Computer vision: models, learning and inference

Simon J.D. Prince

July 7, 2012

The most recent version of this book can be downloaded from http://www.computervisionmodels.com.

Please mail errata to s.prince@cs.ucl.ac.uk.

This book is dedicated to Richard Eagle, without whom it would never have been started, and to Lynfa Stroud, without whom it would never have been finished.

Contents

Acknowledgments

I am incredibly grateful to the following people who read parts of this book and gave me feedback: Yun Fu, David Fleet, Alan Jepson, Marc'Aurelio Ranzato, Gabriel Brostow, Oisin Mac Aodha, Xiwen Chen, Po-Hsiu Lin, Jose Tejero Alonso, Amir Sani, Oswald Aldrian, Sara Vicente, Jozef Doboš, Andrew Fitzgibbon, Michael Firman, Gemma Morgan, Daniyar Turmukhambetov, Daniel Alexander, Mihaela Lapusneanu, John Winn, Petri Hiltunen, Jania Aghajanian, Alireza Bossaghzadeh, Mikhail Sizintsev, Roger De Souza-Eremita, Jacques Cali, Roderick de Nijs, James Tompkin, Jonathan O'Keefe, Benedict Kuester, Tom Hart, Marc Kerstein, Alex Borés, Marius Cobzarenco, Luke Dodd, Ankur Agarwal, Ahmad Humayun, Andrew Glennerster, Steven Leigh, Matteo Munaro, Peter van Beek, Hu Feng, Martin Parsley, Jordi Salvador Marcos, Josephine Sullivan, Steve Thompson, Laura Panagiotaki, Damien Teney, Malcolm Reynolds, Francisco Estrada, Peter Hall, James Elder, Paria Mehrani, Vida Movahedi, Eduardo Corral Soto, Ron Tal, Bob Hou, Simon Arridge, Norberto Goussies, Steve Walker, Tracy Petrie, Kostantinos Derpanis, Bernard Buxton, Matthew Pediaditis, Fernando Flores-Mangas, Jan Kautz, Alastair Moore, Yotam Doron, Tahir Majeed, David Barber, Pedro Quelhas, Wenchao Zhang, Alan Angold, Andrew Davison, Alex Yakubovich, Fatemeh Jamali, David Lowe, Ricardo David, Jamie Shotton, Andrew Zisserman, Sanchit Singh, Vincent Lepetit, David Liu, Marc Pollefeys, Christos Panagiotou, Ying Li, Shoaib Ehsan, Olga Veksler, Modesto Castrillón Santana, Axel Pinz, Matteo Zanotto, Gwynfor Jones, Brian Jensen, Mischa Schirris, Jacek Zienkiewicz, Erik Sudderth, Etienne Beauchesne, Moos Hueting, Giovanni Saponaro, Phi Hung Nguyen, Tran Duc Hieu, Simon Julier, Oscar Plag and Thomas Hoyoux. This book is much better because of your selfless efforts!

I am also especially grateful to Sven Dickinson, who hosted me at the University of Toronto for nine months during the writing of this book; Stephen Boyd, who let me use his beautiful LaTeX template; and Mikhail Sizintsev, for his help in summarizing the bewildering literature on dense stereo vision. I am extremely indebted to Gabriel Brostow, who read the entire draft and spent hours of his valuable time discussing it with me. Finally, I am grateful to Bernard Buxton, who taught me most of this material in the first place and has supported my career in computer vision and every stage.

Foreword

I was very pleased to be asked to write this foreword, having seen snapshots of the development of this book since its inception. I write this having just returned from BMVC 2011, where I found that others had seen draft copies, and where I heard comments like "What amazing figures!", "It's so comprehensive!", and "He's so Bayesian!".

But I don't want you to read this book just because it has amazing figures, and provides new insights into vision algorithms of every kind, or even because it's "Bayesian" (although more on that later). I want you to read it because it makes clear the most important distinction in computer vision research: the difference between "model" and "algorithm". This is akin to the distinction that Marr made with his three-level computational theory, but Prince's two-level distinction is made beautifully clear by his use of the language of probability.

Why is this distinction so important? Well, let us look at one of the oldest and apparently easiest problems in vision: separating an image into "figure" and "ground." It is still common to hear students new to vision address this problem just as the early vision researchers did, by reciting an algorithm: first I'll use PCA to find the dominant color axis, then I'll generate a grayscale image, then I'll threshold that at some value, then I'll clean up the holes using morphological operators. Trying their recipe on some test images, the novice discovers that real images are rather more complicated, so new steps are added: I'll need some sort of adaptive threshold, I can get that by blurring the edge map and locally computing maxima.

However, as most readers will already know, such recipes are extremely brittle, meaning that the various "magic numbers" controlling each step all interact, making it impossible to find a set of parameters that works for all images (or even a useful subset). The root of this problem is that the objective of the algorithm has never been defined. What do we *mean* by figure and ground separation? Can we specify what we mean mathematically?

When vision researchers began to address these problems, the language of statistics and Markov random fields allowed a clean distinction between the objective and the algorithm to be drawn. We write down not the steps to solve the problem, but the problem itself, for example as a function to be minimized. In the language of this book, we write down formulae for all the probability distributions that define the problem and then perform operations on those distributions in order to provide answers. This book shows how this can be done for a huge variety of vision problems, and how doing so provides more robust solutions that are much easier to reason about.

This is not to say that one can just write down the model and ask others to solve for its parameters, because the space of possible models is so much vaster than the space of ones in which the solution is tractable. Thus, one always has at the back of one's mind a collection of models known to be soluble, and one always tries to find a model for one's problem, which is nearby some soluble one. At that stage, one may well think in terms of strategies such as "I can probably generalize alpha expansion a bit to solve for the discrete

parameters, and then I can use a Gauss-Newton method for the continuous ones, and that will probably be slow, but it will tell me if it's worth trying to invent a faster combined algorithm". Such strategies are common and can be helpful, providing one always retains an idea of the model underlying them.

However, even armed with the attitudes this book will engender, experienced researchers today can fall into the trap of failing to distinguish model and algorithm. They find themselves thinking thoughts like: "I'll fit a mixture of Gaussians to the color distribution. Then I'll model the mixture weights as an MRF and use graph cuts to update them. Then I'll go back to step 1 and repeat." The good news is that often such recipes can be turned back into models. Even if the only known way of fitting the model is to use the recipe you just thought of, the discipline of thinking of it as a model allows you to reason about it, to make use of alternative techniques, and ultimately to do better research. Reading this book is a sure way to improve your ability to make that jump.

