayesian network according to their prior probabilities $\mathbf{P}(X)$.
2. Generate samples for the children of the sampled nodes, according to their conditional probabilities $\mathbf{P}(X \mid \mathbf{pa}(X))$.
3. Repeat the second step until all leaf nodes are reached.

The previous procedure is repeated n times to generate n samples. Then the probability of possible values of each variable is estimated as the frequency that the value occurs in the n samples, i.e.,

$$\mathbf{P}(X = x_k) = \frac{1}{n} \sum_{i=1}^{n} I_{x_k}(x_i), \tag{4.23}$$

where the indicator function $I_{x_k}(x_i) = 1$ if $x_k = x_i$, and 0 otherwise.

The direct application of the previous procedure gives an estimate of the marginal probabilities of all the variables when there is no evidence. If there is evidence, i.e., some variables are instantiated, all samples that are not consistent with the evidence are discarded and the posterior probabilities are estimated from the remaining samples.

A disadvantage of logic sampling when evidence exists is that the data-efficiency is very low since many samples have to be discarded. This implies that a larger number of samples are required to have a relatively good estimation.

Likelihood weighting

Likelihood weighting generates samples in the same way as logic sampling, however when there is evidence the non-consistent samples are not discarded. Instead, each sample is given a weight according to the weight of the evidence for this sample. Given a sample of all non-instantiated nodes H and the evidence variables E, the weight of sample i is calculated as:

$$w_i = \mathbf{P}(E \mid H_i), \tag{4.24}$$

then the posterior probability of possible values of each variable is estimated as a weighted average over all n samples:

$$\mathbf{P}(X = x_k) = \frac{\sum_{i=1}^{n} w_i I_{x_k}(x_i)}{\sum_{i=1}^{n} w_i}. \tag{4.25}$$

4.3 Parameter learning

When the network structure is known, parameter learning of a Bayesian network consists in estimating the conditional probability tables from data. If we have sufficient and complete data for all the variables, the conditional probability table for each variable can be estimated based on the frequency of each value or combination of values via maximum likelihood estimation (MLE). For example, to estimate the conditional probability table of variable C with two parents A and B given the observed n samples, each entry of the table can be estimated according to

$$\mathbf{P}(C = c_k \mid A = a_i, B = b_j) = \frac{\sum_{i'=1}^{n} I_{a_i,b_j,c_k}(a_{i'}, b_{i'}, c_{i'})}{\sum_{i'=1}^{n} I_{a_i,b_j}(a_{i'}, b_{i'})} \tag{4.26}$$

with indicator function I. However, it can sometimes happen that we do not have a perfect dataset for learning the parameters.

4.3.1 Smoothing

When we estimate probabilities from data, it can sometimes happen that a particular event never occurs in the data set. This leads to the corresponding probability value being zero, implying an impossible case. Hence, if in the inference process this probability is considered, it will also make the result zero. This situation occurs, in many cases, because there is insufficient data to have a robust estimate of the parameters, and not because it is really an impossible event. This problem can be addressed by using some type of smoothing for the probabilities, eliminating zero probability values. From a Bayesian point of view, smoothing corresponding to estimate the posterior distribution of the parameters given some priors.

Uniform prior

One of the most common and simplest prior one can consider is a uniform distribution on the variable domain, which is called *Laplacian smoothing* (or *additive smoothing*). Consider a discrete variable X with m possible values. Given a dataset with n samples, the estimation of its probability will be the following:

$$\mathbf{P}(x_i) = \frac{\alpha + \sum_{i'=1}^{n} I_{x_i}(x_{i'})}{\alpha m + n}, \quad i = 1, \dots, m, \tag{4.27}$$

where I is the indication function of x_i and α is the smoothing parameter, with $\alpha = 0$ corresponding to no smoothing. When there is no observed sample, i.e., $n = 0$, the estimated parameter distribution is simply the uniform prior

$$\mathbf{P}(x_i) = \frac{1}{m}, \quad i = 1, \dots, m.$$

As the number of observed samples increases, the parameter estimation will converge to the true data distribution since in this case,

$$\lim_{n\to\infty} \mathbf{P}(x_i) = \lim_{n\to\infty} \frac{\alpha + \sum_{i'=1}^{n} I_{x_i}(x_{i'})}{\alpha m + n} = \frac{\sum_{i'=1}^{n} I_{x_i}(x_{i'})}{n},$$

for all $i = 1, \dots, m$.

Beta prior

Other than the non-informative uniform prior, we can try to integrate more expert information into the parameter estimation by considering an informative prior. For binary variables, we can build a prior based on the expectations of a beta distribution, $\mathrm{Beta}(\alpha, \beta)$. The expected value of random variable $X \sim \mathrm{Beta}(\alpha, \beta)$ is controlled by the *shape parameters* α and β, that is

$$\mathbf{E}_{\mathrm{Beta}(\alpha,\beta)}[X] = \mathbf{P}(X = 1 \mid \alpha, \beta) = \frac{\alpha}{\alpha + \beta}. \tag{4.28}$$

Then similar to (4.27), given a dataset with n samples, the estimation of binary variable X can be obtained as

$$\mathbf{P}(X = 1) = \frac{\alpha + \sum_{i'=1}^{n} I_1(x_{i'})}{\alpha + \beta + n}, \tag{4.29}$$

and $\mathbf{P}(X = 0) = 1 - \mathbf{P}(X = 1)$. In this case, the fraction $\frac{\alpha}{\alpha+\beta}$ determines the expert's prior for $X = 1$, while the denominator $\alpha + \beta$ defines the confidence about the prior. This means that a higher $\alpha + \beta$ value assigns more confidence on the prior while a lower value places more weight on the observations.

Example 4.7 *Smoothing with beta prior.* Assuming that an expert gives an estimate of $\mathbf{E}_{\mathrm{Beta}(\alpha,\beta)}[X] = 0.7$ for a certain parameter, and that the experimental data provides 40 positive cases among 100 samples. The parameter estimation for different confidences assigned to the expert will be the following:

- Low confidence ($\alpha + \beta = 10$): $\mathbf{P}(X = 1) = \frac{7+40}{10+100} = 0.43$.
- Medium confidence ($\alpha + \beta = 100$): $\mathbf{P}(X = 1) = \frac{70+40}{100+100} = 0.55$.

- High confidence ($\alpha + \beta = 1000$): $\mathbf{P}(X = 1) = \frac{700+40}{1000+100} = 0.67$.

In the first case, the expert prior is dominated by the data, while in the third case the probability is closer to the expert's estimation, and the second case provides a compromise between them.

Dirichlet prior

The idea of the previous beta prior can be extended to general m-valued random variables by replacing the beta distribution with a Dirichlet distribution, $\mathrm{Dir}(\alpha)$, with the *concentration parameter* α being an m-vector. The expected value of each entry of an m-dimensional random vector $X \sim \mathrm{Dir}(\alpha)$ is

$$\mathbf{E}_{\mathrm{Dir}(\alpha)}[X_i] = \mathbf{P}(x_i \mid \alpha) = \frac{\alpha_i}{\alpha^T \mathbf{1}}, \quad i = 1, \dots, m, \tag{4.30}$$

and we also have

$$\sum_{i=1}^{m} \mathbf{E}_{\mathrm{Dir}(\alpha)}[X_i] = 1. \tag{4.31}$$

We can view the m-dimensional random vector as a m-valued discrete random variable by assigning each of the entries an value. Hence, with the given n observations, we can estimate the probability distribution of X as

$$\mathbf{P}(x_i) = \frac{\alpha_i + \sum_{i'=1}^{n} I_{x_i}(x_{i'})}{\alpha^T \mathbf{1} + n}, \quad i = 1, \dots, m. \tag{4.32}$$

Similar to the beta prior, the fraction $\alpha_i/\alpha^T\mathbf{1}$ determines the expert's prior for $X = x_i$, and the confidence about the prior is controlled by the value of $\alpha^T\mathbf{1}$.

4.3.2 Missing data

Another common situation that we may have during parameter learning is to have incomplete data. There are two basic cases:

- Missing values: in some samples there are missing values for one or more variables.
- Hidden nodes: a variable or set of variables in the model cannot be observed.

For dealing with missing values, there are several alternatives. One trivial approach would be to remove all the samples with missing values. This would be acceptable only if there is sufficient data. Another alternative without removing any samples from the dataset is to substitute the missing value by the most common value of that variable based on all available observations. However, this may bias the model since the information from the other variables is not taken into account. In general the best option would be to estimate the missing value based on the values of the other variables in the corresponding sample. In this case, we first learn the parameters of the Bayesian network based on the samples with complete observations, and then for each sample with missing values applying the following process:

1. Instantiate all the known variables in the sample.
2. Through probabilistic inference obtain the posterior probabilities of the missing variables.
3. Assign to each unknown variable the value with highest posterior probability, or sample one value according to the posterior probability.
4. Add this completed sample to the database.

Finally we can re-estimate the model parameters based on the completed dataset.

For hidden nodes, the approach to estimate their parameters is based on the expectation-maximization (EM) algorithm. The algorithm starts by initializing the missing parameters with random values. Then for each EM-iteration, in the E-step, the missing data values are estimated based on the current parameters; in the M-step, the parameters are updated based on the estimated data. This will be repeated multiple times until the parameters get converge.

4.3.3 Discretization

Usually Bayesian networks consider discrete or categorical variables. Although there are some developments for continuous variables, these are restricted to certain distributions, in particular Gaussian variables and linear relations. An alternative to include continuous variables in Bayesian networks is to discretize them, i.e., transform them to categorical variables. Discretization methods can be unsupervised and supervised.

Unsupervised discretization

Unsupervised discretization do not consider the task for which the model is going to be used, meaning that the discretization intervals for each variable are determined independently. The two main types of unsupervised discretization approaches are *equal width* and *equal data*. Equal width consists in dividing the range of a variable into k equal bins, such that each bin has a size of $(\sup(X) - \inf(X))/k$. The number of intervals k is usually assigned by the user. Equal data divides the range of the variable into k intervals, such that each interval includes the same number of data points from the training dataset. In other words, if there are n samples, each interval will contain n/k data points, but the intervals will not necessarily have the same width.

Supervised discretization

Supervised discretization incorporates the task to be performed with the model, such that the variables are discretized to optimize this task, for instance classification accuracy. This can be posed as an optimization problem. For example, consider a continuous feature variable X and a categorical class variable C. Given n training samples with each one having a value for C and X, the problem is to determine the optimal partition of $(\inf(X), \sup(X))$ such that the classification accuracy is maximized. This is then a combinatorial optimization problem that is computationally complex. Different search approaches can be used, including basic ones such as *hill-climbing* or more sophisticated methods like *simulated annealing* and *genetic algorithms*.

4.4 Structure learning

4.4.1 Tree learning

For a particular case where the dependencies between random variables can be represented with a tree-structure, the structure learning procedure can be separated into two steps: (1) learning the skeleton of the tree, i.e., establishing undirected edges between variables, and (2) determining the direction of the edges.

Skeleton learning

The *Chow-Liu procedure* (CLP) obtains the skeleton of a tree, but does not provide the directions of the arcs. Given a set of n random variables $X = \{X_1, \ldots, X_n\}$, we would like to find the tree structure that best approximate the joint distribution of these variables $\mathbf{P}(x)$. Let $\widetilde{\mathbf{P}}(x)$ be the approximated joint distribution obtained from some tree including these variables, the skeleton learning problem consists in minimizing the distance between distributions $\mathbf{P}(x)$ and $\widetilde{\mathbf{P}}(x)$, according to the *KL-divergence* measure:

$$D_{\mathrm{KL}}(\mathbf{P}, \widetilde{\mathbf{P}}) = \sum_{x \in X} \mathbf{P}(x) \log \left(\frac{\mathbf{P}(x)}{\widetilde{\mathbf{P}}(x)} \right). \tag{4.33}$$

However, evaluating the KL-divergence for all possible trees is very expensive. As an alternative of the objective (4.33), let

$$I(X_i, X_j) = \sum_{x_i \in X_i, x_j \in X_j} \mathbf{P}(x_i, x_j) \log \left(\frac{\mathbf{P}(x_i, x_j)}{\mathbf{P}(x_i)\mathbf{P}(x_j)} \right) \tag{4.34}$$

be the *mutual information* between any pair of variables $X_i \in X$, $X_j \in X$, we define the weight $W(X)$ as the sum of the mutual information of the edges that constitute the tree $G = (X, E)$ as

$$W(X) = \sum_{(X_i, X_j) \in E} I(X_i, X_j) = \sum_{i=1}^{n-1} I(X_i, \mathbf{pa}(X_i)). \tag{4.35}$$

(Here we assume the root note is indexed as X_n in the tree without loosing of generality.) It can be shown that minimizing (4.33) is equivalent to maximizing (4.35) over the set of edges E. (See exercise 4.4.) Therefore, obtaining the optimal tree is equivalent to finding the *maximum weight spanning tree*, according to the following procedure:

1. Obtain the mutual information $I(X_i, X_j)$ for all pairs of variables $X_i \in X$, $X_j \in X$.
2. Order the mutual information values in descending order.
3. Select the pair (X_i, X_j) with maximum $I(X_i, X_j)$ and connect the two variables with an edge, this constitutes the initial tree.
4. Add the pair with the next highest mutual information to the tree if they do not make a cycle, otherwise skip it and continue with the following pair.
5. Repeat the previous step until all the variables are in the tree.

Direction learning

Based on the learnt skeleton of the tree from CLP, one trivial way of assigning edge directions would be randomly select one node as the tree root and assign directions to the edges starting from this root. Another option is to obtain directions using external semantics, or using higher order dependency tests. We will introduce the last alternative subsequently.

As discussed in §4.1.1, given three variables X, Y, and Z, there are three possibilities for their dependency:

- Sequential: $X \rightarrow Y \rightarrow Z$.
- Divergent: $X \leftarrow Y \rightarrow Z$.
- Convergent: $X \rightarrow Y \leftarrow Z$.

The first two cases are indistinguishable under statistical independence testing, since in both cases X and Z are independent given Y. However the third case is different, where X and Z are not independent given Y. Hence, this case can be used to determine the directions of the two arcs that connect these three variables, and once we have identified a convergent structure, we can apply this knowledge to learn the directions of other arcs using independence tests. With this, the following algorithm can be used for learning the direction of a tree skeleton:

1. Iterate over the network until a convergent variable triplet is found. We will call the variable to which the arcs converge a *multi-parent node.*
2. Starting with a multi-parent node, determine the directions of other arcs using independence tests for variable triplets. Continue this procedure until it is no longer possible.
3. Repeat the first two steps until no other directions can be determined.

However, we should note that there is no guarantee that the direction for all the arcs in the tree can be obtained via this procedure. If any arcs are left undirected when the algorithm quits, external semantics can be used to infer their directions.

Example 4.8 *Direction learning using independence tests.* Given a tree skeleton as shown in Figure 4.13, we first perform statistical independence tests for variable triplet $\{X_1, X_2, X_4\}$. Suppose we find that X_2 are dependent of X_4 given X_1, meaning the variable triplet $\{X_1, X_2, X_4\}$ falls into the convergent substructure. Hence, we can assign the direction for edge (X_1, X_2) and (X_1, X_4) as $X_2 \rightarrow X_1 \leftarrow X_4$. Based on this knowledge, we can further test the independence relationship between triplet $\{X_1, X_2, X_3\}$ and $\{X_1, X_3, X_4\}$. If

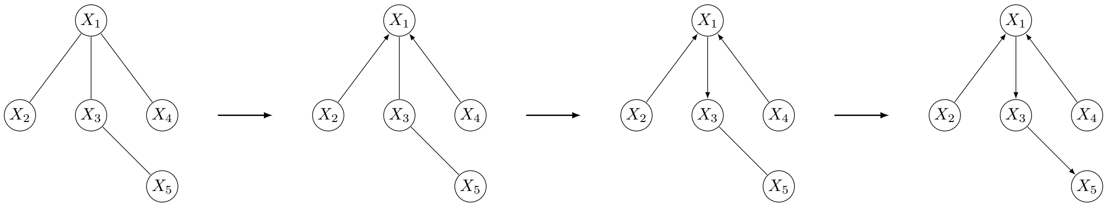


Figure 4.13 An example of learning the edge directions given a tree skeleton using independence tests.

we have $(X_2 \perp\!\!\!\perp X_3 \mid X_1)$ and $(X_3 \perp\!\!\!\perp X_4 \mid X_1)$, the direction of edge (X_1, X_3) must be $X_1 \to X_3$. Otherwise, both the variable triplets $\{X_1, X_2, X_3\}$ and $\{X_1, X_3, X_4\}$ fall into convergent substructure and we can thus assign the edge direction $X_1 \leftarrow X_3$. Finally, the same process can be applied for triplet $\{X_1, X_3, X_5\}$ to determine the direction of edge (X_3, X_5).

4.4.2 Score-based methods

Score-based methods view the structure learning problem as a combinatorial optimization problem w.r.t. some type of scoring function over the network structure. Since this problem is generally NP-hard, we resort to heuristic search techniques, such as hill-climbing, simulated annealing, genetic algorithms, etc. As can be expected, one of the most important decisions we must make in this framework is the choice of scoring function S.

Likelihood score

The likelihood score is defined as the probability of observing the dataset $\mathcal{D}$ given the graph G and its parameters θ_G, typically expressed in logarithmic form:

$$S_{\text{LL}}(G) = l_{\mathcal{D}}(\theta_G) = \log \mathbf{P}(\mathcal{D} \mid \theta_G, G). \tag{4.36}$$

The network structure found by directly maximizing the log-likelihood score (MLE estimation) can be highly complex, which usually implies overfitting the data (poor generalization) and also makes inference more complex.

Bayesian score

An alternative metric which avoids overfitting is expressed by following a Bayesian approach, obtaining the posterior probability of the structure given the data with the Bayes rule:

$$\mathbf{P}(G \mid \mathcal{D}) = \frac{\mathbf{P}(\mathcal{D} \mid G)\mathbf{P}(G)}{\mathbf{P}(\mathcal{D})}. \tag{4.37}$$

Since the denominator $\mathbf{P}(\mathcal{D})$ is a constant that does not depend on the structure, it can be discarded from the metric. Thus we define the Bayesian score as:

$$S_{\text{B}}(G) = \log \mathbf{P}(\mathcal{D} \mid G) + \log \mathbf{P}(G), \tag{4.38}$$

which is again, for convenience, expressed in logarithmic form. The probability $\mathbf{P}(G)$ is the prior over network structures, allowing us to prefer some structures over others. The likelihood term $\mathbf{P}(\mathcal{D} \mid G)$ is called the marginal likelihood of the data since its obtained by marginalizing out the unknown model parameters θ_G:

$$\mathbf{P}(\mathcal{D} \mid G) = \int_{\theta_G} \mathbf{P}(\mathcal{D} \mid \theta_G, G)\mathbf{P}(\theta_G \mid G)\ d\theta_G, \tag{4.39}$$

where $\mathbf{P}(\mathcal{D} \mid \theta_G, G)$ is the likelihood of the data given the network G and its parameter θ_G, and $\mathbf{P}(\theta_G \mid G)$ is the prior distribution over different parameter values for the network G.

It is important to realize that maximizing the marginal likelihood (4.39) is quite different from maximizing the likelihood score (4.36). Both terms examine the likelihood of the data given the structure. The maximum likelihood score evaluates the likelihood of the training data using the best parameter values for the given dataset. This estimate is realistic only if these parameters are also reflective of the data in general, a situation that never occurs. In contrast, in the Bayesian approach, by integrating $\mathbf{P}(\mathcal{D} \mid \theta_G, G)$ over the different choices of parameters θ_G, we are measuring the expected likelihood, averaged over different possible choices of θ_G. Thus, we are being more conservative in our estimate of the goodness of the model, which contributes to avoid overfitting.

In practice, the value of the marginal likelihood $\mathbf{P}(\mathcal{D} \mid G)$ depends on the parameter prior $\mathbf{P}(\theta_G \mid G)$ that we select, as well as the number of samples in the dataset, etc. For example, if we use a Dirichlet parameter prior for all parameters in the network, then, when the number of samples in the dataset $n \to \infty$, we have the *Bayesian information criterion* (BIC):

$$\begin{aligned} S_{\mathrm{BIC}}(G) &= l_{\mathcal{D}}(\theta_G) - \frac{k}{2}\log n \\ &= \log \mathbf{P}(\mathcal{D} \mid \theta_G, G) - \frac{k}{2}\log n, \end{aligned} \tag{4.40}$$

where k is the number of parameters in the model and n is the number of samples in the dataset. From this example, we can see that the Bayesian score seems to be biased toward simpler structures, but as it gets more data, it is willing to recognize that a more complex structure is necessary. In other words, it appears to trade off fit to data with model complexity, thereby reducing the extent of overfitting.

4.4.3 PC algorithm

The *PC algorithm* first recovers the skeleton (underlying undirected graph) of the Bayesian network, and then it determines the direction of the edges. To determine the skeleton, it starts from a fully connected undirected graph, and determines the conditional independence of each pair of variables given some subset of the other variables. For this it assumes that there is a procedure that can determine if two variables, X and Y, are independent given a subset of variables Z. For example, using statistical tests or by calculating the *conditional cross-entropy measure*. If the independence measure is below some threshold value set according to a certain confidence level, the edge between the pair of variables is eliminated. These tests are iterated for all pairs of variables in the graph. Then in the second step, where the direction of the edges are determined, the same independence testing procedure based on variable triplets as introduced in §4.4.1 can be applied.

If the set of independencies are *faithful* to a graph, meaning that that conditional independence relations are due to the causal structure rather than because of accidents in parameter values, and the independence tests are perfect, the algorithm produces a graph equivalent to the original one.

Bibliography

An introduction to Bayesian networks is given in the classic book by Pearl [Pea88]. The books by Jensen and Nielsen [JN07] and by Neapolitan [Nea90] also includes some useful information on Bayesian networks. A more recent account with emphasis on modeling, inference, and complexity analysis is given in [Dar09]. Other books with more emphasis on applications are [PNM+08] and [KN10].

A formal definition about *d*-separation, including some useful properties and corresponding mathematical analysis can be found in [Pea09, §1.2].

An overview of canonical models is presented in the article [DD06].

The belief propagation algorithm was initially proposed in [Pea86] for the inference of singly connected Bayesian networks. This idea can be generalized to multi-connected networks, which is called the *loopy propagation* algorithm. It has been found empirically that for certain structures this algorithm converges to the true posterior probabilities, but for other structures that it does not converge, it can only provide approximate solution of the inference problem [MWJ13].

The computational complexity in terms of space and time of the variable elimination algorithm is determined by the size of the factors. The book [Dar09] can be referred to for some detailed information.

The book [Pea88] provides a detailed introduction about conditioning algorithms. Several variants of the conditioning algorithm have been proposed, including local conditioning [Díe96] and recursive conditioning [Dar01].

In this chapter we do not include specific algorithms for finding the minimal triangulations of graphs. The readers can refer to the survey by Heggernes [Heg06] for more information.

The junction tree algorithm was initially introduced in [LS88]. There are two main variations on the junction tree algorithm, which are known as the Hugin [JA13] and Shafer-Shenoy [SS90] architectures. The description in this chapter is based on the Hugin architecture. The main differences between them are in the information they store, and in the way they compute the messages. These differences have implications in their computational complexity. In general, the Shafer-Shenoy architecture will require less space but more time.

In this chapter we introduced inference methods for discrete-valued Bayesian networks. When dealing with continuous variables, probabilistic inference techniques have been developed for some distribution families, in particular Gaussian variables [Pea88].

For the equal width discretization method, specially for Bayesian classifiers, one useful way to determine the number of intervals is called *proportional k-interval discretization* (PKID) [YW01]. This strategy seeks a trade-off between the bias and variance of the parameter estimates by adjusting the number and size of intervals to the number of training instances. Given a continuous variable with n training instances, it is discretize to $\sqrt{n}$ intervals, with $\sqrt{n}$ instances in each interval. This method was compared empirically with other discretization methods for learning naive Bayesian classifiers, where PKID achieves the lowest mean error.

For heuristic methods solving combinatorial optimization problems, the books [PS98] and [RN16] provide a in depth introduction about the hill-climbing algorithm and its variants. Information about simulated annealing method can be found in [KGJV83]. The paper from Holland [Hol73] can be referred to for more details about the class of genetic algorithms.

The Chow-Liu procedure was first described in the paper [CL68]. Pearl [Pea88] provides a modern analysis of the algorithm as a Bayesian network. The statistical independence test based algorithm for assigning directions given a tree skeleton was initially introduced in [RP87].

Except the Bayesian information criterion that we introduced in this chapter, there are many other Bayesian scores based on different parameter priors and assumptions that have been widely used for structure learning, including the Bayesian-Dirichlet (BD) score [HGC95], BDe score ('e' for likelihood-equivalence) [HGC95], BDeu score ('u' for uniform joint distribution [Bun91]), and K2 score [CH92].

For a detailed discussion about the PC algorithm, and the *faithfulness condition* of a set of independencies to a graph, one can referred to [SGS01].

Exercises

4.1 *Graphoid axioms.* Using the property of d-separation, illustrate graphically that the following graphoid axioms introduced in §1.1.1 are true.

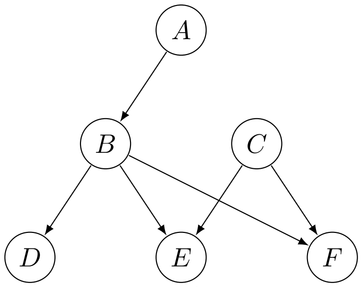
(a) $(X \perp\!\!\!\perp Y \mid Z) \implies (Y \perp\!\!\!\perp X \mid Z)$.

(b) $(X \perp\!\!\!\perp YW \mid Z) \implies (X \perp\!\!\!\perp Y \mid Z)$.

(c) $(X \perp\!\!\!\perp YW \mid Z) \implies (X \perp\!\!\!\perp Y \mid ZW)$.

(d) $(X \perp\!\!\!\perp Y \mid Z)$ & $(X \perp\!\!\!\perp W \mid ZY) \implies (X \perp\!\!\!\perp YW \mid Z)$.

4.2 *Belief propagation.* Given the following Bayesian network and corresponding conditional probability tables, considering all the variables are binary, obtain the posterior probabilities of all unknown variables via belief propagation with the evidence $A = a_1$, $E = e_2$.



a_1	a_2
0.2	0.8

c_1	c_2
0.6	0.4

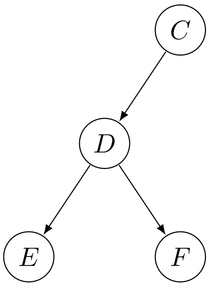
	a_1	a_2
b_1	0.7	0.1
b_2	0.3	0.9

	b_1	b_2
d_1	0.2	0.7
d_2	0.8	0.3

	(b_1, c_1)	(b_1, c_2)	(b_2, c_1)	(b_2, c_2)
e_1	0.8	0.6	0.5	0.1
e_2	0.2	0.4	0.5	0.9

	(b_1, c_1)	(b_1, c_2)	(b_2, c_1)	(b_2, c_2)
f_1	0.4	0.9	0.3	0.5
f_2	0.6	0.1	0.7	0.5

4.3 *Stochastic inference.* Given the following Bayesian network and corresponding conditional probability tables, considering all the variables are binary, implement the logic sampling and likelihood weighting algorithm to estimate the posterior probabilities of all unknown variables with the evidence $E = e_1$.



c_1	c_2
0.8	0.2

	c_1	c_2
d_1	0.9	0.7
d_2	0.1	0.3

	d_1	d_2
e_1	0.9	0.5
e_2	0.1	0.5

	d_1	d_2
f_1	0.7	0.4
f_2	0.3	0.6

4.4 *Chow-Liu procedure.* Given a set of n random variables $X = \{X_1, \ldots, X_n\}$, finding the best tree skeleton to represent the joint distribution $\mathbf{P}(x)$ of these variables consists in minimizing the KL-divergence between the true distribution $\mathbf{P}(x)$ and tree approximated distribution $\widetilde{\mathbf{P}}(x)$:

$$D_{\mathrm{KL}}(\mathbf{P}, \widetilde{\mathbf{P}}) = \sum_{x \in X} \mathbf{P}(x) \log \left(\frac{\mathbf{P}(x)}{\widetilde{\mathbf{P}}(x)} \right).$$

Show that minimizing this KL-divergence is equivalent to maximizing the weight $W(X)$ defined on some tree $G = (V, E)$ as

$$W(X) = \sum_{(X_i, X_j) \in E} I(X_i, X_j) = \sum_{i=1}^{n-1} I(X_i, \mathbf{pa}(X_i)),$$

where we assume the root note is indexed as X_n in the tree, and

$$I(X_i, X_j) = \sum_{x_i \in X_i, x_j \in X_j} \mathbf{P}(x_i, x_j) \log \left(\frac{\mathbf{P}(x_i, x_j)}{\mathbf{P}(x_i)\mathbf{P}(x_j)} \right)$$

is the mutual information between variables $X_i \in X$, $X_j \in X$.
Hint. $\sum_x \mathbf{P}(x) \log \mathbf{P}(x_i) = \sum_{x_i} \mathbf{P}(x_i) \log \mathbf{P}(x_i)$.

Part II

Decision models

Chapter 5

Markov decision problems

5.1 Markov decision processes

A *Markov decision process* (MDP) can be used to model the interaction of an *agent* and an *environment*. At each time step t of the interaction, the agent receives some representation of the environment's *state* $s_t \in \mathcal{S}$, and follows a *policy* π to take an *action* $a_t \in \mathcal{A}$. In response, the environment emits a real-valued reward signal $r(s_t, a_t)$ and enters a new state $s_{t+1} \in \mathcal{S}$. A graphical representation of this process is shown in Figure 5.1. The set $\mathcal{S}$ and $\mathcal{A}$ are called the state space and action space, respectively. The policy is in general stochastic, with $\pi(a \mid s)$ being the probability of choosing action a in state s. We use $\pi(s)$ to denote the conditional probability over $\mathcal{A}$ if the policy is stochastic, or the action it chooses if it is deterministic. The function $r\colon \mathcal{S} \times \mathcal{A} \to \mathbf{R}$ is called the *reward function*. The process at every step is called a *transition*; at time t, it consists of the tuple (s_t, a_t, s_{t+1}), where $a_t \sim \pi(s_t)$ and $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$. Hence, under policy π, the probability of generating a trajectory $\tau = (s_0, a_0, s_1, a_1, \ldots, s_T)$ of length T can be written explicitly as

$$p(\tau) = p(s_0) \prod_{t=0}^{T-1} \pi(a_t \mid s_t) p(s_{t+1} \mid s_t, a_t). \tag{5.1}$$

In general, the state and action sets of an MDP can be discrete or continuous. When both sets are finite, we can represent these functions as lookup tables; this is known as a *tabular representation.*

Note that the field of control theory uses slightly different terminology. In particular, the environment is called the *plant*, and the agent is called the *controller*. States are denoted by $x_t \in \mathcal{X} \subseteq \mathbf{R}^m$, actions are denoted by $u_t \in \mathcal{U} \subseteq \mathbf{R}^n$, and rewards are denoted by costs $c_t \in \mathbf{R}$. In this course we will use the former notation because they are meaningful to a wider audience.

5.1.1 Episodes and returns

The Markov decision process describes how a trajectory $\tau = (s_0, a_0, s_1, a_1, \ldots)$ is stochastically generated. If the agent can potentially interact with the environment forever, we call it a *continuing task.* Alternatively, the agent is in an *episodic task*, if its interaction terminates once the system enters a terminal state or absorbing state (the next state is always itself with 0 reward). After entering a terminal state, the agent will start a new episode from a new initial state $s_0 \sim p(s_0)$. The episode length is in general random. For example, the amount of time a robot takes to reach its goal may be quite variable, depending on the decisions it makes, and the randomness in the environment. Note that we can convert an episodic MDP to a continuing MDP by redefining the transition model in terminal states to be the initial-state distribution $p(s_0)$. Finally, if the trajectory length T in an episodic task is fixed and known, it is called a *finite horizon problem.*

Let τ be a trajectory of length T, where T may be ∞ if the task is continuing. We define the *return* for the state at time t to be the sum of expected rewards obtained going forwards, where each reward is multiplied by a *discount factor* $\gamma \in [0, 1]$:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t-1} r_{T-1}$$

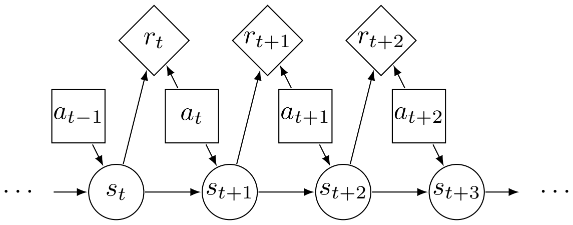


Figure 5.1 Graphical representation of an MDP.

$$= \sum_{k=0}^{T-t-1} \gamma^k r_{t+k} = \sum_{i=t}^{T-1} \gamma^{i-t} r_i. \tag{5.2}$$

For episodic tasks that terminate at time T, we define $G_t = 0$ for $t \geq T$. Clearly, the return satisfies the following recursive relationship:

$$G_t = r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \cdots) = r_t + \gamma G_{t+1}. \tag{5.3}$$

The discount factor γ plays two roles. First, it ensures the return G_t is finite even if $T \to \infty$ (i.e., infinite horizon), if we use $\gamma < 1$ and the rewards r_t are bounded. Second, it puts more weight on short-term rewards, which generally has the effect of encouraging the agent to achieve its goals more quickly. However, if γ is too small, the agent will become too greedy. In the extreme case where $\gamma = 0$, the agent is completely *myopic*, and only tries to maximize its immediate reward. In general, the discount factor reflects the assumption that there is a probability of $1 - \gamma$ that the interaction will end at the next step. For finite horizon problems, where T is known, we can set $\gamma = 1$, since we know the life time of the agent a priori.

5.1.2 Value functions

Let π be a given policy, we define the *state-value function* V, or *value function* for short, as follows:

$$V^\pi(s) = \mathbf{E}_\pi[G_0 \mid s_0 = s] = \mathbf{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, s_0 = s\right], \tag{5.4}$$

for all $s \in \mathcal{S}$, where $\mathbf{E}_\pi$ indicating that actions are selected according to π. This is the expected return obtained if we start in state $s \in \mathcal{S}$ and follow π to choose actions in a continuing task. Similarly, we define the *action-value function* Q, also known as the *Q-function*, as follows:

$$Q^\pi(s,a) = \mathbf{E}_\pi[G_0 \mid s_0 = s, a_0 = a] = \mathbf{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, s_0 = s, a_0 = a\right], \tag{5.5}$$

for all $s \in \mathcal{S}, a \in \mathcal{A}$. The action-value function represents the expected return obtained if we start by taking action a in state s, and then follow π to choose actions thereafter. Finally, we define the *advantage function* as follows:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s), \tag{5.6}$$

which tells us the benefit of picking action a in state s then switching to policy π, relative to the baseline return of always following π. Note that $A^\pi(s,a)$ can be both positive and negative, and $\mathbf{E}_{\pi(s|a)}[A^\pi(s,a)] = 0$ since

$$V^\pi(s) = \mathbf{E}_{\pi(a|s)}[Q^\pi(s,a)]. \tag{5.7}$$

A fundamental property of value functions is that they satisfy recursive relationships similar to that which we have already established for the return (5.3). For any policy π and any state s, the following consistency condition holds between the value of s and the value of its possible successor states:

$$V^\pi(s) = \mathbf{E}_\pi[G_0 \mid s_0 = s]$$

$$
\begin{aligned}
&= \mathbf{E}_\pi[r_0 + \gamma G_1 \mid s_0 = s] && \text{(by (5.3))} \\
&= \sum_{a \in \mathcal{A}} \pi(a \mid s) \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \mathbf{E}_\pi[G_1 \mid s_1 = s'] \right] \\
&= \sum_{a \in \mathcal{A}} \pi(a \mid s) \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^\pi(s') \right]. && (5.8)
\end{aligned}
$$

The equation (5.8) is called the *Bellman equation* for value function V^π. It expresses a relationship between the value of a state and the values of its successor states. Similarly, we have the Bellman equation for action-value function Q^π as

$$
\begin{aligned}
Q^\pi(s,a) &= \mathbf{E}_\pi[G_0 \mid s_0 = s, a_0 = a] \\
&= \mathbf{E}_\pi[r_0 + \gamma G_1 \mid s_0 = s, a_0 = a] && \text{(by (5.3))} \\
&= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \mathbf{E}_\pi[G_1 \mid s_1 = s'] \\
&= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \sum_{a' \in \mathcal{A}} \pi(a' \mid s') Q^\pi(s',a'). && (5.9)
\end{aligned}
$$

Note that in the above discussion we overload the notation for density function p to also represent the probability measure $\mathbf{P}$ when talking about the transition model $p(s' \mid s,a)$. The intentions behind this is that we would like to use a unified representation for both discrete and continuous MDPs, and to emphasize the fact that the transition model p plays an important role in MDPs. One can distinguish whether p is a density function or a probability measure based on the context, especially according to how the probabilities are calculated (‘$\sum$’ or ‘$\int$’).