So what is this language of probabilities that will allow us to become better researchers? Well, let me provide my "Engineer's view of Bayes' theorem." It is common to hear a distinction between "Bayesians" and "Frequentists", but I think many engineers have a much more fundamental problem with Bayes: Bayesians must lie. Their estimates, biased toward the prior mean, are deliberately different from the most probable reading of their sensor. Consider the example of an "I speak your height" machine whose sensor has a uniformly distributed ± 1 cm error. You receive £1 every time you correctly predict someone's height to within 1 cm. Bayesian principles suggest that if your sensor reads 200cm ± 1 cm, you should report 199 cm; you will make more money than guessing the actual sensor reading, because more 199 cm people will appear than those of 200 cm. So I, as an engineer, believe in Bayes as a way of getting better answers, and thus very much welcome this book's pragmatic (but much more subtle than mine) embrace of Bayes. I wonder if it might even be considered a book on statistics with vision examples rather than a book on vision built on probability.

But it would be wrong to finish this foreword without mentioning the figures. They really are good, not because they're beautiful (they often are), but because they provide crucial insights into the workings of even the most basic of algorithms and ideas. The illustrations in chapters 2 to 4 are fundamental to the understanding of modern Bayesian inference, and yet I doubt that there are more than a handful of researchers who have ever seen them all. Later figures express extremely complex ideas more clearly than I have ever seen, as well as representing fabulously "clean" implementations of fundamental algorithms, which really show us how the underlying models influence our capabilities.

Finally I believe it is worth directly comparing this book to the recent textbook by my colleague Richard Szeliski. That book too is marked by an enormously comprehensive view of computer vision, by excellent illustration, by insightful notation, and intellectual synthesis of large groups of existing ideas. But in a real sense the two books operate at opposite ends of the pedagogical spectrum: Szeliski is a comprehensive summary of the state of the art in computer vision, the frontier of our knowledge and abilities, while this book addresses the fundamentals of how we make progress in this challenging and exciting field. I look forward to many decades with both on my shelf, or indeed, I suspect, open on my desktop.

Andrew Fitzgibbon
September 2011

Preface

There are already many computer vision textbooks, and it is reasonable to question the need for another. Let me explain why I chose to write this volume.

Computer vision is an engineering discipline; we are primarily motivated by the real-world concern of building machines that see. Consequently, we tend to categorize our knowledge by the real-world problem that it addresses. For example, most existing vision textbooks contain chapters on object recognition and stereo vision. The sessions at our research conferences are organized in the same way. The role of this book is to question this orthodoxy: is this really the way that we should organize our knowledge?

Consider the topic of object recognition. A wide variety of methods have been applied to this problem (e.g., subspace models, boosting methods, bag of words models, and constellation models). However, these approaches have little in common. Any attempt to describe the grand sweep of our knowledge devolves into an unstructured list of techniques. How can we make sense of it all for a new student? I will argue for a different way to organize our knowledge, but first let me tell you how I see computer vision problems.

We observe an image and from this we extract *measurements*. For example, we might use the RGB values directly or we might filter the image or perform some more sophisticated preprocessing. The *vision problem* or *goal* is to use the measurements to infer the *world state*. For example, in stereo vision we try to infer the depth of the scene. In object detection we attempt to infer the presence or absence of a particular class of object.

To accomplish the goal, we build a *model*. The model describes a family of statistical relationships between the measurements and the world state. The particular member of that family is determined by a set of *parameters*. In *learning* we choose these parameters so they accurately reflect the relationship between the measurements and the world. In *inference* we take a new set of measurements and use the model to tell us about the world state. The methods for learning and inference are embodied in *algorithms*. I believe that computer vision should be understood in these terms: the goal, the measurements, the world state, the model, the parameters, and the learning and inference algorithms.

We could choose to organize our knowledge according to any of these quantities, but in my opinion what is most critical is the model itself – the statistical relationship between the world and the measurements. There are three reasons for this. First, the model type often transcends the application (the same model can be used for diverse vision tasks). Second, the models naturally organize themselves neatly into distinct families (e.g., regression, Markov random fields, camera models) that can be understood in relative isolation. Finally, discussing vision on the level of models allows us to draw connections between algorithms and applications that initially appear unrelated. Accordingly, this book is organized so that each main chapter considers a different family of models.

On a final note, I should say that I found most of the ideas in this book very hard to grasp when I was first exposed to them. My goal was to make this process easier for subsequent students following the same path; I hope that this book achieves this and inspires the reader to learn more about computer vision.

Chapter 1

Introduction

The goal of computer vision is to extract useful information from images. This has proved a surprisingly challenging task; it has occupied thousands of intelligent and creative minds over the last four decades, and despite this we are still far from being able to build a general-purpose "seeing machine."

Part of the problem is the complexity of visual data. Consider the image in figure 1.1. There are hundreds of objects in the scene. Almost none of these are presented in a "typical" pose. Almost all of them are partially occluded. For a computer vision algorithm, it is not even easy to establish where one object ends and another begins. For example, there is almost no change in the image intensity at the boundary between the sky and the white building in the background. However, there is a pronounced change in intensity on the back window of the SUV in the foreground, although there is no object boundary or change in material here.

We might have grown despondent about our chances of developing useful computer vision algorithms if it were not for one thing: we have concrete proof that vision is possible because our own visual systems make light work of complex images such as figure 1.1. If I ask you to count the trees in this image, or to draw me a sketch of the street layout, you can do this easily. You might even be able to pinpoint where this photo was taken on a world map by extracting subtle visual clues such as the ethnicity of the people, the types of cars and trees, and the weather.

So, computer vision is not impossible, but it is very challenging; perhaps this was not appreciated at first because what we perceive when we look at a scene is already highly processed. For example, consider observing a lump of coal in bright sunlight and then moving to a dim indoor environment and looking at a piece of white paper. The eye will receive far more photons per unit area from the coal than from the paper, but we nonetheless perceive the coal as black and the paper as white. The visual brain performs many tricks of this kind, but unfortunately when we build vision algorithms we do not have the benefit of this preprocessing.

Nonetheless, there has been remarkable recent progress in our understanding of computer vision, and the last decade has seen the first large scale deployments of consumer computer vision technology. For example, most digital cameras now have embedded algorithms for face detection, and at the time of writing the Microsoft Kinect (a peripheral that allows real-time tracking of the human body) holds the

Figure 1.1 A visual scene containing many objects, almost all of which are partially occluded. The red circle indicates a part of the scene where there is almost no brightness change to indicate the boundary between the sky and the building. The green circle indicates a region in which there is a large intensity change but this is due to irrelevant lighting effects; there is no object boundary or change in the object material here.