5.1.3 Optimal value functions and policies

Suppose π^* is a policy such that $V^{\pi^*} \geq V^\pi$ for all $s \in \mathcal{S}$ and all policy π, then it is an *optimal policy*. There can be multiple optimal policies for the same MDP, but by definition their value functions must be the same, and are denoted by V^* and Q^*, respectively. We call V^* the *optimal (state-)value function*, and Q^* the *optimal action-value function*.

Intuitively, the value of a state under an optimal policy must equal the expected return for the best action from that state:

$$
\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}} Q^*(s,a) && (5.10) \\
&= \max_{a \in \mathcal{A}} \mathbf{E}_{\pi^*}[G_0 \mid s_0 = s, a_0 = a] \\
&= \max_{a \in \mathcal{A}} \mathbf{E}_{\pi^*}[r_0 + \gamma G_1 \mid s_0 = s, a_0 = a] \\
&= \max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \mathbf{E}_{\pi^*}[G_1 \mid s_1 = s'] \right] \\
&= \max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^*(s') \right], && (5.11)
\end{aligned}
$$

for all $s \in \mathcal{S}$. The last equation formulate the *Bellman optimality equation* for value function V^*. This derivation procedure also informs us about the Bellman optimality equation for action-value function Q^*:

$$
Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \max_{a' \in \mathcal{A}} Q^*(s',a'), \qquad (5.12)
$$

for all $s \in \mathcal{S}, a \in \mathcal{A}$. Given a value function (V or Q), the discrepancy between the right- and left-hand sides of (5.11) and (5.12) are called *Bellman error* or *Bellman residual.* The backup diagrams in Figure 5.2 show graphically the spans of future states and actions considered in the Bellman optimality equations for V^* and Q^*. For finite MDPs, the Bellman optimality equations (5.11) and (5.12) has a unique solution π^*. The Bellman optimality equation is actually a system of equations, one for each state, so if there are n states, then there are n equations in n unknowns. If the dynamics p of the environment are known, then in principle one can solve this system of

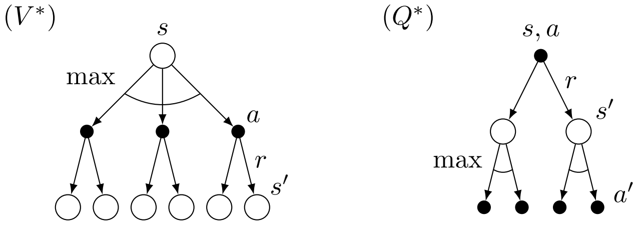


Figure 5.2 Backup diagrams for V^* and Q^*.

equations for V^* and Q^* using any one of a variety of methods for solving systems of nonlinear equations.

Once one has V^*, it is relatively easy to determine an optimal policy π^*. For each state s, there will be one or more actions at which the maximum is obtained in the Bellman optimality equation. Any policy that assigns nonzero probability only to these actions is an optimal policy. This can be considered as an one-step search. Given the optimal value function V^*, then the actions that appear best after a one-step search will be optimal actions, i.e.,

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^*(s') \right], \tag{5.13}$$

for all $s \in \mathcal{S}$. Another way of saying this is that any policy that is *greedy* with respect to the optimal evaluation function V^* is an optimal policy. The beauty of V^* is that if one uses it to evaluate the short-term consequences of actions — specifically, the one-step consequences — then a greedy policy is actually optimal in the long-term sense in which we are interested because V^* already takes into account the reward consequences of all possible future behavior.

Having Q^* makes choosing optimal actions even easier. With Q^*, the agent does not even have to do a one-step-ahead search: for any state s, it can simply find any action that maximizes $Q^*(s,a)$, i.e.,

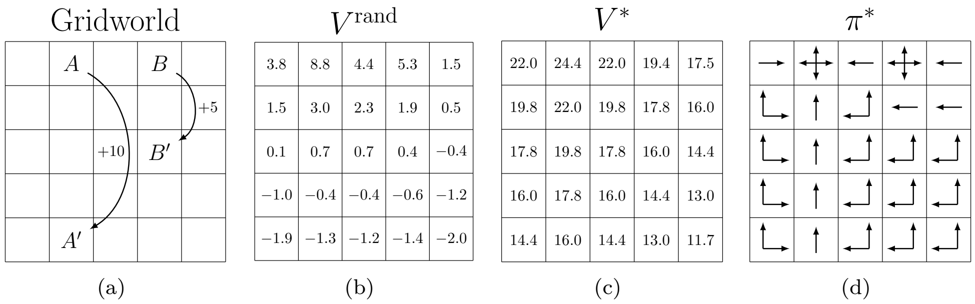
$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} Q^*(s,a). \tag{5.14}$$

The action-value function effectively caches the results of all one-step-ahead searches. It provides the optimal expected long-term return as a value that is locally and immediately available for each state-action pair. Hence, at the cost of representing a function of state-action pairs, instead of just of states, the optimal action-value function allows optimal actions to be selected without having to know anything about possible successor states and their values, that is, without having to know anything about the environment's dynamics.

5.1.4 Example

We now show a simple example, to make concepts like value functions more concrete. Figure 5.3(a) shows a rectangular *gridworld* representation of a simple finite MDP. The cells of the grid correspond to the states of the environment. At each cell, four actions are possible: *up*, *down*, *left*, and *right*, which deterministically cause the agent to move one cell in the respective direction on the grid. Actions that would take the agent off the grid leave its location unchanged, but also result in a reward of -1. Other actions result in a reward of 0, except those that move the agent out of the special states A and B. From state A, all four actions yield a reward of $+10$ and take the agent to A'. From state B, all actions yield a reward of $+5$ and take the agent to B'.

Suppose the agent selects all four actions with equal probability in all states. Figure 5.3(b) shows the value function V^π for this policy, for the discounted reward case with $\gamma = 0.9$. This value function was computed by solving the system of linear equations (5.8). Notice the negative values near the lower edge; these are the result of the high probability of hitting the edge of the grid there under the random policy. State A is the best state to be in under this policy. Note that A's expected return is less than its immediate reward of 10, because from A the agent is taken to state A' from which it is likely to run into the edge of the grid. State B, on the other hand, is valued more than its immediate reward of 5, because from B the agent is taken to B'



3.8	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	−0.4
−1.0	−0.4	−0.4	−0.6	−1.2
−1.9	−1.3	−1.2	−1.4	−2.0

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

Figure 5.3 Gridworld example: exceptional reward dynamics (a) and state-value function for the equiprobable random policy (b). (c) and (d) shows the optimal value function V^* and optimal policy π^* of this gridworld example.

which has a positive value. From B' the expected penalty (negative reward) for possibly running into an edge is more than compensated for by the expected gain for possibly stumbling onto A or B.

Suppose we now solve the Bellman optimality equation for V^* in this gridworld task. Figure 5.3(c) and 5.3(d) shows the optimal value function and the corresponding optimal policies, respectively. Where there are multiple arrows in a cell, all of the corresponding actions are optimal.

5.2 Dynamic programming

The term *dynamic programming* (DP) refers to a collection of iterative algorithms that can be used to compute optimal policies given a perfect model of the environment as an MDP. The key idea of DP is the use of value functions to organize and structure the search for good policies.

5.2.1 Policy iteration

Policy evaluation

First we consider how to compute the value function V^π for an arbitrary policy π. This called *policy evaluation* in the DP literature. We also refer to it as the *prediction problem*. According to (5.8), we have

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^\pi(s') \right],$$

for all $s \in \mathcal{S}$, where $\pi(a \mid s)$ is the probability of taking action a in state s under policy π. The existence and uniqueness of V^π is guaranteed as long as either $\gamma < 1$ or eventual termination is guaranteed for all states under the policy π.

If the environment's dynamics are completely known, then (5.8) is a system of $\mathbf{card}(\mathcal{S})$ simultaneous linear equations in $\mathbf{card}(\mathcal{S})$ unknowns ($V^\pi(s)$, for all $s \in \mathcal{S}$). In practice, we perfer to solve the system of linear equations with iterative methods. Consider a sequence of approximate value functions $V^{(0)}, V^{(1)}, V^{(2)}, \ldots$, with $V^{(i)} \colon \mathcal{S} \to \mathbf{R}$. The initial approximation $V^{(0)}$ is chosen arbitrarily (except that the terminal state, if any, must be given value 0), and each successive approximation is obtained by using the Bellman equation for value function (5.8) as an update rule:

$$V^{(i+1)}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^{(i)}(s') \right], \tag{5.15}$$

for all $s \in \mathcal{S}$. The sequence $V^{(0)}, V^{(1)}, \ldots, V^{(i)}, \ldots$ can be shown in general to converge to V^π as $i \to \infty$ under the same conditions that guarantee the existence of V^π. This algorithm is called *iterative policy evaluation*.

To write a sequential computer program to implement iterative policy evaluation as given by (5.15), we would have to use two arrays, one for the old values $V^{(i)}$, and one for the new values $V^{(i+1)}$. With two arrays, the new values can be computed one by one from the old values without the old values being changed. Alternatively, you could use one array and update the values 'in place', that is, with each new value immediately overwriting the old one. Then, depending on the order in which the states are updated, sometimes new values are used instead of old ones on the right-hand side of (5.15). This in-place algorithm also converges to V^π; in fact, it usually converges faster than the two-array version, as you might expect, because it uses new data as soon as they are available. We think of the updates as being done in a sweep through the state space. For the in-place algorithm, the order in which states have their values updated during the sweep has a significant influence on the rate of convergence. We usually have the in-place version in mind when we think of DP algorithms. A complete in-place version of iterative policy evaluation is shown in pseudocode in Algorithm 5.1.

Algorithm 5.1 *Iterative policy evaluation.*

```
given the policy $\pi$ to be evaluated.
initialize $V(s)$ for all $s \in \mathcal{S}$ arbitrarily, if $s$ is not terminal, otherwise 0.
repeat
    for $s \in \mathcal{S}$ do
        $V(s) := \sum_{a\in\mathcal{A}} \pi(a \mid s) \left[r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s' \mid s,a) V(s')\right]$.
    end for
until stop criterion reached.
```

Policy improvement

After obtaining the value function for some policy π from policy evaluation, we can find a new greedy policy π', according to

$$\pi'(s) = \operatorname*{argmax}_{a\in\mathcal{A}} Q^\pi(s,a) = \operatorname*{argmax}_{a\in\mathcal{A}} \left[r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s' \mid s,a) V^\pi(s') \right], \tag{5.16}$$

for all $s \in \mathcal{S}$. The new greedy policy takes the action that looks best in the short term — after one step of lookahead — according to V^π. The process of making a new policy that improves on an original policy, by making it greedy with respect to the value function of the original policy, is called *policy improvement*.

It can be easily shown that the new greedy policy π' is as good as, or better than the old policy π. Specifically, in the former case, the old policy π must be the optimal policy π^*. Suppose π' is as good as, but not better than π. Then $V^\pi = V^{\pi'}$, and from (5.16) it follows that for all $s \in \mathcal{S}$:

$$V^{\pi'}(s) = \max_{a\in\mathcal{A}} \left[r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s' \mid s,a) V^{\pi'}(s') \right],$$

which is exactly the Bellman optimality equation. Therefore, $V^{\pi'}$ must be V^*, and both π and π' must be optimal policies.

Policy iteration

Once a policy π, has been improved using V^π to yield a better policy π', we can then compute $V^{\pi'}$ and improve it again to yield an even better π''. We can thus obtain a sequence of monotonically improving policies and value functions:

$$\pi^{(0)} \xrightarrow{\text{E}} V^{\pi^{(0)}} \xrightarrow{\text{I}} \pi^{(1)} \xrightarrow{\text{E}} V^{\pi^{(1)}} \xrightarrow{\text{I}} \pi^{(2)} \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi^* \xrightarrow{\text{E}} V^*,$$

where $\xrightarrow{\text{E}}$ denotes a policy evaluation and $\xrightarrow{\text{I}}$ denotes a policy improvement. Each policy is guaranteed to be a strict improvement over the previous one (unless it is already optimal). Because a finite MDP has only a finite number of deterministic policies, this process must converge to an optimal policy and the optimal value function in a finite number of iterations.

This way of finding an optimal policy is called *policy iteration*. A complete algorithm is given in Algorithm 5.2. Note that each policy evaluation, itself an iterative computation, is started with the value function for the previous policy. This typically results in a great increase in the speed of convergence of policy evaluation (presumably because the value function changes little from one policy to the next).

Algorithm 5.2 *Policy iteration.*

```
1.  Initialization.
initialize $V(s) \in \mathbf{R}$ and $\pi(s) \in \mathcal{A}$ for all $s \in \mathcal{S}$.
repeat
    2.  Policy evaluation.
    repeat
        for $s \in \mathcal{S}$ do
            $V(s) \coloneqq \sum_{a \in \mathcal{A}} \pi(a \mid s) \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V(s')\right]$.
        end for
    until stop criterion reached.
    3.  Policy improvement.
    for $s \in \mathcal{S}$ do
        $\pi(s) \coloneqq \operatorname{argmax}_{a \in \mathcal{A}} \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V(s')\right]$.
    end for
until policy is stable.
```

5.2.2 Value iteration

One drawback to policy iteration is that each of its iterations involves policy evaluation, which may itself be a protracted iterative computation requiring multiple sweeps through the state set. In fact, the policy evaluation step of policy iteration can be truncated in several ways without losing the convergence guarantees of policy iteration. One important special case is when policy evaluation is stopped after just one sweep (one update of each state). This algorithm is called *value iteration* (VI). It can be written as a particularly simple update operation that combines the policy improvement and truncated policy evaluation steps:

$$V^{(i+1)}(s) = \max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^{(i)}(s') \right], \tag{5.17}$$

for all $s \in \mathcal{S}$. For arbitrary $V^{(0)}$, the sequence $V^{(0)}, V^{(1)}, \ldots, V^{(i)}, \ldots$ can be shown to converge to V^* under the same conditions that guarantee the existence of V^*.

Another way of understanding value iteration is by reference to the Bellman optimality equation (5.11). Note that value iteration is obtained simply by turning the Bellman optimality equation into an update rule. Also note how the value iteration update is identical to the policy evaluation update (5.15) except that it requires the maximum to be taken over all actions. A complete in-place version of value iteration is shown in pseudocode in Algorithm 5.3.

Algorithm 5.3 *Value iteration.*

```
initialize $V(s)$ for all $s \in \mathcal{S}$ arbitrarily, if $s$ is not terminal, otherwise 0.
repeat
    for $s \in \mathcal{S}$ do
        $V(s) \coloneqq \max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V(s')\right]$.
    end for
until stop criterion reached.
output a deterministic policy $\pi \coloneqq \operatorname{argmax}_{a \in \mathcal{A}} \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V(s')\right]$.
```

Bibliography

Markov decision processes were known at least as early as the 1950s [Bel57]. A core body of research on Markov decision processes resulted from Ronald Howard's book [How60].

For a detailed mathematical analysis about some properties Markov decision processes and dynamical programming, one can refer to the book [Put14].

MDPs are closely connected with *reinforcement learning* (RL), one can refer to [Mur23, §35] for an overview about RL. More details can be found in the famous textbook [SB18].

Exercises

5.1 *Dynamic programming.* An agent is moving on a 4×4 grid and the goal is to reach one of the terminal states T at the top left or the bottom right corner. A visualization of this environment is shown below.

T			
	A		
			T

The agent can go *up*, *down*, *left*, and *right*. Actions leading off the edge do not change the state. The agent receives a reward of -1 in each step until it reaches a terminal state. An implementation of this environment is given in `gridworld.py`.

(a) *Policy evaluation.* Implement the policy evaluation function,

```
policy_eval(policy, env, discount_factor=1.0, theta=0.00001),
```

in `policy_iteration.py`, where

- `policy` is a $[\mathbf{card}(\mathcal{S}), \mathbf{card}(\mathcal{A})]$ shaped matrix representing the policy,
- `env` is a discrete OpenAI-style environment and `env.P[s][a]` is a transition tuple `(transition_probability, next_state, reward, done)` for state s and action a, and
- `theta` is the stopping threshold. We stop the evaluation once our value-function change (difference between two iterations) is less than `theta` for all states.

It returns a vector of length $\mathbf{card}(\mathcal{S})$ representing the value-function.

(b) *Policy improvement.* Implement the policy improvement function,

```
policy_improvement(env, policy_eval_fn=policy_eval, discount_factor=1.0),
```

in `policy_iteration.py`. It returns a tuple `(policy, V)` where `policy` is the optimal policy — a matrix of shape $[\mathbf{card}(\mathcal{S}), \mathbf{card}(\mathcal{A})]$ where each state $s \in \mathcal{S}$ contains a valid probability distribution over actions, and `V` is the value-function for the optimal policy.

(c) *Value iteration.* Implement the value iteration function,

```
value_iteration(env, theta=0.0001, discount_factor=1.0),
```

in `value_iteration.py`. It again returns a tuple `(policy, V)` of the optimal policy and the optimal value-function.

You can find the tests for your implementation in `5-1.py`. Run them by

```
python 5-1.py -v,
```

or by

```
python -m unittest 5-1.py -v.
```

Chapter 6

Control as probabilistic inference

The framework of *reinforcement learning* or *optimal control* provides variants of methods for solving Markov decision problems. In this chapter, we present the basic graphical model that allows us to embed reinforcement learning problems into the framework of probabilistic graphical models.

In the following discussion, we use $s \in \mathcal{S}$ to denote states and $a \in \mathcal{A}$ to denote actions, which may each be discrete of continuous. States evolve according to the stochastic dynamics $p(s_{t+1} \mid s_t, a_t)$, which are in general unknown. We will follow a discrete-time finite-horizon derivation, which horizon T, and omit discount factor for now. A discount γ can be readily incorporated into this framework simply by modifying the transition dynamics, such that any action produces a transition into an absorbing state with probability $1 - \gamma$, and all standard transition probabilities are multiplied by γ. A task in this framework can be defined by a reward function $r(s_t, a_t)$. Solving a task typically involves recovering a policy

$$\pi_\theta(s_t \mid a_t) = p(a_t \mid s_t, \theta),$$

which specifies a distribution over actions conditioned on the state parameterized by some parameter vector θ. A standard reinforcement learning policy search problem is then given by the following optimization problem:

$$\text{maximize} \quad \textstyle\sum_{t=1}^{T} \mathbf{E}_{(s_t,a_t)\sim p(s_t,a_t|\theta)}[r(s_t, a_t)], \tag{6.1}$$

with θ being the optimization variable. This problem aims to find a vector of policy parameters θ that maximize the total expected reward $\sum_t r(s_t, a_t)$ of the policy, and the expectation is taken under the policy's trajectory distribution $p(\tau)$, given by

$$p(\tau) = p(s_1, a_1, \ldots, s_T, a_T \mid \theta) = p(s_1) \prod_{t=1}^{T} p(a_t \mid s_t, \theta) p(s_{t+1} \mid s_t, a_t). \tag{6.2}$$

6.1 Exact inference

6.1.1 The graphical model and policy search

To embed the control problem into a graphical model, we can begin simply by modeling the relationship between states, actions, and next states. This relationship is simple, and corresponds to a graphical model with factors of the form $p(s_{t+1} \mid s_t, a_t)$, as shown in Figure 6.1(a). However, this graphical model is insufficient for solving control problems, because it has no notion of rewards or costs. We therefore have to introduce an additional variable into this model, which we will denote $\mathcal{O}$. This additional variable is a binary random variable, where $o_t = 1$ denotes that step t is optimal, and $o_t = 0$ denotes that it is not optimal. We will choose the distribution over this variable to be given by the following equation:

$$p(o_t = 1 \mid s_t, a_t) = \exp(r(s_t, a_t)), \tag{6.3}$$

where $r(s_t, a_t)$ is the reward function and we have assumed without much lose of generality that $r(s_t, a_t) < 0$ for all $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$, so that (6.3) gives a valid probability. The graphical model with these additional variables is summarized in Figure 6.1(b).

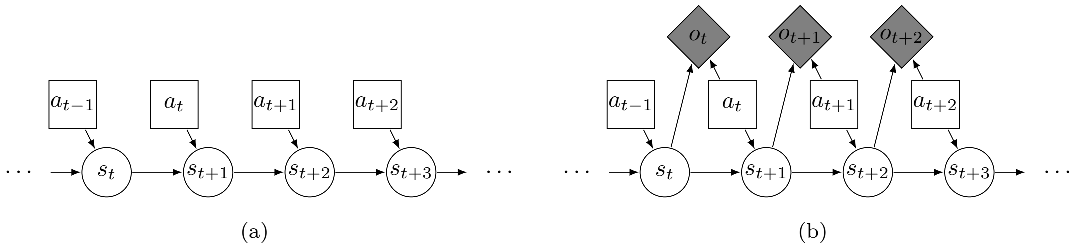


Figure 6.1 The graphical model for control problems with states and actions (a), and with optimality variables (b).

In this graphical model, the optimal policy can be written as $p(a_t \mid s_t, o_{1:T} = \mathbf{1})$. (We will write $o_{1:T} = \mathbf{1}$ as $o^*_{1:T}$ in the remainder of the derivation for conciseness.) This distribution is somewhat analogous to $p(a_t \mid s_t, \theta^*)$ with θ^* being the optimal value of θ for (6.1). We can then recover the optimal policy $p(a_t \mid s_t, o^*_{1:T})$ using a standard sum-product inference algorithm, analogously to inference in HMM-style dynamic Bayesian networks. First, note that $o^*_{1:(t-1)}$ is conditionally independent of a_t given s_t, which means that $p(a_t \mid s_t, o^*_{1:T}) = p(a_t \mid s_t, o^*_{t:T})$. Let

$$\beta(s_t, a_t) = p(o^*_{t:T} \mid s_t, a_t) \tag{6.4}$$

being the backward message of state-action pairs (s_t, a_t), which denotes the probability that a trajectory can be optimal for time steps from t to T if it begins in state s_t with the action a_t. Slightly overloading the notation, we will also introduce the message

$$\beta(s_t) = p(o^*_{t:T} \mid s_t), \tag{6.5}$$

which denotes the probability that the trajectory from t to T is optimal if it begins in state s_t. We can recover the state-only message from the state-action message by integrating out the action:

$$\beta(s_t) = p(o^*_{t:T} \mid s_t) = \int_{\mathcal{A}} p(o^*_{t:T} \mid s_t, a_t) p(a_t \mid s_t)\ da_t = \int_{\mathcal{A}} \beta(s_t, a_t) p(a_t \mid s_t)\ da_t, \tag{6.6}$$

where the factor $p(a_t \mid s_t)$ is the *action prior*. Note that it is not conditioned on $o^*_{1:T}$ in any way: it does not denote the probability of an optimal action, but simply the prior probability of actions. The graphical model in Figure 6.1(b) doesn't actually contain this factor, and we can assume that $p(a_t \mid s_t) = \frac{1}{\mathbf{card}(\mathcal{A})}$ without losing of generality, since any non-uniform priors can be incorporated instead into (6.3) via the reward function. The recursive message passing algorithm for computing $\beta(s_t, a_t)$ proceeds from the last time step $t = T$ backward through time to $t = 1$. In the base case, we note that $\beta(s_T, a_T)$ is simply proportional to $\exp(r(s_T, a_T))$ according to (6.3), since there is only one factor to consider. The recursive case is then given as following:

$$\beta(s_t, a_t) = p(o^*_{t:T} \mid s_t, a_t) = \int_{\mathcal{S}} \beta(s_{t+1}) p(s_{t+1} \mid s_t, a_t) p(o^*_t \mid s_t, a_t)\ ds_{t+1}. \tag{6.7}$$

From these backward messages, we can then derive the optimal policy $p(a_t \mid s_t, o^*_{t:T})$ as:

$$\begin{aligned} p(a_t \mid s_t, o^*_{t:T}) &= \frac{p(s_t, a_t \mid o^*_{t:T})}{p(s_t \mid o^*_{t:T})} = \frac{p(o^*_{t:T} \mid s_t, a_t) p(a_t \mid s_t) p(s_t)}{p(o^*_{t:T} \mid s_t) p(s_t)} \\ &\propto \frac{p(o^*_{t:T} \mid s_t, a_t)}{p(o^*_{t:T} \mid s_t)} = \frac{\beta(s_t, a_t)}{\beta(s_t)}, \end{aligned} \tag{6.8}$$

where the order of conditioning in the third step is flipped by using Bayes' rule, and cancelling the factor of $p(o^*_{t:T})$ that appears in both the numerator and denominator. The term $p(a_t \mid s_t)$ disappears, since we previously assumed it was a uniform distribution.

6.1.2 Connection to Bellman equations

The intuition about (6.8) can be recovered by considering what these equations are doing in log space. To that end, we will introduce the log-space messages as

$$Q(s_t, a_t) = \log \beta(s_t, a_t), \tag{6.9}$$

and

$$V(s_t) = \log \beta(s_t). \tag{6.10}$$

The use of Q and V here is not accidental: the log-space messages correspond to 'soft' variants of the state and state-action value functions. First, consider the marginalization over actions in log-space:

$$V(s_t) = \log \int_{\mathcal{A}} \exp(Q(s_t, a_t))\ da_t. \tag{6.11}$$

When the values of $Q(s_t, a_t)$ are large, the above equation resembles a hard maximum over a_t. That is, for large $Q(s_t, a_t)$, we have

$$V(s_t) = \log \int_{\mathcal{A}} \exp(Q(s_t, a_t))\ da_t \approx \max_{a_t} Q(s_t, a_t).$$

For smaller values of $Q(s_t, a_t)$, the maximum is soft. Hence, we can refer to V and Q as soft optimal value functions and Q-functions, respectively. We can also consider the backup in (6.7) in log-space. In the case of deterministic dynamics, this backup is given by

$$Q(s_t, a_t) = r(s_t, a_t) + V(s_{t+1}), \tag{6.12}$$

which exactly corresponds to the Bellman optimality equations (5.12). However, when the dynamics are stochastic, the backup is given by

$$\begin{aligned} Q(s_t, a_t) &= r(s_t, a_t) + \log \int_{\mathcal{S}} p(s_{t+1} \mid s_t, a_t) \exp(V(s_{t+1}))\ ds_{t+1} \\ &= r(s_t, a_t) + \log \mathbf{E}_{s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)}[\exp(V(s_{t+1}))]. \end{aligned} \tag{6.13}$$

The backup (6.13) is peculiar, since it does not consider the expected value at the next state, but a 'softmax' over the next expected value. Intuitively, this produces Q-functions that are optimistic: if among the possible outcomes for the next state there is one outcome with a very high value, it will dominate the backup, even when there are other possible states that might be likely and have extremely low value. This creates risk-seeking behavior: if an agent behaves according to this Q-function, it might take actions that have extremely high risk, so long as they have some non-zero probability of a high reward.

6.2 Approximate inference

6.2.1 Maximum entropy control

Given the graphical model in Figure 6.1(b), and recall that we consider the distribution of the optimality variable $\mathcal{O}$ to be given by

$$p(o_t^* \mid s_t, a_t) = \exp(r(s_t, a_t)).$$

We then obtain the posterior distribution over trajectories τ when we condition on $o_t = 1$ for all $t = 1, \dots, T$, i.e., all actions are optimal:

$$\begin{aligned} p(\tau \mid o_{1:T}^*) \propto p(\tau, o_{1:T}^*) &= p(s_1) \prod_{t=1}^{T} p(o_t^* \mid s_t, a_t) p(s_{t+1} \mid s_t, a_t) \\ &= p(s_1) \prod_{t=1}^{T} \exp(r(s_t, a_t)) p(s_{t+1} \mid s_t, a_t) \end{aligned}$$

$$= \left[p(s_1) \prod_{t=1}^{T} p(s_{t+1} \mid s_t, a_t) \right] \exp\left(\sum_{t=1}^{T} r(s_t, a_t) \right). \tag{6.14}$$

Suppose we are given some policy π_θ parameterized by θ, the distribution over trajectories τ can be written as

$$p_\theta(\tau) = p(s_1) \prod_{t=1}^{T} p(s_{t+1} \mid s_t, a_t) \pi_\theta(a_t \mid s_t). \tag{6.15}$$

Obviously, the optimal policy π^* has to result in a $p^*(\tau)$ according to (6.15) that match exactly to the optimal posterior trajectory distribution $p(\tau \mid o^*_{1:T})$ in (6.14). We can therefore view the inference process as minimizing the *KL-divergence* between $p_\theta(\tau)$ and $p(\tau \mid o^*_{1:T})$, which corresponds to the following optimization problem:

$$\text{minimize} \quad D_{\text{KL}}(p_\theta(\tau) \parallel p(\tau \mid o^*_{1:T})), \tag{6.16}$$

where θ is the optimization variable and the objective is given by:

$$D_{\text{KL}}(p_\theta(\tau) \parallel p(\tau \mid o^*_{1:T})) = -\mathbf{E}_{\tau \sim p_\theta(\tau)}[\log p(\tau \mid o^*_{1:T}) - \log p_\theta(\tau)]. \tag{6.17}$$

Negating both sides and substituting in the equations for $p_\theta(\tau)$ and $p(\tau \mid o^*_{1:T})$, we get

$$\begin{aligned} -D_{\text{KL}}(p_\theta(\tau) \parallel p(\tau \mid o^*_{1:T})) &= \mathbf{E}_{\tau \sim p_\theta(\tau)} \Bigg[\log p(s_1) + \sum_{t=1}^{T} (\log p(s_{t+1} \mid s_t, a_t) + r(s_t, a_t)) \\ &\qquad - \log p(s_1) - \sum_{t=1}^{T} (\log p(s_{t+1} \mid s_t, a_t) + \log \pi_\theta(a_t \mid s_t)) \Bigg] \\ &= \mathbf{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} r(s_t, a_t) - \log \pi_\theta(a_t \mid s_t) \right] \\ &= \sum_{t=1}^{T} \mathbf{E}_{(s_t, a_t) \sim p_\theta(s_t, a_t)}[r(s_t, a_t) - \log \pi_\theta(a_t \mid s_t)] \\ &= \sum_{t=1}^{T} \mathbf{E}_{(s_t, a_t) \sim p_\theta(s_t, a_t)}[r(s_t, a_t)] + \sum_{t=1}^{T} \mathbf{E}_{s_t \sim p_\theta(s_t)}[\mathcal{H}(\pi_\theta(s_t))], \end{aligned} \tag{6.18}$$

where $\mathcal{H}(\pi_\theta(s_t))$ denotes the entropy of policy π_θ at state s_t. Therefore, minimizing the KL-divergence corresponds to maximizing the expected reward and the expected policy entropy, in contrast to the standard control objective in (6.1), which only maximizes reward. This type of control objective is sometimes referred to as *maximum entropy reinforcement learning* or *maximum entropy control*.

6.2.2 Connection to variational inference

One way to interpret the objective function (6.18) is as a particular type of structured variational inference. In structured variational inference, our goal is to approximate some distribution $p(x)$ with another, potentially simpler distribution $q(x)$. Typically, $q(x)$ is taken to be some tractable factorized distribution, such as a product of conditional distributions connected in a chain or tree, which lends itself to tractable exact inference. In our case, we aim to approximate $p(\tau \mid o^*_{1:T})$, given by

$$p(\tau \mid o^*_{1:T}) = \left[p(s_1) \prod_{t=1}^{T} p(s_{t+1} \mid s_t, a_t) \right] \exp\left(\sum_{t=1}^{T} r(s_t, a_t) \right),$$

with the distribution

$$q(\tau) = q(s_1) \prod_{t=1}^{T} q(s_{t+1} \mid s_t, a_t) q(a_t \mid s_t). \tag{6.19}$$

Let $q(s_1) = p(s_1)$ and $q(s_{t+1} \mid s_t, a_t) = p(s_{t+1} \mid s_t, a_t)$, then $q(\tau)$ is exactly the distribution $p_\theta(\tau)$ from (6.15) with $q(a_t \mid s_t) = \pi_\theta(a_t \mid s_t)$. In structured variational inference, approximate inference is performed by optimizing the *variational lower bound* (also called the *evidence lower* *bound*). Recall that our evidence here is that $o_t = 1$ for all $t = 1, \dots, T$, thus the variational lower bound is given by

$$\begin{aligned}
\log p(o^*_{1:T}) &= \log \iint p(o^*_{1:T}, s_{1:T}, a_{1:T})\ ds_{1:T} da_{1:T} \\
&= \log \iint p(o^*_{1:T}, s_{1:T}, a_{1:T}) \frac{q(s_{1:T}, a_{1:T})}{q(s_{1:T}, a_{1:T})}\ ds_{1:T} da_{1:T} \\
&= \log \mathbf{E}_{(s_{1:T}, a_{1:T}) \sim q(s_{1:T}, a_{1:T})} \left[\frac{p(o^*_{1:T}, s_{1:T}, a_{1:T})}{q(s_{1:T}, a_{1:T})} \right] \\
&\geq \mathbf{E}_{(s_{1:T}, a_{1:T}) \sim q(s_{1:T}, a_{1:T})} [\log p(o^*_{1:T}, s_{1:T}, a_{1:T}) - \log q(s_{1:T}, a_{1:T})], \qquad (6.20)
\end{aligned}$$

where the last inequality holds because of Jensen's inequality. Substituting $p(o^*_{1:T}, s_{1:T}, a_{1:T})$ and $q(s_{1:T}, a_{1:T})$ according to (6.14) and (6.19), the bound reduces to

$$\log p(o^*_{1:T}) \geq \mathbf{E}_{(s_{1:T}, a_{1:T}) \sim q(s_{1:T}, a_{1:T})} \left[\sum_{t=1}^{T} r(s_t, a_t) - \log q(a_t \mid s_t) \right] \qquad (6.21)$$

up to an additive constant. Optimizing this objective with respect to the policy $q(a_t \mid s_t)$ corresponds exactly to the objective in (6.18).