Guinness World Record for being the fastest-selling consumer electronics device ever. The principles behind both of these applications and many more are explained in this book.

There are a number of reasons for the rapid recent progress in computer vision. The most obvious is that the processing power, memory, and storage capacity of computers has vastly increased; before we disparage the progress of early computer vision pioneers, we should pause to reflect that they would have needed specialized hardware to hold even a single high-resolution image in memory. Another reason for the recent progress in this area has been the increased use of machine learning. The last 20 years have seen exciting developments in this parallel research field, and these are now deployed widely in vision applications. Not only has machine learning provided many useful tools, it has also helped us understand existing algorithms and their connections in a new light.

The future of computer vision is exciting. Our understanding grows by the day, and it is likely that artificial vision will become increasingly prevalent in the next

decade. However, this is still a young discipline. Until recently, it would have been unthinkable to even try to work with complex scenes such as that in figure 1.1. As Szeliski (2010) puts it, "It may be many years before computers can name and outline all of the objects in a photograph with the same skill as a two year old child." However, this book provides a snapshot of what we have achieved and the principles behind these achievements.

Organization of the book

The structure of this book is illustrated in figure 1.2. It is divided into six parts.

The first part of the book contains background information on probability. All the models in this book are expressed in terms of probability, which is a useful language for describing computer vision applications. Readers with a rigorous background in engineering mathematics will know much of this material already but should skim these chapters to ensure they are familiar with the notation. Those readers who do not have this background should read these chapters carefully. The ideas are relatively simple, but they underpin everything else in the rest of the book. It may be frustrating to be forced to read fifty pages of mathematics before the first mention of computer vision, but please trust me when I tell you that this material will provide a solid foundation for everything that follows.

The second part of the book discusses machine learning for machine vision. These chapters teach the reader the core principles that underpin all of our methods to extract useful information from images. We build statistical models that relate the image data to the information that we wish to retrieve. After digesting this material, the reader should understand how to build a model to solve almost any vision problem, although that model may not yet be very practical.

The third part of the book introduces graphical models for computer vision. Graphical models provide a framework for simplifying the models that relate the image data to the properties we wish to estimate. When both of these quantities are high dimensional, the statistical connections between them become impractically complex; we can still define models that relate them, but we may not have the training data or computational power to make them useful. Graphical models provide a principled way to assert sparseness in the statistical connections between the data and the world properties.

The fourth part of the book discusses image preprocessing. This is not necessary to understand most of the models in the book, but that is not to say that it is unimportant. The choice of preprocessing method is at least as critical as the choice of model in determining the final performance of a computer vision system. Although image processing is not the main topic of this book, this section provides a compact summary of the most important and practical techniques.

The fifth part of the book concerns geometric computer vision; it introduces the projective pinhole camera – a mathematical model that describes where a given point in the 3D world will be imaged in the pixel array of the camera. Associated with this model are a set of techniques for finding the position of the camera relative to a scene and for reconstructing 3D models of objects.

Finally, in the sixth part of the book, we present several families of vision models