6.2.3 Obtaining the optimal policy

To maximize the maximum entropy control objective

$$\begin{aligned}
-D_{\mathrm{KL}}(p_\theta(\tau) \parallel p(\tau \mid o^*_{1:T})) &= \sum_{t=1}^{T} \mathbf{E}_{(s_t, a_t) \sim p_\theta(s_t, a_t)} [r(s_t, a_t) - \log \pi_\theta(a_t \mid s_t)] \\
&= \sum_{t=1}^{T} \mathbf{E}_{(s_t, a_t) \sim p_\theta(s_t, a_t)} [r(s_t, a_t)] + \sum_{t=1}^{T} \mathbf{E}_{s_t \sim p_\theta(s_t)} [\mathcal{H}(\pi_\theta(s_t))],
\end{aligned}$$

we have to derive the backward messages from an optimization perspective as a dynamic programming algorithm. We will begin with the base case of optimizing $\pi(s_T \mid a_T)$, which consists in maximizing

$$\begin{aligned}
&\mathbf{E}_{(s_T, a_T) \sim p_\theta(s_T, a_T)} [r(s_T, a_T) - \log \pi_\theta(a_T \mid s_T)] \\
&\quad = \mathbf{E}_{(s_T, a_T) \sim p_\theta(s_T, a_T)} \left[\log \frac{\exp(r(s_T, a_T))}{\exp(V(s_T))} - \log \pi_\theta(a_T \mid s_T) + V(s_T) \right] \\
&\quad = \mathbf{E}_{s_T \sim p_\theta(s_T)} \left[-D_{\mathrm{KL}} \left(\pi_\theta(s_T) \middle\| \frac{1}{\exp(V(s_T))} \exp(r(s_T)) \right) + V(s_T) \right]. \qquad (6.22)
\end{aligned}$$

where the last equality holds from the definition of KL-divergence, and $\exp(V(s_T))$ is the normalizing constant for $\exp(r(s_T))$ w.r.t. a_T, i.e.,

$$V(s_T) = \log \int_{\mathcal{A}} \exp(r(s_T, a_T))\ da_T. \qquad (6.23)$$

Since we know that the KL-divergence is minimized when the two arguments represent the same distribution, the optimal policy is given by

$$\pi_\theta(a_T \mid s_T) = \exp(r(s_T, a_T) - V(s_T)). \qquad (6.24)$$

The recursive case can then computed as following: for a given time step t, $\pi_\theta(a_t \mid s_t)$ must maximize two terms:

$$\begin{aligned}
&\mathbf{E}_{(s_t, a_t) \sim p_\theta(s_t, a_t)} [r(s_t, a_t) - \log \pi_\theta(a_t \mid s_t)] + \mathbf{E}_{(s_t, a_t) \sim p_\theta(s_t, a_t)} [\mathbf{E}_{s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)} [V(s_{t+1})]] \\
&\quad = \mathbf{E}_{(s_t, a_t) \sim p_\theta(s_t, a_t)} [r(s_t, a_t) + \mathbf{E}_{s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)} [V(s_{t+1})] - \log \pi_\theta(a_t \mid s_t)] \\
&\quad = \mathbf{E}_{(s_t, a_t) \sim p_\theta(s_t, a_t)} \left[\log \frac{\exp(r(s_t, a_t) + \mathbf{E}_{s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)} [V(s_{t+1})])}{\exp(V(s_t))} - \log \pi_\theta(a_t \mid s_t) + V(s_t) \right]
\end{aligned}$$

$$= \mathbf{E}_{s_t \sim p_\theta(s_t)} \left[-D_{\mathrm{KL}} \left(\pi_\theta(s_t) \,\middle\|\, \frac{1}{\exp(V(s_t))} \exp(Q(s_t)) \right) + V(s_t) \right], \tag{6.25}$$

where we now define

$$Q(s_t, a_t) = r(s_t, a_t) + \mathbf{E}_{s_{t+1} \sim p(s_{t+1} | s_t, a_t)}[V(s_{t+1})], \tag{6.26}$$

and

$$V(s_t) = \log \int_{\mathcal{A}} \exp(Q(s_t, a_t)) \; da_t, \tag{6.27}$$

which corresponds to the standard Bellman optimality equations with a soft maximization for the value function. Choosing

$$\pi_\theta(a_t \mid s_t) = \exp(Q(s_t, a_t) - V(s_t)), \tag{6.28}$$

we again see that the KL-divergence evaluates to zero, where the objective function (6.25) is maximized. This means that we recover a Bellman backup operator that uses the expected value of the next state, rather than the optimistic estimate we saw in (6.13), which provides a solution to the practical problem of risk-seeking policies.

Bibliography

This chapter is mostly based on [Lev18]. Interested readers can also refer to [Att03, TS06, Tou09, BT12, KGO12, HR17] for more theoretical and empirical discussion about control as inference problems.

Appendices

Appendix A

Inference with Monte Carlo methods

A.1 Monte Carlo methods

A.1.1 Monte Carlo integration

We often want to compute the expected value of some function of a random variable, $\mathbf{E}[f(X)]$. This requires computing the following integral:

$$\mathbf{E}[f(X)] = \int f(x)p(x)\ dx, \tag{A.1}$$

where $x \in \mathbf{R}^n$, the function $f \colon \mathbf{R}^n \to \mathbf{R}^m$, and $p(x)$ is the target distribution of random variable X. Note that in many cases, the target distribution may be some posterior $p(x \mid y)$, which can be hard to compute. In such problems, instead, we often work with the unnormalized distribution, $\tilde{p}(x) = p(x, y)$, and then normalize the results using

$$Z = \int p(x, y)\ dx = p(y). \tag{A.2}$$

In low dimensions (up to, say, 3), we can compute the above integral efficiently using numerical integration, which (adaptively) computes a grid, and then evaluates the function at each point on the grid. But this does not scale to higher dimensions. An alternative approach is to draw multiple (say, n) random samples, $x \sim p(x)$, and then to compute

$$\mathbf{E}[f(X)] \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i). \tag{A.3}$$

This is called *Monte Carlo* (MC) *integration*. It has the advantage over numerical integration that the function is only evaluated in places where there is non-negligible probability, so it does not need to uniformly cover the entire space.

If we denote the exact mean by $\mu = \mathbf{E}[f(X)]$, and the MC approximation by $\hat{\mu}$, according to the central limit theorem, it can be shown that with independent samples,

$$(\hat{\mu} - \mu) \to \mathcal{N}\left(0, \frac{\hat{\sigma}^2}{n}\right),$$

where

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n} (f(x_i) - \hat{\mu})^2.$$

Thus for large enough n, we have

$$\mathbf{P}\left(\hat{\mu} - 1.96\sqrt{\frac{\hat{\sigma}^2}{n}} \le \mu \le \hat{\mu} + 1.96\sqrt{\frac{\hat{\sigma}^2}{n}}\right) \approx 0.95.$$

The term $\sqrt{\frac{\hat{\sigma}^2}{n}}$ is called the (numerical or empirical) *standard error*, and is an estimate of our uncertainty about our estimate of μ. The remarkable thing to note about the above results is that the standard error in the estimate, is theoretically independent of the dimensionality of the integral.

Example A.1 *Estimating π by Monte Carlo integration.* A *(Euclidean) ball* (or just ball) in $\mathbf{R}^n$ has the form

$$B(x_c, r) = \{x \mid \|x - x_c\|_2 \leq r\} = \{x \mid (x - x_c)^T(x - x_c) \leq r^2\},$$

where $r > 0$, and $\|\cdot\|_2$ denotes the Euclidean norm, i.e., $\|u\|_2 = (u^T u)^{1/2}$. The vector $x_c \in \mathbf{R}^n$ is the *center* of the ball and the scalar r is its *radius*; $B(x_c, r)$ consists of all points within a distance r of the center x_c.

Specifically, let $B(r) = \{x, y \mid x^2 + y^2 \leq r^2\}$ denotes a ball in $\mathbf{R}^2$ centered in the origin with radius r, we know that its area is πr^2, but it is also equal to the following definite integral

$$S = \int_{-r}^{r} \int_{-r}^{r} I_B(x, y)\ dxdy,$$

where $I_B(x, y)$ is an indicator function of set $B(r)$ which is 1 for points inside the ball, and 0 outside. Hence, the constant $\pi = S/r^2$. We can approximate this by Monte Carlo integration. Let $p(x)$ and $p(y)$ be uniform distribution on $[-r, r]$, so $p(x) = p(y) = 1/(2r)$ for all $x, y \in [-r, r]$, and 0 otherwise. Then

$$\begin{aligned}
\pi &= \frac{1}{r^2} S \\
&= \frac{1}{r^2}(2r)(2r) \iint I_B(x, y)p(x)p(y)\ dxdy \\
&= \frac{1}{r^2} 4r^2 \iint I_B(x, y)p(x)p(y)\ dxdy \\
&\approx 4 \times \frac{1}{n} \sum_{i=1}^{n} I_B(x_i, y_i).
\end{aligned}$$

A.1.2 Sampling from simple distributions

The main computational challenge of MC integration is to efficiently generate samples from the probability distribution $p(x)$. In this section, we discuss a sampling method that is suitable for parametric univariate distributions. These can be used as building blocks for sampling from more complex multivariate distributions.

The simplest method for sampling from a univariate distribution is based on the inverse probability transform. Let F be a *cumulative density function* (CDF) of some distribution we want to sample from, and let F^{-1} be its inverse. If $U \sim \mathcal{U}(0, 1)$ is a uniform random variable, then $F^{-1}(U) \sim F$. This can be easily shown as follows:

$$\begin{aligned}
\mathbf{P}\left(F^{-1}(U) \leq x\right) &= \mathbf{P}(U \leq F(x)) && \text{(applying } F \text{ to both sides)} \\
&= F(x), && \text{(because } \mathbf{P}(\mathcal{U} \leq y) = y\text{)}
\end{aligned}$$

where the first line follows since F is a monotonic function, and the second line follows since U is uniform on the unit interval.

Hence we can sample from any univariate distribution, for which we can evaluate its inverse CDF, as follows: generate a random number $u \sim \mathcal{U}(0, 1)$ using a *pseudorandom number generator* Let u represent the height up the y axis. Then 'slide along' the x axis until you intersect the F curve, and then 'drop down' and return the corresponding x value. This corresponds to computing $x = F^{-1}(u)$. This process is illustrated in Figure A.1.

Example A.2 *Sampling from an exponential distribution.* Consider the exponential distribution $\text{Exp}(\lambda)$ with density function

$$p_\lambda(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0. \end{cases}$$

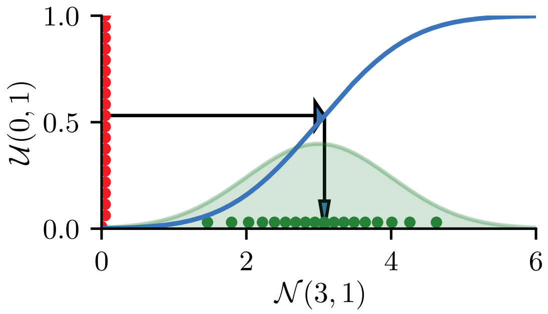


Figure A.1 Sampling from $\mathcal{N}(3,1)$ using an inverse CDF. The blue and green curves show the CDF and PDF of the target distribution, and the red and greed dots denote the samples from $\mathcal{U}(0,1)$ and $\mathcal{N}(3,1)$, respectively.

The CDF is

$$F_\lambda(x) = \begin{cases} 1 - e^{-\lambda x} & x \geq 0 \\ 0 & x < 0, \end{cases}$$

whose inverse is the quantile function

$$F_\lambda^{-1}(u) = -\frac{\log(1-u)}{\lambda}$$

with domain $\mathbf{dom}(F_\lambda^{-1}) = [0,1)$. If $U \sim \mathcal{U}(0,1)$, we know that $F_\lambda^{-1}(U) \sim \text{Exp}(\lambda)$. So we can sample from the exponential distribution by first sampling from the uniform and then transforming the results using $-\log(1-u)/\lambda$.

A.1.3 Rejection sampling

Suppose we want to sample from the target distribution

$$p(x) = \tilde{p}(x)/Z_p, \tag{A.4}$$

where $\tilde{p}(x)$ is the unnormalized version, and

$$Z_p = \int \tilde{p}(x)\ dx \tag{A.5}$$

is the (possibly unknown) normalization constant. One of the simplest approaches to this problem is *rejection sampling*.

In rejection sampling, we require access to a *proposal distribution* $q(x)$ which satisfies $Cq(x) \geq \tilde{p}(x)$, for some constant C. The function $Cq(x)$ provides an upper envelop for $\tilde{p}$. We can use the proposal distribution to generate samples from the target distribution as follows. For each sample, we first sample $x_i \sim q(x)$, which corresponds to picking a random x axis location, and then we sample $u_i \sim \mathcal{U}(0, Cq(x_i))$, which corresponds to picking a random height (y axis location) under the envelope. If $u_i > \tilde{p}(x_i)$, we reject the sample, otherwise we accept it. This process is illustrated in a 1-dimensional example in Figure A.2.

We now show this procedure is correct. First note that the probability of any given sample x_i being accepted equals the probability of a sample $u_i \sim \mathcal{U}(0, Cq(x_i))$ being less than or equal to $\tilde{p}(x_i)$, i.e.,

$$q(\text{accept} \mid x_i) = \int_0^{\tilde{p}(x_i)} \frac{1}{Cq(x_i)}\ du = \frac{\tilde{p}(x_i)}{Cq(x_i)}. \tag{A.6}$$

Therefore we have,

$$q(\text{propose and accept } x_i) = q(x_i)q(\text{accept} \mid x_i) = q(x_i)\frac{\tilde{p}(x_i)}{Cq(x_i)} = \frac{\tilde{p}(x_i)}{C}. \tag{A.7}$$

Integrating both sides give:

$$\int q(x_i)q(\text{accept} \mid x_i)\ dx_i = q(\text{accept}) = \frac{\int \tilde{p}(x_i)\ dx_i}{C} = \frac{Z_p}{C}. \tag{A.8}$$

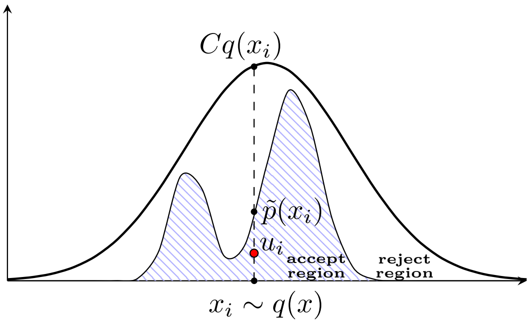


Figure A.2 Schematic illustration of rejection sampling. The acceptance region is shown shaded, and the rejection region is the white region between the shaded zone and the upper envelope.

Hence we see that the distribution of accepted points is given by the target distribution:

$$q(x_i \mid \text{accept}) = \frac{q(x_i, \text{accept})}{q(\text{accept})} = \frac{\tilde{p}(x_i)}{C} \frac{C}{Z_p} = \frac{\tilde{p}(x_i)}{Z_p} = p(x_i). \tag{A.9}$$

From (A.8) we also know that if $\tilde{p}$ is a normalized target distribution, the acceptance probability is $1/C$. Thus we might want to choose C as small as possible while still satisfying $Cq(x) \geq \tilde{p}(x)$. This means that we want to make our proposal $q(x)$ as close as possible to the target distribution $p(x)$, while still being an upper bound. But this is quite hard to achieve, especially in high dimensions. To see this, consider sampling from $p(x) = \mathcal{N}(0, \sigma_p^2 I)$ using the proposal $q(x) = \mathcal{N}(0, \sigma_q^2 I)$. Obviously we must have $\sigma_q^2 \geq \sigma_p^2$ in order to be an upper bound. In n dimensions, the optimum value is given by $C = (\sigma_q/\sigma_p)^n$. The acceptance rate is $1/C$ (since both p and q are normalized), which decreases exponentially fast with dimension. For example, if σ_q exceeds σ_p by just 1%, then in 1000 dimensions the acceptance ratio will be about 1/20000. This is a fundamental weakness of rejection sampling.

A.1.4 Importance sampling

We now introduce a Monte Carlo method known as *importance sampling* for approximating integrals of the form (A.1):

$$\mathbf{E}[f(X)] = \int f(x)p(x)\, dx,$$

where the function f is the target function, and $p(x)$ is the target distribution, which is often a conditional distribution of the form $p(x) = p(x \mid y)$. Since in general it is difficult to draw from the target distribution, we will instead draw from some proposal distribution $q(x)$ (which will usually depend on y). We then adjust for the inaccuracies of this by associating weights with each sample, so we end up with a weighted MC approximation:

$$\mathbf{E}[f(X)] \approx \sum_{i=1}^{n} W_i f(x_i). \tag{A.10}$$

We discuss two cases, first when the target is normalized, and then when it is unnormalized. This will affect the ways the weights are computed, as well as statistical properties of the estimator.

Direct importance sampling

In this section, we assume that we can evaluate the normalized target distribution $p(x)$, but we cannot sample from it. So instead we will sample from the proposal $q(x)$. We can then write

$$\int f(x)p(x)\, dx = \int f(x)\frac{p(x)}{q(x)}q(x)\, dx. \tag{A.11}$$

We require that the proposal be non-zero whenever the target is non-zero, i.e., the support of $q(x)$ needs to be greater than or equal to the support of $p(x)$. If we draw n samples $x \sim q(x)$, we can write

$$\mathbf{E}[f(X)] \approx \frac{1}{n}\sum_{i=1}^{n}\frac{p(x_i)}{q(x_i)}f(x_i) = \frac{1}{n}\sum_{i=1}^{n} w_i f(x_i), \tag{A.12}$$

where we define the *importance weight* w_i for each sample as follows:

$$w_i = \frac{p(x_i)}{q(x_i)}, \quad i = 1, \ldots, n. \tag{A.13}$$

The result is an unbiased estimate of the true mean $\mathbf{E}[f(X)]$.

Self-normalized importance sampling

The disadvantage of direct importance sampling is that we need a way to evaluate the normalized target distribution p in order to compute the weights. It is often much easier to evaluate the unnormalized target distribution $\tilde{p}(x) = Z_p p(x)$, where $Z_p = \int \tilde{p}(x)\ dx$ is the normalization constant. The key idea is to also approximate the normalization constant Z_p with importance sampling. This method is called *self-normalized importance sampling* (SNIS). The resulting estimate is a ratio of two estimates, and hence is biased. However as the number of samples $n \to \infty$, the bias goes to zero (under some weak assumptions, cf. the references listed in page 92).

In more detail, SNIS is based on this approximation:

$$\begin{aligned}\mathbf{E}[f(X)] = \int f(x)p(x)\ dx &= \frac{\int f(x)\tilde{p}(x)\ dx}{\int \tilde{p}(x)\ dx} = \frac{\int \left[\frac{\tilde{p}(x)}{q(x)}f(x)\right] q(x)\ dx}{\int \left[\frac{\tilde{p}(x)}{q(x)}\right] q(x)\ dx} \\ &\approx \frac{\frac{1}{n}\sum_{i=1}^{n}\tilde{w}_i f(x_i)}{\frac{1}{n}\sum_{i=1}^{n}\tilde{w}_i},\end{aligned} \tag{A.14}$$

where $x_i \sim q(x)$ for all $i = 1, \ldots, n$, and $\tilde{w}_i$ is the unnormalized weight for each sample, defined as

$$\tilde{w}_i = \frac{\tilde{p}(x_i)}{q(x_i)}, \quad i = 1, \ldots, n. \tag{A.15}$$

We can write (A.14) more compactly as

$$\mathbf{E}[f(X)] \approx \sum_{i=1}^{n} W_i f(x_i), \tag{A.16}$$

where W_i is the normalized weight for each sample, defined as

$$W_i = \frac{\tilde{w}_i}{\sum_{i'=1}^{n}\tilde{w}_{i'}}, \quad i = 1, \ldots, n. \tag{A.17}$$

This is equivalent to approximating the target distribution using a weighted sum of delta functions:

$$p(x) \approx \hat{p}(x) = \sum_{i=1}^{n} W_i \delta(x - x_i). \tag{A.18}$$

As a byproduct of this algorithm we get the following approximation to the normalization constant:

$$Z_p \approx \hat{Z}_p = \frac{1}{n}\sum_{i=1}^{n}\tilde{w}_i. \tag{A.19}$$

A.2 Markov chain Monte Carlo

In §A.1, we considered non-iterative Monte Carlo methods, including rejection sampling and importance sampling, which generate independent samples from some target distribution. The trouble with these methods is that they often do not work well in high dimensional spaces. In this section, we discuss a popular method for sampling from high-dimensional distributions known as *Markov chain Monte Carlo* (MCMC).

The basic idea behind MCMC is to construct a Markov chain (§3.1) on the state space X whose stationary distribution is the target density $p^*(x)$ of interest. (In a Bayesian context, this is usually a posterior, $p^*(x) \propto p(x \mid y)$, but MCMC can be applied to generate samples from any kind of distribution.) That is, we perform a random walk on the state space, in such a way that the fraction of time we spend in each state x is proportional to $p^*(x)$. By drawing (correlated) samples $x_0, x_1, \ldots$ from the chain, we can perform Monte Carlo integration w.r.t. p^*.

Note that the initial samples from the chain do not come from the stationary distribution, and should be discarded. The amount of time it takes to reach stationarity is called the *mixing time* or *burn-in time*. Reducing the burn-in time is one of the most important factors in making the algorithm fast.

A.2.1 Metropolis-Hastings algorithm

In this section, we describe the simplest kinds of MCMC algorithm known as the *Metropolis-Hastings* (MH) algorithm. The basic idea in MH is that at each step, we propose to move from the current state x to a new state x' with probability $q(x' \mid x)$, where q is called the proposal distribution (or *kernel*). The user is free to use any kind of proposal they want, subject to some conditions which we explain below. This makes MH quite a flexible method. Having proposed a move to x', we then decide whether to accept this proposal, or to reject it, according to some formula, which ensures that the long-term fraction of time spent in each state is proportional to $p^*(x)$. If the proposal is accepted, the new state is x', otherwise the new state is the same as the current state, x (i.e., we repeat the sample).

If the proposal is symmetric, so $q(x' \mid x) = q(x \mid x')$, the acceptance probability is given as follows:

$$A = \min\left\{1, \frac{p^*(x')}{p^*(x)}\right\}. \tag{A.20}$$

We see that if x' is more probable than x, we definitely move there (since $\frac{p^*(x')}{p^*(x)} > 1$), but if x' is less probable than x, we may still move there anyway, depending on the relative probabilities. So instead of greedily moving to only more probable states, we occasionally allow ‘downhill’ moves to less probable states.

If the proposal is asymmetric, so $q(x' \mid x) \neq q(x \mid x')$, we need the *Hastings correction*, given by the following:

$$A = \min\{1, \alpha\}, \tag{A.21}$$

where

$$\alpha = \frac{p^*(x')q(x \mid x')}{p^*(x)q(x' \mid x)} = \frac{p^*(x')/q(x' \mid x)}{p^*(x)/q(x \mid x')}. \tag{A.22}$$

This correction is needed to compensate for the fact that the proposal distribution itself (rather than just the target distribution) might favor certain states.

An important reason why MH is a useful algorithm is that, when evaluating α, we only need to know the target density up to a normalization constant. In particular, suppose $p^*(x) = \frac{1}{Z_p}\tilde{p}(x)$, where $\tilde{p}(x)$ is an unnormalized distribution and Z_p is the normalization constant. Then we have

$$\alpha = \frac{(\tilde{p}(x')/Z_p)q(x \mid x')}{(\tilde{p}(x)/Z_p)q(x' \mid x)},$$

where the Z_p’s cancel. Hence, we can sample from p^* even if Z_p is unknown.

A proposal distribution q is valid or admissible if it ‘covers’ the support of the target. Formally, we can write this as

$$\mathbf{supp}(p^*) \subseteq \cup_x \mathbf{supp}(q(\cdot \mid x)). \tag{A.23}$$

With this, we can state the overall algorithm as in Algorithm A.1.

Convergence analysis

To show that the MH procedure generates samples from p^*, we need a bit of Markov chain theory, as discussed in §3.1.2. The MH algorithm defines a Markov chain with the following transition matrix:

$$p(x' \mid x) = \begin{cases} q(x' \mid x)A(x' \mid x) & x' \neq x \\ q(x \mid x) + \sum_{x' \neq x} q(x' \mid x)(1 - A(x' \mid x)) & \text{otherwise.} \end{cases} \tag{A.24}$$

Algorithm A.1 *Metropolis-Hastings algorithm.*

```
given proposal distribution $q$.
initialize $x$.
repeat
    Sample $x' \sim q(x' \mid x)$.
    Compute $\alpha := \frac{p^*(x')q(x|x')}{p^*(x)q(x'|x)}$.
    Compute acceptance probability $A := \min\{1, \alpha\}$.
    Sample $u \sim \mathcal{U}(0,1)$.
    Set new sample to
        $x := \begin{cases} x' & u \le A \text{ (accept)} \\ x & u > A \text{ (reject).} \end{cases}$
until number of iterations reached.
```

This follows from a case analysis: if you move to x' from x, you must have proposed it (with probability $q(x' \mid x)$) and it must have been accepted (with probability $A(x' \mid x)$); otherwise you stay in state x, either because that is what you proposed (with probability $q(x \mid x)$), or because you proposed something else (with probability $q(x' \mid x)$) but it was rejected (with probability $1 - A(x' \mid x)$).

Let us analyze this Markov chain. Recall that a chain satisfies *detailed balance* if

$$p(x' \mid x)p^*(x) = p(x \mid x')p^*(x'). \tag{A.25}$$

This means in the in-flow to state x' from x is equal to the out-flow from state x' back to x, and vice versa. If a chain satisfies detailed balance, then p^* is its stationary distribution. Our goal is to prove that the MH algorithm defines a transition function p that satisfies detailed balance and hence that p^* is its stationary distribution. (If (A.25) holds, we say that p^* is an invariant distribution w.r.t. the Markov transition kernel p.) To show this, we assume that the Markov chain with transition matrix given by (A.24) is ergodic and irreducible. Consider two states x and x'. Either

$$p^*(x)q(x' \mid x) < p^*(x')q(x \mid x') \tag{A.26}$$

or

$$p^*(x)q(x' \mid x) \ge p^*(x')q(x \mid x'). \tag{A.27}$$

Without loss of generality, assume that $p^*(x)q(x' \mid x) > p^*(x')q(x \mid x')$. Then,

$$\alpha(x' \mid x) = \frac{p^*(x')q(x \mid x')}{p^*(x)q(x' \mid x)} < 1.$$

Hence, we have $A(x' \mid x) = \alpha(x' \mid x)$ and $A(x \mid x') = 1$. Now to move from x to x' we must first propose x' and then accept it, i.e.,

$$p(x' \mid x) = q(x' \mid x)A(x' \mid x) = q(x' \mid x)\frac{p^*(x')q(x \mid x')}{p^*(x)q(x' \mid x)} = \frac{p^*(x')}{p^*(x)}q(x \mid x'), \tag{A.28}$$

which indicates that

$$p^*(x)p(x' \mid x) = p^*(x')q(x \mid x'). \tag{A.29}$$

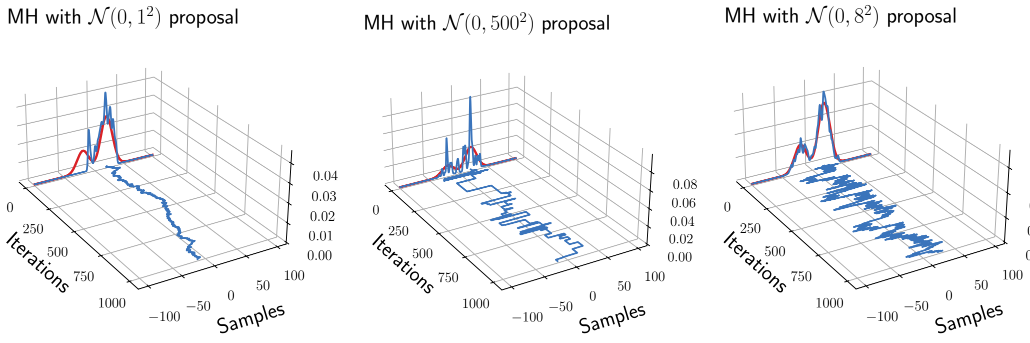
Since $A(x \mid x') = 1$, the backward probability can be written as

$$p(x \mid x') = q(x \mid x')A(x \mid x') = q(x \mid x'). \tag{A.30}$$

Inserting this into (A.29), we get

$$p^*(x)p(x' \mid x) = p^*(x')p(x \mid x'), \tag{A.25}$$

so detailed balance holds w.r.t. p^*. This shows that given the MH transition kernel p, the target distribution p^* is the unique stationary distribution of the Markov chain, since we have assumed that the chain is ergodic and irreducible.



(a) (b) (c)

Figure A.3 An example of the random walk Metropolis algorithm for sampling from a mixture of two 1-dimensional Gaussians, $\mathcal{N}(-20, 100)$ and $\mathcal{N}(20, 100)$, with mixing weights 0.3 and 0.7. The variance σ^2 of the Gaussian random walk proposal are 1^2 (a), 500^2 (b), and 8^2 (c).

Proposal distributions

We now discuss some common proposal distributions q. Note, however, that good proposal design is often intimately dependent on the form of the target distribution (most often the posterior).

Independence sampler. If we use a proposal of the form $q(x' \mid x) = q(x')$, where the new state is independent of the old state, we get a method known as the *independence sampler*, which is similar to importance sampling (§A.1.4). The function $q(x')$ can be any suitable distribution, such as a Gaussian. Since Gaussian distribution has non-zero probability density on the entire state space, so it is a valid proposal for any unconstrained continuous state space.

Random walk Metropolis. The *random walk Metropolis* (RWM) algorithm corresponds to MH with the following proposal

$$x' \sim \mathcal{N}(x, \sigma^2 I), \tag{A.31}$$

where the mean of this Gaussian distribution is the previous sample x, and the variance σ^2 is a scale factor chose to facilitate rapid mixing. This is equivalent to saying that the random vector $x' - x$ is Gaussian with mean 0 and variance $\sigma^2 I$, i.e., $(x' - x) \sim \mathcal{N}(0, \sigma^2 I)$.

Example A.3 *Sampling from a mixture of Gaussians with random walk Metropolis.* Figure A.3 shows an example where we use RWM to sample from a mixture of two 1-dimensional Gaussians. This is a somewhat tricky target distribution, since it consists of two somewhat separated modes. It is very important to set the variance of the proposal σ^2 correctly: if the variance is too low, the chain will only explore one of the modes, as shown in Figure A.3(a), but if the variance is too large, most of the moves will be rejected, and the chain will be very sticky, i.e., it will stay in the same state for a long time. This is evident from the long stretches of repeated values in Figure A.3(b). If we set the proposal's variance just right, we get the trace in Figure A.3(c), where the samples clearly explore the support of the target distribution.

Composing proposals. If there are several proposals that might be useful, one can combine them using a mixture proposal, which is a convex combination of some base proposals:

$$q(x' \mid x) = \sum_{i=1}^{m} w_i q_i(x' \mid x), \tag{A.32}$$

where w_i are the mixing weights that sum to one. As long as each q_i is an individually valid proposal, and each $w_i > 0$, then the overall mixture proposal will also be valid. In particular, if each proposal is reversible, so it satisfies detailed balance, then so does the mixture.

A.2.2 Gibbs sampling

The major problems with MH are the need to choose the proposal distribution, and the fact that the acceptance rate may be low. In this section, we describe an MH method that exploits conditional independence properties of a graphical model to automatically create a good proposal, with acceptance probability 1. This method is known as *Gibbs sampling.* In physics, this method is also known as *Glauber dynamics* or the *heat bath* method. This is the MCMC analog of coordinate descent.

The idea behind Gibbs sampling is to sample each variable in turn, conditioned on the values of all the other variables in the distribution. For example, if we have variable $X \in \mathbf{R}^3$, we use

$$\begin{aligned} x_1' &\sim p(x_1' \mid x_2, x_3) \\ x_2' &\sim p(x_2' \mid x_1', x_3) \\ x_3' &\sim p(x_3' \mid x_1', x_2'). \end{aligned}$$

This readily generalizes to n-dimensional variables. (Note that if X_i is a known variable, we do not sample it, but it may be used as input to the another conditional distributions.) The expression $p(x_i' \mid x_{-i})$ is called the *full conditional* for variable X_i. In general, X_i may only depend on some of the other variables. If we represent $p(x)$ as a graphical model, we can infer the dependencies by looking at the Markov blanket of X_i, which are its neighbors in the graph (see §4.1.1), so we can write

$$x_i' \sim p(x_i' \mid x_{-i}) = p(x_i' \mid \mathbf{mb}(x_i)). \tag{A.33}$$

Connections to MH

It turns out that Gibbs sampling is a special case of MH where we use a sequence of proposals of the form

$$q_i(x' \mid x) = p(x' \mid x_{-i}) I_{x_{-i}}(x'_{-i}), \quad i = 1, \ldots, n, \tag{A.34}$$

for some variable $X \in \mathbf{R}^n$, where $I_{x_{-i}}$ is the indicator function. That is, we move to a new state where x_i is sampled from its full conditional, but x_{-i} is left unchanged. We now show that the acceptance rate of each such proposal is 100%, so the overall algorithm also has an acceptance rate of 100%. For each proposal q_i, we have

$$\begin{aligned} \alpha = \frac{p(x')q_i(x \mid x')}{p(x)q_i(x' \mid x)} &= \frac{p(x_i' \mid x'_{-i})p(x'_{-i})p(x_i \mid x'_{-i})}{p(x_i \mid x_{-i})p(x_{-i})p(x_i' \mid x_{-i})} \\ &= \frac{p(x_i' \mid x_{-i})p(x_{-i})p(x_i \mid x_{-i})}{p(x_i \mid x_{-i})p(x_{-i})p(x_i' \mid x_{-i})} = 1, \end{aligned} \tag{A.35}$$

where we exploited the fact that $x'_{-i} = x_{-i}$.

Example A.4 *Gibbs sampling for Ising models.* Consider the 2-dimensional lattice $G = (X, E)$ in Figure A.4. We can represent the joint distribution of X as follows:

$$p(x) = \frac{1}{Z_p} \prod_{(X_i, X_j) \in E} \psi_{ij}(x_i, x_j), \tag{A.36}$$

where $\psi_{ij}(x_i, x_j)$ is named as the *potential function* of clique $C = \{X_i, X_j\}$. This is called a *lattice model.* An *Ising model* is a special case of lattice models, where the variables X_i are binary for all $i = 1, \ldots, n$. Such models are often used to represent magnetic materials. In particular, each node represents an atom, which can have a magnetic dipole, or *spin*, which is in one of two states, $+1$ and -1. In some magnetic systems, neighboring spins like to be similar; in other systems, they like to be dissimilar. We can capture this interaction by defining the clique potentials as follows:

$$\psi_{ij}(x_i, x_j) = \begin{cases} e^{J_{ij}} & x_i = x_j \\ e^{-J_{ij}} & x_i \neq x_j, \end{cases} \tag{A.37}$$

where J_{ij} is the coupling strength between nodes X_i and X_j. If two nodes are not connected in the graph, we set $J_{ij} = 0$. We assume that the weight matrix is symmetric, so $J_{ij} = J_{ji}$.

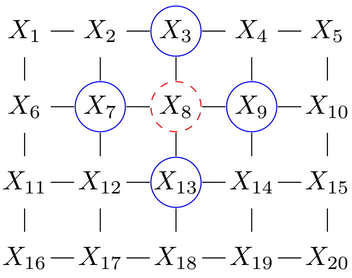


Figure A.4 A 2-dimensional lattice for random vector $X \in \mathbf{R}^{20}$ represented as a undirected graph. The red node X_8 is independent of the other nodes given its neighbors (blue nodes).