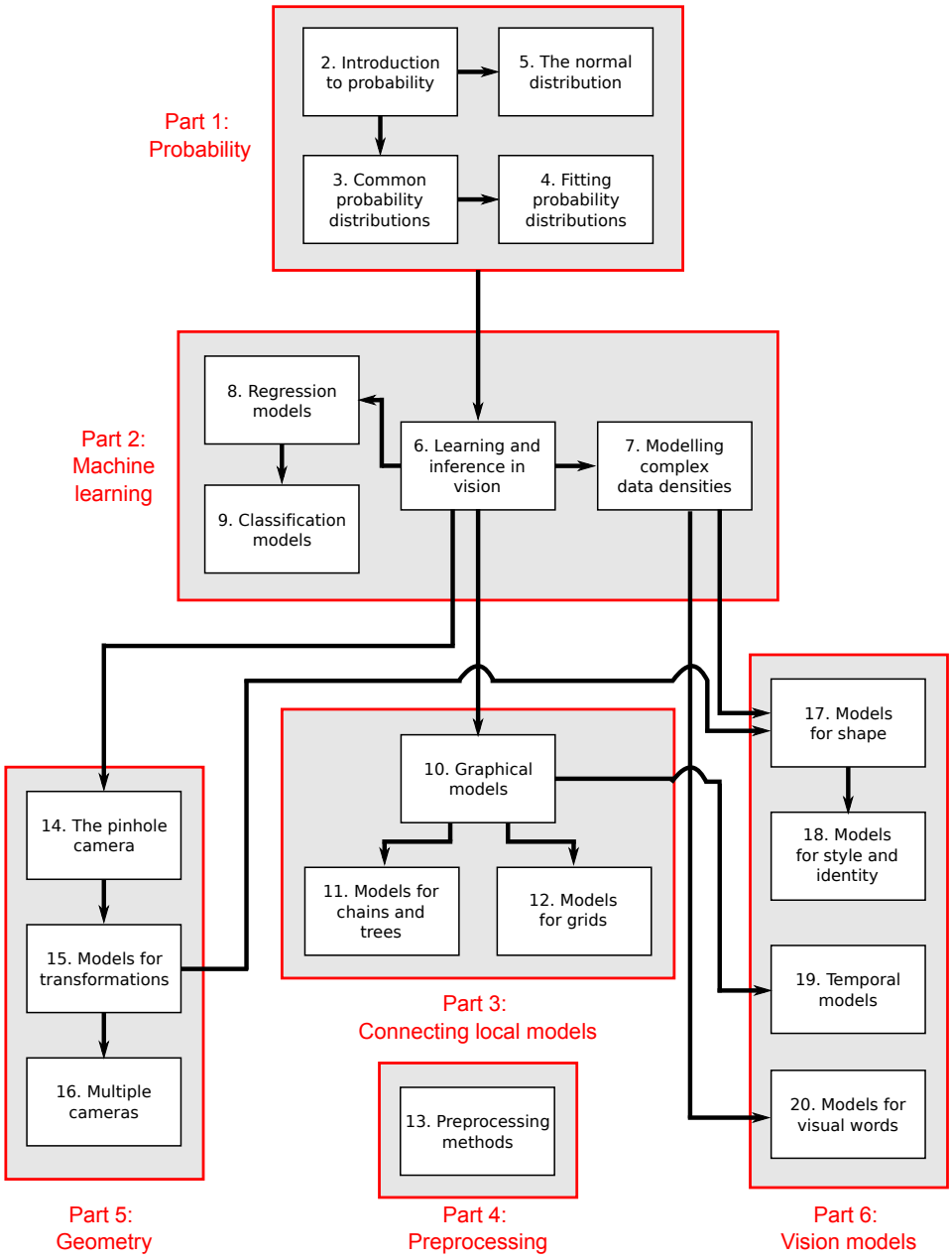


Figure 1.2 Chapter dependencies. The book is organized into six sections. The first section is a review of probability and is necessary for all subsequent chapters. The second part concerns machine learning and inference. It describes both generative and discriminative models. The third part concerns graphical models: visual representations of the probabilistic dependencies between variables in large models. The fourth part describes preprocessing methods. The fifth part concerns geometry and transformations. Finally, the sixth part presents several other important families of vision models.

that build on the principles established earlier in the book. These models address some of the most central problems in computer vision including face recognition, tracking, and object recognition.

The book concludes with several appendices. There is a brief discussion of the notational conventions used in the book, and compact summaries of linear algebra and optimization techniques. Although this material is widely available elsewhere, it makes the book more self-contained and is discussed in the same terminology as the main text.

At the end of every chapter is a brief notes section. This provides details of the related research literature. It is heavily weighted toward the most useful and recent papers and does not reflect an accurate historical description of each area. There are also a number of exercises for the reader at the end of each chapter. In some cases, important but tedious derivations have been excised from the text and turned into problems to retain the flow of the main argument. Here, the solution will be posted on the main book website (http://www.computervisionmodels.com). A series of applications are also presented at the end of each chapter (apart from chapters 1-5 and chapter 10 which contain only theoretical material). Collectively, these represent a reasonable cross-section of the important vision papers of the last decade.

Finally, pseudocode for over 70 of the algorithms discussed is available and can be downloaded in a separate document from the associated website. This pseudocode uses the same notation as the book and will make it easy to implement many of the models. I chose not to include this in the main text because it would have decreased the readability. However, I encourage all readers of this book to implement as many of the models as possible. Computer vision is a practical engineering discipline and you can learn a lot by experimenting with real code.

Other books

I am aware that most people will not learn computer vision from this book alone so here is some advice about other books that complement this volume. To learn more about machine learning and graphical models I would recommend *Pattern Recognition and Machine Learning* by Bishop (2006) as a good starting point. There are many books on preprocessing, but my favorite is *Feature Extraction and Image Processing* by Nixon & Aguado (2008). The best source for information about geometrical computer vision is without a doubt *Multiple View Geometry* by Hartley & Zisserman (2004). Finally, for a much more comprehensive overview of the state of the art of computer vision and its historical development, consider *Computer Vision: Algorithms and Applications* by Szeliski (2010).

Part I

Probability

Part I: Probability

We devote the first part of this book (chapters 2–5) to a brief review of probability and probability distributions. Almost all models for computer vision can be interpreted in a probabilistic context, and in this book we will present all the material in this light. The probabilistic interpretation may initially seem confusing, but it has a great advantage: it provides a common notation that will be used throughout the book and will elucidate relationships between different models that would otherwise remain opaque.

So why is probability a suitable language to describe computer vision problems? In a camera, the three-dimensional world is projected onto the optical surface to form the image: a two-dimensional set of measurements. Our goal is to take these measurements and use them to establish the properties of the world that created them. However, there are two problems. First, the measurement process is noisy; what we observe is not the amount of light that fell on the sensor, but a noisy estimate of this quantity. We must describe the noise in these data, and for this we use probability. Second, the relationship between world and measurements is generally many to one: there may be many real-world configurations that are compatible with the same measurements. The chance that each of these possible worlds is present can also be described using probability.

The structure of part I is as follows: in chapter 2, we introduce the basic rules for manipulating probability distributions including the ideas of conditional and marginal probability and Bayes' rule. We also introduce more advanced ideas such as independence and expectation.

In chapter 3, we discuss the properties of eight specific probability distributions. We divide these into four pairs. The first set will be used to describe either the observed data or the state of the world. The second set of distributions model the parameters of the first set. In combination, they allow us to fit a probability model and provide information about how certain we are about the fit.

In chapter 4, we discuss methods for fitting probability distributions to observed data. We also discuss how to assess the probability of new data points under the fitted model and how to take account of uncertainty in the fitted model when we do this. Finally, in chapter 5, we investigate the properties of the multivariate normal distribution in detail. This distribution is ubiquitous in vision applications and has a number of useful properties that are frequently exploited in machine vision.

Readers who are very familiar with probability models and the Bayesian philosophy may wish to skip this part and move directly to part II.

Chapter 2

Introduction to probability

In this chapter, we provide a compact review of probability theory. There are very few ideas, and each is relatively simple when considered separately. However, they combine to form a powerful language for describing uncertainty.

2.1 Random variables

A random variable x denotes a quantity that is uncertain. The variable may denote the result of an experiment (e.g., flipping a coin) or a real-world measurement of a fluctuating property (e.g., measuring the temperature). If we observe several instances $\{x_i\}_{i=1}^{I}$, then it might take a different value on each occasion. However, some values may occur more often than others. This information is captured by the probability distribution $Pr(x)$ of the random variable.

A random variable may be *discrete* or *continuous*. A discrete variable takes values from a predefined set. This set may be ordered (the outcomes 1–6 of rolling a die) or unordered (the outcomes "sunny," "raining," "snowing," upon observing the weather). It may be finite (there are 52 possible outcomes of drawing a card randomly from a standard pack) or infinite (the number of people on the next train is theoretically unbounded). The probability distribution of a discrete variable can be visualized as a histogram or a Hinton diagram (figure 2.1). Each outcome has a positive probability associated with it and the sum of the probabilities for all outcomes is always one.

Continuous random variables take values that are real numbers. These may be finite (the time taken to finish a 2-hour exam is constrained to be greater than 0 hours and less than 2 hours) or infinite (the amount of time until the next bus arrives is unbounded above). Infinite continuous variables may be defined on the whole real range or may be bounded above or below (the 1D velocity of a vehicle may take any value, but the speed is bounded below by 0). The probability distribution of a continuous variable can be visualized by plotting the *probability density function* (pdf). The probability density for an outcome represents the relative propensity of the random variable to take that value (see figure 2.2). It may take any positive value. However, the integral of the pdf always sums to one.