Often we also assume all edges have the same strength, so $J_{ij} = J$ for all edges. Thus we have

$$\psi_{ij}(x_i, x_j) = \begin{cases} e^J & x_i = x_j \\ e^{-J} & x_i \neq x_j. \end{cases} \tag{A.38}$$

To perform the sampling for an n-dimensional random vector X following an Ising model, we need to compute the full conditional:

$$p(x_i \mid x_{-i}) \propto \prod_{X_j \in \mathbf{adj}(X_i)} \psi_{ij}(x_i, x_j), \tag{A.39}$$

for all $i = 1, \ldots, n$. In the case of an Ising model with edge potentials $\psi_{ij}(x_i, x_j) = \exp(Jx_ix_j)$, where $x_i, x_j \in \{-1, +1\}$, the full conditional becomes

$$\begin{aligned} p(x_i = +1 \mid x_{-i}) &= \frac{\prod_{X_j \in \mathbf{adj}(X_i)} \psi_{ij}(x_i = +1, x_j)}{\prod_{X_j \in \mathbf{adj}(X_i)} \psi_{ij}(x_i = +1, x_j) + \prod_{X_j \in \mathbf{adj}(X_i)} \psi_{ij}(x_i = -1, x_j)} \\ &= \frac{\exp(J \sum_{X_j \in \mathbf{adj}(X_i)} x_j)}{\exp(J \sum_{X_j \in \mathbf{adj}(X_i)} x_j) + \exp(-J \sum_{X_j \in \mathbf{adj}(X_i)} x_j)} \\ &= \frac{\exp(J\eta_i)}{\exp(J\eta_i) + \exp(-J\eta_i)}, \end{aligned} \tag{A.40}$$

where J is the coupling strength, and $\eta_i = \sum_{X_j \in \mathbf{adj}(X_i)} x_j$. It is easy to see that $\eta_i = x_i(a_i - d_i)$, where a_i is the number of neighbors that agree with (have the same sign as) node X_i, and d_i is the number of neighbors who disagree. If this number is equal, the 'forces' on xi cancel out, so the full conditional is uniform.

Metropolis within Gibbs

When implementing Gibbs sampling, we have to sample from the full conditionals. If the distributions are conjugate, we can compute the full conditional in closed form, but in the general case, we will need to devise special algorithms to sample from the full conditionals. One approach is to use the MH algorithm, which is called *Metropolis within Gibbs*. In particular, to sample $x'_i \sim p(x'_i \mid x'_{1:i-1}, x_{i+1:n})$, we proceed in 4 steps:

1. Propose $x''_i \sim q(x''_i \mid x_i)$.
2. Compute the acceptance probability $A_i = \min\{1, \alpha_i\}$, where

$$\alpha_i = \frac{p(x''_i \mid x'_{1:i-1}, x_{i+1:n})q(x_i \mid x''_i)}{p(x_i \mid x'_{1:i-1}, x_{i+1:n})q(x''_i \mid x_i)}. \tag{A.41}$$

3. Sample $u \sim \mathcal{U}(0, 1)$.
4. Set $x'_i = x''_i$ if $u < A_i$, and $x'_i = x_i$ otherwise.

A.2.3 Hamiltonian Monte Carlo

Many MCMC algorithms perform poorly in high dimensional spaces, because they rely on a form of random search based on local perturbations. In this section, we discuss a method known as *Hamiltonian Monte Carlo* (HMC), that leverages gradient information to guide the local moves.

Hamiltonian mechanics

Consider a particle rolling around an energy landscape. We can characterize the motion of the particle in terms of its position $\theta \in \mathbf{R}^n$, and its momentum $v \in \mathbf{R}^n$. The set of possible values for (θ, v) is called the *phase space*. We define the Hamiltonian function for each point in phase space as follows:

$$\mathcal{H}(\theta, v) = \mathcal{E}(\theta) + \mathcal{K}(v), \tag{A.42}$$

where $\mathcal{E}(\theta)$ is the *potential energy*, $\mathcal{K}(v)$ is the *kinetic energy*, and the Hamiltonian is the total energy. In a physical setting, the potential energy is due to the pull of gravity, and the momentum is due to the motion of the particle. In a statistical setting, we often take the potential energy to be

$$\mathcal{E}(\theta) = -\log \tilde{p}(\theta), \tag{A.43}$$

where $\tilde{p}(\theta)$ is a possibly unnormalized distribution, such as $p(\theta \mid \mathcal{D})$, and the kinetic energy to be

$$\mathcal{K}(v) = \frac{1}{2} v^T \Sigma^{-1} v, \tag{A.44}$$

where $\Sigma \in \mathbf{S}^n_{++}$ is a symmetric, positive definite matrix, known as the *mass matrix*.

Stable orbits are defined by trajectories in phase space that have a constant energy. The trajectory of a particle within an energy level set can be obtained by solving the following continuous time differential equations, known as *Hamilton's equations*:

$$\begin{cases} \dfrac{d\theta}{dt} = \dfrac{\partial \mathcal{H}}{\partial v} = \dfrac{\partial \mathcal{K}}{\partial v} \\ \dfrac{dv}{dt} = -\dfrac{\partial \mathcal{H}}{\partial \theta} = -\dfrac{\partial \mathcal{E}}{\partial \theta}. \end{cases} \tag{A.45}$$

To see why energy is conserved, note that

$$\frac{d\mathcal{H}}{dt} = \sum_{i=1}^{n} \left(\frac{\partial \mathcal{H}}{\partial \theta_i} \frac{d\theta_i}{dt} + \frac{\partial \mathcal{H}}{\partial v_i} \frac{dv_i}{dt} \right) = \sum_{i=1}^{n} \left(\frac{\partial \mathcal{H}}{\partial \theta_i} \frac{\partial \mathcal{H}}{\partial v_i} - \frac{\partial \mathcal{H}}{\partial \theta_i} \frac{\partial \mathcal{H}}{d v_i} \right) = 0. \tag{A.46}$$

Intuitively, we can understand this result as follows: a satellite in orbit around a planet will 'want' to continue in a straight line due to its momentum, but will get pulled in towards the planet due to gravity, and if these forces cancel, the orbit is stable. If the satellite starts spiraling towards the planet, its kinetic energy will increase but its potential energy will decrease. Note that the mapping from (θ_t, v_t) to $(\theta_{t+\Delta t}, v_{t+\Delta t})$ for some time increment Δt is invertible for small enough time steps. Furthermore, this mapping is volume preserving, so has a Jacobian determinant of 1. These facts will be important later when we turn this system into an MCMC algorithm.

Integrating Hamilton's equations

In this section, we discuss how to simulate Hamilton's equations in discrete time.

Euler's method. The simplest way to model the time evolution is to update the position and momentum simultaneously by a small amount, known as the step size η:

$$\begin{cases} v_{t+1} = v_t + \eta \left.\dfrac{dv}{dt}\right|_{\theta=\theta_t, v=v_t} = v_t - \eta \left.\dfrac{\partial \mathcal{E}}{\partial \theta}\right|_{\theta=\theta_t} \\ \theta_{t+1} = \theta_t + \eta \left.\dfrac{d\theta}{dt}\right|_{\theta=\theta_t, v=v_t} = \theta_t + \eta \left.\dfrac{\partial \mathcal{K}}{\partial v}\right|_{v=v_t}. \end{cases} \tag{A.47}$$

If the kinetic energy has the form in (A.44), then the second expression simplifies to

$$\theta_{t+1} = \theta_t + \eta \Sigma^{-1} v_t. \tag{A.48}$$

This is known as the *Euler's method*.

Modified Euler's method. The *modified Euler's method* is slightly more accurate, and works as follows. It first update the momentum, and then update the position using the new momentum:

$$\begin{cases} v_{t+1} = v_t + \eta \frac{dv}{dt}\Big|_{\theta=\theta_t, v=v_t} = v_t - \eta \frac{\partial \mathcal{E}}{\partial \theta}\Big|_{\theta=\theta_t} \\ \theta_{t+1} = \theta_t + \eta \frac{d\theta}{dt}\Big|_{\theta=\theta_t, v=v_{t+1}} = \theta_t + \eta \frac{\partial \mathcal{K}}{\partial v}\Big|_{v=v_{t+1}} . \end{cases} \tag{A.49}$$

Unfortunately, the asymmetry of this method can cause some theoretical problems.

Leapfrog integrator. The *leapfrog integrator* is a symmetrized version of the modified Euler's method. We first perform a 'half' update of the momentum, then a full update of the position, and then finally another 'half' update of the momentum:

$$\begin{cases} v_{t+1/2} = v_t - \frac{\eta}{2} \frac{\partial \mathcal{E}}{\partial \theta}\Big|_{\theta=\theta_t} \\ \theta_{t+1} = \theta_t + \eta \frac{\partial \mathcal{K}}{\partial v}\Big|_{v=v_{t+1/2}} \\ v_{t+1} = v_{t+1/2} - \frac{\eta}{2} \frac{\partial \mathcal{E}}{\partial \theta}\Big|_{\theta=\theta_{t+1}} . \end{cases} \tag{A.50}$$

If we perform multiple leapfrog steps, it is equivalent to performing a half step update of v at the beginning and end of the trajectory, and alternating between full step updates of θ and v in between.

The HMC algorithm

We now describe how to use Hamiltonian dynamics to define an MCMC sampler in the expanded state space (θ, v). The target distribution has the form

$$p(\theta, v) = \frac{1}{Z} \exp(-\mathcal{H}(\theta, v)) = \frac{1}{Z} \exp\left(-\mathcal{E}(\theta) - \frac{1}{2} v^T \Sigma^{-1} v\right). \tag{A.51}$$

Then we can just 'throw away' the v's so that the result will be the samples θ from the desired marginal:

$$p(\theta) = \int p(\theta, v)\, dv = \frac{1}{Z_\theta} e^{-\mathcal{E}(\theta)} \int \frac{1}{Z_v} e^{-\frac{1}{2} v^T \Sigma^{-1} v}\, dv = \frac{1}{Z_\theta} e^{-\mathcal{E}(\theta)}. \tag{A.52}$$

Suppose the previous state of the Markov chain is (θ_{t-1}, v_{t-1}), to sample the next state, we proceed as follows. We set the initial position to $\theta'_0 = \theta_{t-1}$, and sample a new random momentum, $v'_0 \sim \mathcal{N}(0, \Sigma)$[1]. We then initialize a random trajectory in the phase space, starting at (θ'_0, v'_0), and followed for L leapfrog steps, until we get to the final proposed state $(\theta^*, v^*) = (\theta'_L, v'_L)$. If we have simulated Hamiltonian mechanics correctly, the energy should be the same at the start and the end of this process. If not, we say the HMC has *diverged*, and we reject the sample. If the energy is constant, we compute the MH acceptance probability as

$$A = \min\left\{1, \frac{p(\theta^*, v^*)}{p(\theta_{t-1}, v_{t-1})}\right\} = \min\left\{1, \exp\left(-\mathcal{H}(\theta^*, v^*) + \mathcal{H}(\theta_{t-1}, v_{t-1})\right)\right\}, \tag{A.53}$$

where the transition probabilities cancel since the proposal is reversible. Finally, we accept the proposal by setting $(\theta_t, v_t) = (\theta^*, v^*)$ with probability A, otherwise we set $(\theta_t, v_t) = (\theta_{t-1}, v_{t-1})$. (In practice we don't need to keep the momentum term v, it is only used inside of the leapfrog algorithm.) The pseudocode for this procedure is shown in Algorithm A.2.

Note that we need to sample a new momentum at each iteration to satisfy ergodicity. To see why, recall that $\mathcal{H}(\theta, v)$ stays approximately constant as we move through phase space. If $\mathcal{H}(\theta, v) = \mathcal{E}(\theta) + \frac{1}{2} v^T \Sigma^{-1} v$, then clearly $\mathcal{E}(\theta) \leq \mathcal{H}(\theta, v) = h$ for all locations θ along the trajectory. Thus the sampler cannot reach states where $\mathcal{E}(\theta) > h$. To ensure the sampler explores the full space, we must pick a random momentum at the start of each iteration.

[1] Note that the Σ here denotes the covariance matrix of some Gaussian distribution, instead of the mass matrix in Hamiltonian mechanics.

Algorithm A.2 *Hamiltonian Monte Carlo.*

```
given the number of leapfrog steps $L$, the step size $\eta$, and the covariance matrix $\Sigma$.
repeat
    Generate random momentum $v_{t-1} \sim \mathcal{N}(0, \Sigma)$.
    Set $(\theta'_0, v'_0) \coloneqq (\theta_{t-1}, v_{t-1})$.
    Half step for momentum: $v'_{1/2} \coloneqq v'_0 - \frac{\eta}{2} \nabla \mathcal{E}(\theta'_0)$.
    for $l = 1, \ldots, L-1$ do
        $\theta'_l \coloneqq \theta'_{l-1} + \eta \Sigma^{-1} v'_{l-1/2}$.
        $v'_{l+1/2} \coloneqq v'_{l-1/2} - \eta \nabla \mathcal{E}(\theta'_l)$.
    end for
    Full step for location: $\theta'_L \coloneqq \theta'_{L-1} + \eta \Sigma^{-1} v'_{L-1/2}$.
    Half step for momentum: $v'_L \coloneqq v'_{L-1/2} - \frac{\eta}{2} \nabla \mathcal{E}(\theta'_L)$.
    Obtain proposal $(\theta^*, v^*) \coloneqq (\theta'_L, v'_L)$.
    Compute acceptance probability $A \coloneqq \min\{1, \exp(-\mathcal{H}(\theta^*, v^*) + \mathcal{H}(\theta_{t-1}, v_{t-1}))\}$.
    Set $\theta_t \coloneqq \theta^*$ with probability $A$, other wise $\theta_t \coloneqq \theta_{t-1}$.
until number of iterations reached.
```

Bibliography

This chapter is mostly adapted from [Mur23, §11 and §12]. For more details on Monte Carlo methods see also [LL01, RCC99, KTB13, BZ+20].

Except for using inverse probability transform introduced in this chapter, the *Box-Muller method* provides another approach to sample from a Gaussian distribution, which was originally developed by Box and Muller [BM58].

The statistical properties of the SNIS estimator as we mentioned briefly in §A.1.4 are discussed in detail in [RCC99].

The MCMC algorithm has an interesting history. It was discovered by physicists working on the atomic bomb at Los Alamos during World War II, and was first published in the open literature [MRR+53] in a chemistry journal. An extension was published in the statistics literature [Has70], but was largely unnoticed. A special case (Gibbs sampling, §A.2.2) was independently invented by Geman and Geman [GG84] in the context of Ising models. But it was not until [GS90] that the algorithm became well-known to the wider statistical community. Since then it has become wildly popular in Bayesian statistics, and is becoming increasingly popular in machine learning. For more details on the MCMC theory, see e.g., [GRS95] and [BZ+20]. For more details on the implementation side, see e.g., the article by Lao et al. [LSL+20].

The paper [RR01] provided some analysis regarding the mixing rate of random walk Metropolis methods. They showed that if the posterior is Gaussian, the asymptotically optimal value of σ^2 is $2.38^2/n$, where n is the dimensionality of x. This results in an acceptance rate of 0.234, which in this case, is the optimal tradeoff between exploring widely enough to cover the distribution without being rejected too often. For a more recent account of optimal acceptance rates for RWM, see also [Bed08].

The papers [TZ02] and [MKSK12] proposed that in the case where the target distribution is a posterior given some data, $p^*(x) = p(x \mid \mathcal{D})$, it is helpful to condition the proposal of MH not just on the previous hidden state, but also the visible data, i.e., to use $q(x' \mid x, \mathcal{D})$. This is called *data-driven MCMC*, and a detailed introduction of this method can be found in their publications.

One can change the parameters of the MH proposal as the algorithm is running to increase efficiency. This is called *adaptive MCMC*. This allows one to start with a broad covariance (say), allowing large moves through the space until a mode is found, followed by a narrowing of the covariance to ensure careful exploration of the region around the mode. However, one must be careful not to violate the Markov property; thus the parameters of the proposal should not depend on the entire history of the chain. It turns out that a sufficient condition to ensure this is that the adaption is 'faded out' gradually over time. See e.g., [AT08] for details.

It is necessary to start MCMC in an initial state that has non-zero probability. A natural approach is to first use an optimizer to find a local mode. However, at such points the gradients of the log joint are zero, which can cause problems for some gradient-based MCMC methods. Several alternatives for selecting the MCMC initial state was discussed in a tutorial by Andrieu and Thoms [AT08].

In Gibbs sampling, we can sample some of the nodes in parallel, without affecting correctness. In particular, suppose we can create a *coloring* of the (moralized) undirected graph, such that no two neighboring nodes have the same color. Then we can sample all the nodes of the same color in parallel, and cycle through the colors sequentially. Details about this approach can be found in [GLGG11]. In general, computing an optimal coloring is NP-complete, but we can use efficient heuristics such as those in [Kub04].

The fact that the acceptance rate is 100% does not necessarily mean that Gibbs will converge rapidly, since it only updates one coordinate at a time. In some cases we can efficiently sample groups of variables at a time. This is called *blocked Gibbs sampling*, and can make much bigger moves through the state space. The articles [JKK95] and [WY02] can be referred to for more detailed information about this approach. Besides, we can sometimes gain even greater speedups by analytically integrating out some of the unknown quantities. This is called a *collapsed Gibbs sampler*, and it tends to be more efficient, since it is sampling in a lower dimensional space. Some examples about how this method works are included in [Mur23, §12.3.8].

Lattice models and Ising models are all special cases of undirected graphical models, which are also called *Markov random fields* (MRFs). For more information about probabilistic calculation on MRFs and their properties, one can refer to [Mur23, §4.3].