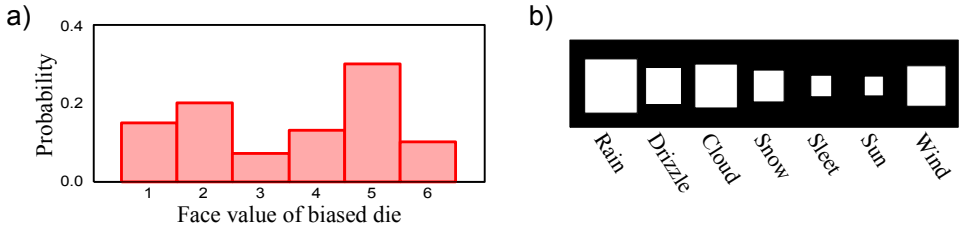
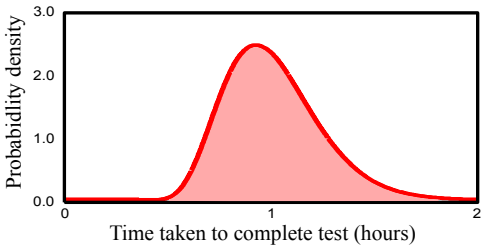


Figure 2.1 Two different representations for discrete probabilities a) A bar graph representing the probability that a biased six-sided die lands on each face. The height of the bar represents the probability: the sum of all heights is one. b) A Hinton diagram illustrating the probability of observing different weather types in England. The area of the square represents the probability, so the sum of all areas is one.

Figure 2.2 Continuous probability distribution (probability density function or pdf for short) for time taken to complete a test. Note that the probability density can exceed one, but the area under the curve must always have unit area.



2.2 Joint probability

Problem 2.1

Consider two random variables, x and y. If we observe multiple paired instances of x and y, then some combinations of the two outcomes occur more frequently than others. This information is encompassed in the *joint* probability distribution of x and y, which is written as $Pr(x, y)$. The comma in $Pr(x, y)$ can be read as the English word "and" so $Pr(x, y)$ is the probability of x *and* y. A joint probability distribution may relate variables that are all discrete or all continuous, or it may relate discrete variables to continuous ones (see figure 2.3). Regardless, the total probability of all outcomes (summing over discrete variables and integrating over continuous ones) is always one.

In general, we will be interested in the joint probability distribution of more than two variables. We will write $Pr(x, y, z)$ to represent the joint probability distribution of scalar variables x, y, and z. We may also write $Pr(\mathbf{x})$ to represent the joint probability of all of the elements of the multidimensional variable $\mathbf{x} = [x_1, x_2, \ldots, x_K]^T$. Finally, we will write $Pr(\mathbf{x}, \mathbf{y})$ to represent the joint distribution of all of the elements from multidimensional variables $\mathbf{x}$ and $\mathbf{y}$.

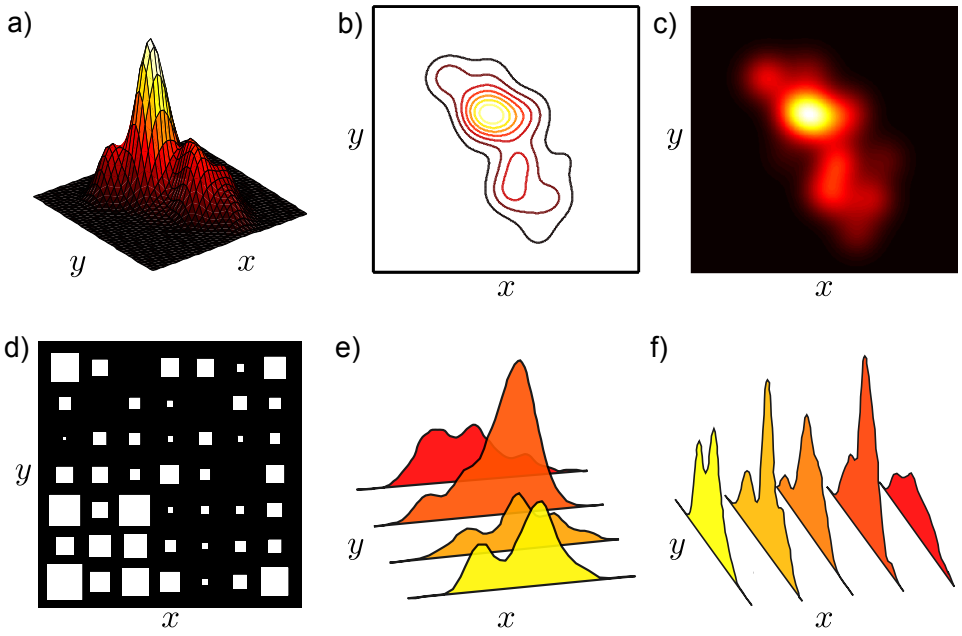


Figure 2.3 Joint probability distributions between variables x and y. a-c) The same joint pdf of two continuous variables represented as a surface, contour plot, and image, respectively. d) Joint distribution of two discrete variables represented as a 2D Hinton diagram. e) Joint distribution of a continuous variable x and discrete variable y. f) Joint distribution of a discrete variable x and continuous variable y.

2.3 Marginalization

We can recover the probability distribution of any single variable from a joint distribution by summing (discrete case) or integrating (continuous case) over all the other variables (figure 2.4). For example, if x and y are both continuous and we know $Pr(x, y)$, then we can recover the distributions $Pr(x)$ and $Pr(y)$ using the relations

$$\begin{aligned} Pr(x) &= \int Pr(x, y)\, dy, \\ Pr(y) &= \int Pr(x, y)\, dx. \end{aligned} \tag{2.1}$$

The recovered distributions $Pr(x)$ and $Pr(y)$ are referred to as *marginal* distributions, and the process of integrating/summing over the other variables is called *marginalization*. Calculating the marginal distribution $Pr(x)$ from the joint distribution $Pr(x, y)$ by marginalizing over the variable y has a simple interpretation:

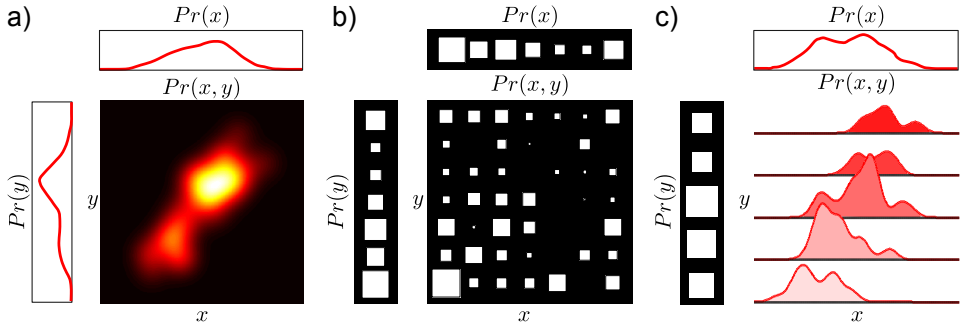


Figure 2.4 Joint and marginal probability distributions. The marginal probability $Pr(x)$is found by summing over all values of y (discrete case) or integrating over y (continuous case) in the joint distribution $Pr(x, y)$. Similarly, the marginal probability $Pr(y)$ is found by summing or integrating over x. Note that the plots for the marginal distributions have different scales from those for the joint distribution (on the same scale, the marginals would look larger as they sum all of the mass from one direction). a) Both x and y are continuous. b) Both x and y are discrete. c) The random variable x is continuous and the variable y is discrete.

we are finding the probability distribution of x regardless of (or in the absence of information about) the value of y.

Problem 2.2

In general, we can recover the joint probability of any subset of variables, by marginalizing over all of the others. For example, given variables, w, x, y, z, where w is discrete and z is continuous, we can recover $Pr(x, y)$ using

$$Pr(x, y) = \sum_{w} \int Pr(w, x, y, z)\, dz. \tag{2.2}$$

2.4 Conditional probability

The conditional probability of x given that y takes value y^* tells us the relative propensity of the random variable x to take different outcomes given that the random variable y is fixed to value y^*. This conditional probability is written as $Pr(x|y = y^*)$. The vertical line "|" can be read as the English word "given."

The conditional probability $Pr(x|y = y^*)$ can be recovered from the joint distribution $Pr(x, y)$. In particular, we examine the appropriate slice $Pr(x, y = y^*)$ of the joint distribution (figure 2.5). The values in the slice tell us about the relative probability that x takes various values having observed $y = y^*$, but they do not themselves form a valid probability distribution; they cannot sum to one as they constitute only a small part of the joint distribution which did itself sum to one. To calculate the conditional probability distribution, we hence normalize by the total probability in the slice:

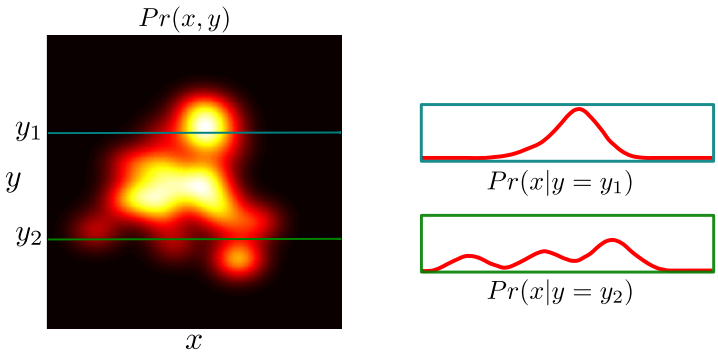


Figure 2.5 Conditional probability. Joint pdf of x and y and two conditional probability distributions $Pr(x|y=y_1)$ and $Pr(x|y=y_2)$. These are formed by extracting the appropriate slice from the joint pdf and normalizing so that the area is one. A similar operation can be performed for discrete distributions.

$$Pr(x|y=y^*) = \frac{Pr(x,y=y^*)}{\int Pr(x,y=y^*)dx} = \frac{Pr(x,y=y^*)}{Pr(y=y^*)}, \tag{2.3}$$

where we have used the marginal probability relation (Equation 2.1) to simplify the denominator. It is common to write the conditional probability relation without explicitly defining the value $y=y^*$ to give the more compact notation

$$Pr(x|y) = \frac{Pr(x,y)}{Pr(y)}. \tag{2.4}$$

This relationship can be re-arranged to give

$$Pr(x,y) = Pr(x|y)Pr(y), \tag{2.5}$$

and by symmetry we also have

$$Pr(x,y) = Pr(y|x)Pr(x). \tag{2.6}$$

When we have more than two variables, we may repeatedly take conditional probabilities to divide up the joint probability distribution into a product of terms

Problem 2.3

$$\begin{aligned} Pr(w,x,y,z) &= Pr(w,x,y|z)Pr(z) \\ &= Pr(w,x|y,z)Pr(y|z)Pr(z) \\ &= Pr(w|x,y,z)Pr(x|y,z)Pr(y|z)Pr(z). \end{aligned} \tag{2.7}$$

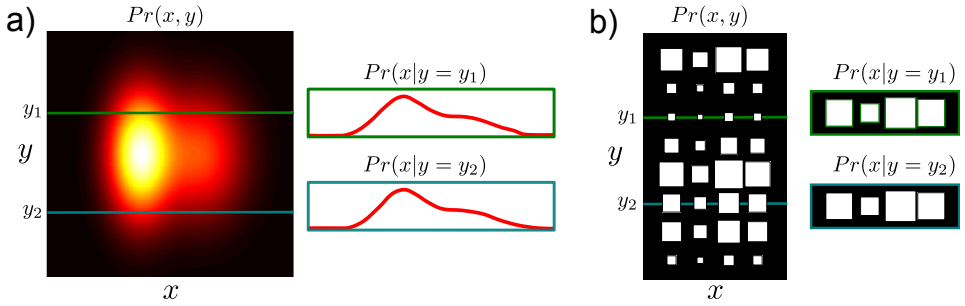


Figure 2.6 Independence. a) Joint pdf of continuous independent variables x and y. The independence of x and y means that every conditional distribution is the same: the value of y tells us nothing about x and vice-versa. Compare this to figure 2.5 which illustrated variables that were dependent. b) Joint distribution of discrete independent variables x and y. The conditional distributions of x given y are all the same.

2.5 Bayes' rule

In equations 2.5 and 2.6 we expressed the joint probability in two ways. We can combine these formulations to find a relationship between $Pr(x|y)$ and $Pr(y|x)$,

$$Pr(y|x)Pr(x) = Pr(x|y)Pr(y), \tag{2.8}$$

or, rearranging, we have

$$\begin{aligned} Pr(y|x) &= \frac{Pr(x|y)Pr(y)}{Pr(x)} \\ &= \frac{Pr(x|y)Pr(y)}{\int Pr(x,y)\, dy} \\ &= \frac{Pr(x|y)Pr(y)}{\int Pr(x|y)Pr(y)\, dy}, \end{aligned} \tag{2.9}$$

Problem 2.4

where in the second and third lines we have expanded the denominator using the definitions of marginal and conditional probability, respectively. These three equations are all commonly referred to as *Bayes' rule*.

Each term in Bayes' rule has a name. The term $Pr(y|x)$ on the left-hand side is the *posterior*. It represents what we know about y given x. Conversely, the term $Pr(y)$ is the *prior* as it represents what is known about y before we consider x. The term $Pr(x|y)$ is the *likelihood* , and the denominator $Pr(x)$ is the *evidence*.

In computer vision, we often describe the relationship between variables x and y in terms of the conditional probability $Pr(x|y)$. However, we may be primarily interested in the variable y, and in this situation Bayes' rule is exploited to compute the probability $Pr(y|x)$.

2.6 Independence

If knowing the value of variable x tells us nothing about variable y (and vice-versa) then we say x and y are independent (figure 2.6). Here, we can write

Problem 2.5

$$\begin{aligned} Pr(x|y) &= Pr(x) \\ Pr(y|x) &= Pr(y). \end{aligned} \tag{2.10}$$

Substituting into equation 2.5, we see that for independent variables the joint probability $Pr(x, y)$ is the product of the marginal probabilities $Pr(x)$ and $Pr(y)$,

Problem 2.6
Problem 2.7

$$Pr(x, y) = Pr(x|y)Pr(y) = Pr(x)Pr(y). \tag{2.11}$$

2.7 Expectation

Given a function $f[\bullet]$ that returns a value for each possible value x^* of the variable x and a probability $Pr(x = x^*)$ that each value of x occurs, we sometimes wish to calculate the *expected* output of the function. If we drew a very large number of samples from the probability distribution, calculated the function for each sample, and took the average of these values, the result would be the *expectation*. More precisely, the expected value of a function $f[\bullet]$ of a random variable x is defined as

Problem 2.8

$$\begin{aligned} \mathrm{E}[f[x]] &= \sum_x f[x]Pr(x), \\ \mathrm{E}[f[x]] &= \int f[x]Pr(x)\, dx, \end{aligned} \tag{2.12}$$

for the discrete and continuous cases, respectively. This idea generalizes to functions $f[\bullet]$ of more than one random variable so that, for example,

$$\mathrm{E}[f[x, y]] = \iint f[x, y]Pr(x, y)\, dx\, dy. \tag{2.13}$$

For some choices of the function $f[\bullet]$, the expectation is given a special name (table 2.1). Such quantities are commonly used to summarize the properties of complex probability distributions.

There are four rules for manipulating expectations, which can be easily proved from the original definition (equation 2.12):

Problem 2.9
Problem 2.10

1. The expected value of a constant κ with respect to the random variable x is just the constant itself:
$$\mathrm{E}[\kappa] = \kappa. \tag{2.14}$$

Function $f[\bullet]$	Expectation
x	mean, μ_x
x^k	k^{th} moment about zero
$(x-\mu_x)^k$	k^{th} moment about the mean
$(x-\mu_x)^2$	variance
$(x-\mu_x)^3$	skew
$(x-\mu_x)^4$	kurtosis
$(x-\mu_x)(y-\mu_y)$	covariance of x and y

Table 2.1 Special cases of expectation. For some functions $f(x)$, the expectation $\mathrm{E}[f(x)]$ is given a special name. Here we use the notation μ_x to represent the mean with respect to random variable x and μ_y the mean with respect to random variable y.

2. The expected value of a constant κ times a function $f[x]$ of the random variable x is κ times the expected value of the function:

$$\mathrm{E}[\kappa f[x]] = \kappa \mathrm{E}[f[x]]. \tag{2.15}$$

3. The expected value of the sum of two functions of a random variable x is the sum of the individual expected values of the functions:

$$\mathrm{E}[f[x] + g[x]] = \mathrm{E}[f[x]] + \mathrm{E}[g[x]]. \tag{2.16}$$

4. The expected value of the product of two functions $f[x]$ and $g[y]$ of random variables x and y is equal to the product of the individual expected values if the variables x and y are independent:

$$\mathrm{E}[f[x]g[y]] = \mathrm{E}[f[x]]\mathrm{E}[g[y]], \qquad \text{if } x, y \text{ independent.} \tag{2.17}$$

Discussion

The rules of probability are remarkably compact and simple. The concepts of marginalization, joint and conditional probability, independence, and Bayes' rule will underpin all of the machine vision algorithms in this book. There is one remaining important concept related to probability, which is *conditional independence*. We discuss this at length in chapter 10.

Notes

For a more formal discussion of probability, the reader is encouraged to investigate one of the many books on this topic (e.g., Papoulis 1991). For a view of probability from a machine learning perspective, consult the first chapter of Bishop (2006).

Problems

Problem 2.1 Give a real-world example of a joint distribution $Pr(x, y)$ where x is discrete and y is continuous.

Problem 2.2 What remains if I marginalize a joint distribution $Pr(v, w, x, y, z)$ over five variables with respect to variables w and y? What remains if I marginalize the resulting distribution with respect to v?

Problem 2.3 Show that the following relation is true:

$$Pr(w, x, y, z) = Pr(x, y)Pr(z|w, x, y)Pr(w|x, y).$$

Problem 2.4 In my pocket there are two coins. Coin 1 is unbiased, so the likelihood $Pr(h = 1|c = 1)$ of getting heads is 0.5 and the likelihood $Pr(h = 0|c = 1)$ of getting tails is also 0.5. Coin 2 is biased, so the likelihood $Pr(h = 1|c = 2)$ of getting heads is 0.8 and the likelihood $Pr(h = 0|c = 2)$ of getting tails is 0.2. I reach into my pocket and draw one of the coins at random. There is an equal prior probability I might have picked either coin. I flip the coin and observe a head. Use Bayes' rule to compute the posterior probability that I chose coin 2.

Problem 2.5 If variables x and y are independent and variables x and z are independent, does it follow that variables y and z are independent?

Problem 2.6 Use equation 2.3 to show that when x and y are independent, the marginal distribution $Pr(x)$ is the same as the conditional distribution $Pr(x|y = y^*)$ for any y^*.