Hamiltonian Monte Carlo were originally derived from physics [DKPR87, Nea93, Mac03, N$^+$11, Bet17]. The method was originally called *hybrid Monte Carlo* [DKPR87]. It was introduced to the statistics community by Neal [Nea93], and was renamed to Hamiltonian Monte Carlo in [Mac03]. For a more detailed theory and mathematical analysis about Hamiltonian Monte Carlo methods, one can refer to [BZ$^+$20]. In practice, there are many widely used libraries for applying HMC in stochastic inference, such as PyMC [APAC$^+$23] and TensorFlow Probability [DLT$^+$17].

In practice, to diagnose the MCMC convergence after sampling is also very important, although we did not go into much detail about this aspect. A thorough introduction about different metrics and techniques can be found in [Mur23, §12.6].

Exercises

A.1 *Random walk Metropolis.* In this exercise, we would like to sample from a given mixture of two Gaussians using Metropolis-Hastings algorithm with a random walk proposal. Please complete the MH algorithm in `src.py`, and check your implementation using the notebook `a-1.ipynb`.

A.2 *Gibbs sampling.* One application of Ising models is as a prior for binary image denoising problems. In particular, suppose y is a noisy version of image x, and we wish to compute the posterior $p(x \mid y) \propto p(x)p(y \mid x)$, where $p(x)$ is an Ising prior, and $p(y \mid x) = \prod_i p(y_i \mid x_i)$ is a per-site likelihood term. It is commonly assumed that the local likelihood term is Gaussian, i.e., $y_i \sim \mathcal{N}(x_i, \sigma^2)$.

(a) Derive the full conditional for the posterior $p(x_i \mid x_{-i}, y)$ required for implementing the Gibbs sampling procedure.

(b) Implement the Gibbs sampling procedure in `src.py` according to your derived full conditionals, and check your implementation using the notebook `a-2.ipynb`.

References

[AASSMDPC11] HH Avilés-Arriaga, LE Sucar-Succar, CE Mendoza-Durán, and LA Pineda-Cortés. A comparison of dynamic naive Bayesian classifiers and hidden Markov models for gesture recognition. *Journal of Applied Research and Technology*, 9(1):81–102, 2011.

[APAC$^+$23] Oriol Abril-Pla, Virgile Andreani, Colin Carroll, Larry Dong, Christopher J Fonnesbeck, Maxim Kochurov, Ravin Kumar, Junpeng Lao, Christian C Luhmann, Osvaldo A Martin, et al. PyMC: A modern, and comprehensive probabilistic programming framework in Python. *PeerJ Computer Science*, 9:e1516, 2023.

[AT08] Christophe Andrieu and Johannes Thoms. A tutorial on adaptive MCMC. *Statistics and Computing*, 18:343–373, 2008.

[Att03] Hagai Attias. Planning by probabilistic inference. In *International Workshop on Artificial Intelligence and Statistics*, pages 9–16. PMLR, 2003.

[Bed08] Mylene Bedard. Optimal acceptance rates for Metropolis algorithms: Moving beyond 0.234. *Stochastic Processes and their Applications*, 118(12):2198–2222, 2008.

[Bel57] Richard Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

[Bet17] Michael Betancourt. A conceptual introduction to Hamiltonian Monte Carlo. *arXiv preprint arXiv:1701.02434*, 2017.

[BM58] George EP Box and Mervin E Muller. A note on the generation of random normal deviates. *The Annals of Mathematical Statistics*, 29(2):610–611, 1958.

[BT12] Matthew Botvinick and Marc Toussaint. Planning as inference. *Trends in Cognitive Sciences*, 16(10):485–488, 2012.

[Bun91] Wray Buntine. Theory refinement on Bayesian networks. In *Uncertainty Proceedings 1991*, pages 52–60. Elsevier, 1991.

[BV04] Stephen P Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[BZ$^+$20] Adrian Barbu, Song-Chun Zhu, et al. *Monte Carlo Methods*, volume 35. Springer, 2020.

[CH92] Gregory F Cooper and Edward Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[CL68] CKCN Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

[Dar01] Adnan Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1-2):5–41, 2001.

[Dar09] Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks.* Cambridge University Press, 2009.

[Daw79] A Philip Dawid. Conditional independence in statistical theory. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 41(1):1–15, 1979.

[DD06] F Javier Dıez and Marek J Druzdzel. Canonical probabilistic models for knowledge engineering. *UNED, Madrid, Spain, Technical Report CISIAD-06*, 1, 2006.

[Díe96] Francisco Javier Díez. Local conditioning in Bayesian networks. *Artificial Intelligence*, 87(1-2):1–20, 1996.

[DKPR87] Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid Monte Carlo. *Physics Letters B*, 195(2):216–222, 1987.

[DLT$^+$17] Joshua V Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A Saurous. Tensorflow distributions. *arXiv preprint arXiv:1711.10604*, 2017.

[Fel50] William Feller. *Probability Theory and Its Applications.* Wiley, New York, 1950.

[FGG97] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29:131–163, 1997.

[GG84] Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, 1984.

[GLGG11] Joseph Gonzalez, Yucheng Low, Arthur Gretton, and Carlos Guestrin. Parallel Gibbs sampling: From colored fields to thin junction trees. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 324–332. JMLR Workshop and Conference Proceedings, 2011.

[GP93] Dan Geiger and Judea Pearl. Logical and algorithmic properties of conditional independence and graphical models. *The Annals of Statistics*, 21(4):2001–2021, 1993.

[GRS95] Walter R Gilks, Sylvia Richardson, and David Spiegelhalter. *Markov Chain Monte Carlo in Practice.* CRC Press, 1995.

[GS90] Alan E Gelfand and Adrian FM Smith. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, 85(410):398–409, 1990.

[Has70] W Keith Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 04 1970.

[Heg06] Pinar Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*, 306(3):297–317, 2006.

[HGC95] David Heckerman, Dan Geiger, and David M Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

[HJ12] Roger A Horn and Charles R Johnson. *Matrix Analysis.* Cambridge University Press, 2012.

[Hol73] John H Holland. Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing*, 2(2):88–105, 1973.

[How60] Ronald A Howard. *Dynamic Programming and Markov Processes.* John Wiley, 1960.

[HPS71] Paul Gerhard Hoel, Sidney C Port, and Charles Joel Stone. *Introduction to Probability Theory.* Houghton Mifflin Company, Boston, 1971.

[HR17] Christian Hoffmann and Philipp Rostalski. Linear optimal control on factor graphs — A message passing perspective. *IFAC-PapersOnLine*, 50(1):6314–6319, 2017.

[HY01] David J Hand and Keming Yu. Idiot's bayes — not so stupid after all? *International Statistical Review*, 69(3):385–398, 2001.

[JA13] Frank Jensen and SK Anderson. Approximations in Bayesian belief universe for knowledge based systems. *arXiv preprint arXiv:1304.1101*, 2013.

[JKK95] Claus S Jensen, Uffe Kjærulff, and Augustine Kong. Blocking Gibbs sampling in very large probabilistic expert systems. *International Journal of Human-Computer Studies*, 42(6):647–666, 1995.

[JL13] George H John and Pat Langley. Estimating continuous distributions in Bayesian classifiers. *arXiv preprint arXiv:1302.4964*, 2013.

[JN07] Finn V Jensen and Thomas Dyhre Nielsen. *Bayesian Networks and Decision Graphs*, volume 2. Springer, 2007.

[KBR18] Andreĭ Nikolaevich Kolmogorov and Albert T Bharucha-Reid. *Foundations of the Theory of Probability: Second English Edition.* Courier Dover Publications, 2018.

[KF09] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques.* MIT Press, 2009.

[KGJV83] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[KGO12] Hilbert J Kappen, Vicenç Gómez, and Manfred Opper. Optimal control as a graphical model inference problem. *Machine Learning*, 87:159–182, 2012.

[KN10] Kevin B Korb and Ann E Nicholson. *Bayesian Artificial Intelligence.* CRC Press, 2010.

[KS$^+$69] John G Kemeny, J Laurie Snell, et al. *Finite Markov chains*, volume 26. van Nostrand Princeton, NJ, 1969.

[KTB13] Dirk P Kroese, Thomas Taimre, and Zdravko I Botev. *Handbook of Monte Carlo Methods.* John Wiley & Sons, 2013.

[Kub04] Marek Kubale. *Graph Colorings*, volume 352. American Mathematical Soc., 2004.

[Lev18] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

[LL01] Jun S Liu and Jun S Liu. *Monte Carlo Strategies in Scientific Computing*, volume 10. Springer, 2001.

[LR19] Roderick JA Little and Donald B Rubin. *Statistical Analysis with Missing Data*, volume 793. John Wiley & Sons, 2019.

[LS88] Steffen L Lauritzen and David J Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 50(2):157–194, 1988.

[LSL$^+$20] Junpeng Lao, Christopher Suter, Ian Langmore, Cyril Chimisov, Ashish Saxena, Pavel Sountsov, Dave Moore, Rif A Saurous, Matthew D Hoffman, and Joshua V Dillon. tfp. mcmc: Modern Markov chain Monte Carlo tools built for modern hardware. *arXiv preprint arXiv:2002.01184*, 2020.

[Mac03] David JC MacKay. *Information Theory, Inference and Learning Algorithms.* Cambridge University Press, 2003.

[MAS06] Miriam Martinez-Arroyo and Luis Enrique Sucar. Learning an optimal naive Bayes classifier. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 3, pages 1236–1239. IEEE, 2006.

[MKSK12] Jihong Min, Jungho Kim, Seunghak Shin, and In So Kweon. Efficient data-driven MCMC sampling for vision-based 6D SLAM. In *2012 IEEE International Conference on Robotics and Automation*, pages 3025–3032. IEEE, 2012.

[MN$^+$98] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive Bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*, volume 752, pages 41–48. Madison, WI, 1998.

[MRR$^+$53] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[MSTC95] Donald Michie, David J Spiegelhalter, Charles C Taylor, and John Campbell. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, 1995.

[Mur23] Kevin P. Murphy. *Probabilistic Machine Learning: Advanced Topics.* MIT Press, 2023.

[MWJ13] Kevin Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference: An empirical study. *arXiv preprint arXiv:1301.6725*, 2013.

[N$^+$11] Radford M Neal et al. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11):2, 2011.

[Nea90] Richard E Neapolitan. *Probabilistic Reasoning in Expert Systems: Theory and Algorithms.* John Wiley & Sons, 1990.

[Nea93] Radford M Neal. Probabilistic inference using Markov chain Monte Carlo methods. *Technical Report CRG-TR-93-1*, 1993.

[Paz95] Michael Pazzani. Searching for attribute dependencies in Bayesian classifiers. In *Fifth International Workshop on Artificial Intelligence and Statistics*, pages 424–429. FL Ft. Lauderdale, 1995.

[Pea86] Judea Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29:241–288, 1986.

[Pea88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[Pea09] Judea Pearl. *Causality.* Cambridge University Press, 2009.

[PNM$^+$08] Olivier Pourret, Patrick Na, Bruce Marcot, et al. *Bayesian Networks: A Practical Guide to Applications.* John Wiley & Sons, 2008.

[PP87] Judea Pearl and Azaria Paz. Graphoids: A graph-based logic for reasoning about relevance relations. *Advances in Artificial Intelligence-II*, pages 357–363, 1987.

[PS98] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Courier Corporation, 1998.

[Put14] Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, 2014.

[RCC99] Christian P Robert, George Casella, and George Casella. *Monte Carlo Statistical Methods*, volume 2. Springer, 1999.

[RN16] Stuart J Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson, 2016.

[ROES18] Jesús Joel Rivas, Felipe Orihuela-Espina, and Luis Enrique Sucar. Circular chain classifiers. In *International Conference on Probabilistic Graphical Models*, pages 380–391. PMLR, 2018.

[RP87] George Rebane and Judea Pearl. The recovery of causal poly-trees from statistical data. In *Proceedings of the Third Conference on Uncertainty in Artificial Intelligence*, pages 222–228, 1987.

[RPHF11] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine Learning*, 85:333–359, 2011.

[RR01] Gareth O Roberts and Jeffrey S Rosenthal. Optimal scaling for various Metropolis-Hastings algorithms. *Statistical Science*, 16(4):351–367, 2001.

[RSTK03] Jason D Rennie, Lawrence Shih, Jaime Teevan, and David R Karger. Tackling the poor assumptions of naive Bayes text classifiers. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 616–623, 2003.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction.* MIT press, 2018.

[SBM$^+$14] L Enrique Sucar, Concha Bielza, Eduardo F Morales, Pablo Hernandez-Leal, Julio H Zaragoza, and Pedro Larrañaga. Multi-label classification with Bayesian network-based chain classifiers. *Pattern Recognition Letters*, 41:14–22, 2014.

[SGS01] Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search.* MIT Press, 2001.

[Spo80] Wolfgang Spohn. Stochastic independence, causal independence, and shieldability. *Journal of Philosophical Logic*, 9:73–99, 1980.

[SS90] Prakash P Shenoy and Glenn Shafer. Axioms for probability and belief-function propagation. In *Machine Intelligence and Pattern Recognition*, volume 9, pages 169–198. Elsevier, 1990.

[Suc21] Luis Enrique Sucar. *Probabilistic Graphical Models: Principles and Applications.* Springer, 2nd edition, 2021.

[Sup70] Patrick Suppes. *A Probabilistic Theory of Causality.* North-Holland Publishing Co., 1970.

[TK07] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13, 2007.

[Tou09] Marc Toussaint. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th International Conference on machine learning*, pages 1049–1056, 2009.

[TS06] Marc Toussaint and Amos Storkey. Probabilistic inference for solving discrete and continuous state Markov Decision Processes. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 945–952, 2006.

[TZ02] Zhuowen Tu and Song-Chun Zhu. Image segmentation by data-driven Markov chain Monte Carlo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):657–673, 2002.

[Vit67] Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

[Wes01] Douglas Brent West. *Introduction to Graph Theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[WY02] Darren J Wilkinson and Stephen KH Yeung. Conditional simulation from highly structured Gaussian systems, with application to blocking-MCMC for the Bayesian analysis of very large linear models. *Statistics and Computing*, 12(3):287–300, 2002.

[YW01] Ying Yang and Geoffrey I Webb. Proportional k-interval discretization for naive-Bayes classifiers. In *Machine Learning: ECML 2001: 12th European Conference on Machine Learning Freiburg, Germany, September 5–7, 2001 Proceedings 12*, pages 564–575. Springer, 2001.

Notation

Sets, vectors, and matrices

$\mathbf{R}$	Real numbers.
$\mathbf{R}^n$	Real n-vectors ($n \times 1$ matrices).
$\mathbf{R}^{m\times n}$	Real $m \times n$ matrices.
$\mathbf{R}_+, \mathbf{R}_{++}$	Nonnegative, positive real numbers.
$\mathbf{Z}$	Integers.
$\mathbf{Z}_+, \mathbf{Z}_{++}$	Nonnegative, positive integers.
$\mathbf{S}^n$	Symmetric $n \times n$ matrices.
$\mathbf{S}^n_+, \mathbf{S}^n_{++}$	Symmetric positive semidefinite, positive definite, $n \times n$ matrices.
$\mathbf{card}(C)$	Cardinality of set C.
I_C	Indicator function of set C.
$\mathbf{1}$	Vector with all components one.
e_i	ith standard basis vector.
$x \odot y$	Componentwise multiplication of vectors x and y.
I	Identity matrix.
$X_{i:}, X_{:i}$	The ith column/row of matrix X, represented as a column vector.
X^T	Transpose of matrix X.
X^k	(Square) matrix X to the kth power.
$\mathbf{tr}(X)$	Trace of matrix X.
$\mathbf{diag}(x)$	Diagonal matrix with diagonal entries $x_1, \ldots, x_n$.
$\mathbf{rank}(A)$	Rank of matrix A.

Functions and derivatives

$f\colon A \to B$	f is a function on the set $\mathbf{dom}(f) \subseteq A$ into the set B.
$\mathbf{dom}(f)$	Domain of function f.
∇f	Gradient of function f.
$\nabla^2 f$	Hessian of function f.

Norms and distances

$\|\cdot\|$	A norm.
$\|x\|_1$	l_1-norm of vector x.
$\|x\|_2$	Euclidean (or l_2-) norm of vector x.
$\|x\|_\infty$	l_∞-norm of vector x.
$\mathbf{dist}(A, B)$	Distance between sets (or points) A and B.

Generalized inequalities

$x \preceq y$	Componentwise inequality between vectors x and y.
$x \prec y$	Strict componentwise inequality between vectors x and y.

Probability

$\mathbf{P}(S)$	Probability of event S.
$(X \perp\!\!\!\perp Y \mid Z)$	Conditional independence of random variables X and Y given Z.
$\mathbf{E}[X]$	Expected value of random variable X.
$\mathbf{var}(X)$	Variance of random variable X.
$\sigma(X)$	Standard deviation of random variable X.
$\mathbf{cov}(X, Y)$	Covariance of random variables X and Y.
$\rho(X, Y)$	Correlation coefficient of random variables X and Y.
$r(X, Y)$	Regression coefficient of random variables X and Y.
$p(x)$	Density function of continuous random variable X.
$F(x)$	Cumulative distribution function of continuous random variable X.
$l_x(\theta)$	Log-likelihood function of θ given the observation $X = x$.
$\mathbf{supp}(p)$	Support of density function p.
$\mathcal{N}(\mu, \sigma^2)$	Gaussian distribution with mean μ, variance σ^2.
$\Phi(x)$	Cumulative distribution function of $\mathcal{N}(0, 1)$ random variable X.
$\mathcal{U}(x, y)$	Uniform distribution on interval $[x, y]$.
$\mathrm{Exp}(\lambda)$	Exponential distribution with parameter λ.
$\mathrm{Beta}(\alpha, \beta)$	Beta distribution with shape parameters α and β.
$\mathrm{Dir}(\alpha)$	Dirichlet distribution with concentration parameters $\alpha = (\alpha_1, \ldots, \alpha_k)$.

Graph

$\mathbf{pa}(V)$	Parents of node V.
$\mathbf{mb}(V)$	Markov blanket of node V.
$\mathbf{adj}(V)$	Adjacent nodes of node V.