Problem 2.7 The joint probability $Pr(w, x, y, z)$ over four variables factorizes as

$$Pr(w, x, y, z) = Pr(w)Pr(z|y)Pr(y|x, w)Pr(x).$$

Demonstrate that x is independent of w by showing that $Pr(x, w) = Pr(x)Pr(w)$.

Problem 2.8 Consider a biased die where the probabilities of rolling sides $\{1, 2, 3, 4, 5, 6\}$ are $\{1/12, 1/12, 1/12, 1/12, 1/6, 1/2\}$, respectively. What is the expected value of the die? If I roll the die twice, what is the expected value of the sum of the two rolls?

Problem 2.9 Prove the four relations for manipulating expectations.

$$\begin{aligned}
\mathrm{E}[\kappa] &= \kappa, \\
\mathrm{E}[\kappa f[x]] &= \kappa \mathrm{E}[f[x]], \\
\mathrm{E}[f[x] + g[x]] &= \mathrm{E}[f[x]] + \mathrm{E}[g[x]], \\
\mathrm{E}[f[x]g[y]] &= \mathrm{E}[f[x]]\mathrm{E}[g[y]], \qquad \text{if } x, y \text{ independent.}
\end{aligned}$$

For the last case, you will need to use the definition of independence (see section 2.6).

Problem 2.10 Use the relations from problem 2.9 to prove the following relationship between the second moment around zero and the second moment about the mean (variance):

$$E\left[(x-\mu)^2\right] = E\left[x^2\right] - \mathrm{E}[x]\mathrm{E}[x].$$

Chapter 3

Common probability distributions

In chapter 2 we introduced abstract rules for manipulating probabilities. To use these rules we will need to define some probability distributions. The choice of distribution $Pr(x)$ that we use will depend on the *domain* of the data x that we are modeling (table 3.1).

Data Type	**Domain**	**Distribution**
univariate, discrete, binary	$x \in \{0, 1\}$	Bernoulli
univariate, discrete, multi-valued	$x \in \{1, 2, \ldots, K\}$	categorical
univariate, continuous, unbounded	$x \in \mathbb{R}$	univariate normal
univariate, continuous, bounded	$x \in [0, 1]$	beta
multivariate, continuous, unbounded	$\mathbf{x} \in \mathbb{R}^K$	multivariate normal
multivariate, continuous, bounded, sums to one	$\mathbf{x} = [x_1, x_2, \ldots, x_K]^T$ $x_k \in [0, 1], \sum_{k=1}^{K} x_k = 1$	Dirichlet
bivariate, continuous, x_1 unbounded, x_2 bounded below	$\mathbf{x} = [x_1, x_2]$ $x_1 \in \mathbb{R}$ $x_2 \in \mathbb{R}^+$	normal-scaled inverse gamma
multivariate vector $\mathbf{x}$ and matrix $\mathbf{X}$, $\mathbf{x}$ unbounded, $\mathbf{X}$ square, positive definite	$\mathbf{x} \in \mathbb{R}^K$ $\mathbf{X} \in \mathbb{R}^{K \times K}$ $\mathbf{z}^T\mathbf{X}\mathbf{z} > 0 \quad \forall\, \mathbf{z} \in \mathbb{R}^K$	normal inverse Wishart

Table 3.1: Common probability distributions: the choice of distribution depends on the type/domain of data to be modeled.

Probability distributions such as the categorical and normal distributions are obviously useful for modeling visual data. However, the need for some of the other

distributions is not so obvious; for example, the Dirichlet distribution models K positive numbers that sum to one. Visual data do not normally take this form.

The explanation is as follows: when we fit probability models to data, we need to know how uncertain we are about the fit. This uncertainty is represented as a probability distribution over the parameters of the fitted model. So for each distribution used for modeling, there is a second distribution over the associated parameters (table 3.2). For example, the Dirichlet is used to model the parameters of the categorical distribution. In this context, the parameters of the Dirichlet would be known as *hyperparameters*. More generally, the hyperparameters determine the shape of the distribution over the parameters of the original distribution.

Distribution	Domain	Parameters modeled by
Bernoulli	$x \in \{0,1\}$	beta
categorical	$x \in \{1,2,\ldots,K\}$	Dirichlet
univariate normal	$x \in \mathbb{R}$	normal inverse gamma
multivariate normal	$\mathbf{x} \in \mathbb{R}^k$	normal inverse Wishart

Table 3.2: Common distributions used for modeling (left) and their associated domains (center). For each of these distributions there is a second associated distribution over the parameters (right).

We will now work through the distributions in table 3.2 before looking more closely at the relationship between these pairs of distributions.

3.1 Bernoulli distribution

The *Bernoulli distribution* (figure 3.1) is a discrete distribution that models binary trials: it describes the situation where there are only two possible outcomes $x \in \{0,1\}$ which are referred to as "failure" and "success." In machine vision, the Bernoulli distribution could be used to model the data. For example, it might describe the probability of a pixel taking an intensity value of greater or less than 128. Alternatively, it could be used to model the state of the world. For example, it might describe the probability that a face is present or absent in the image.

Problem 3.1

The Bernoulli has a single parameter $\lambda \in [0,1]$ which defines the probability of observing a success $x = 1$. The distribution is hence

$$\begin{aligned} Pr(x=0) &= 1-\lambda \\ Pr(x=1) &= \lambda. \end{aligned} \tag{3.1}$$

We can alternatively express this as

$$Pr(x) = \lambda^x (1-\lambda)^{1-x}, \tag{3.2}$$

and we will sometimes use the equivalent notation

$$Pr(x) = \text{Bern}_x[\lambda]. \tag{3.3}$$

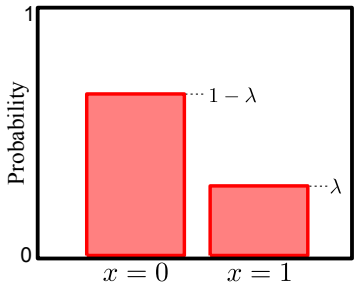


Figure 3.1 Bernoulli distribution. The Bernoulli distribution is a discrete distribution with two possible outcomes, $x \in \{0,1\}$ which are referred to as failure and success, respectively. It is governed by a single parameter λ that determines the probability of success such that $Pr(x=0) = 1-\lambda$ and $Pr(x=1) = \lambda$.

3.2 Beta distribution

The *beta distribution* (figure 3.2) is a continuous distribution defined on single variable λ where $\lambda \in [0,1]$. As such it is suitable for representing uncertainty in the parameter λ of the Bernoulli distribution.

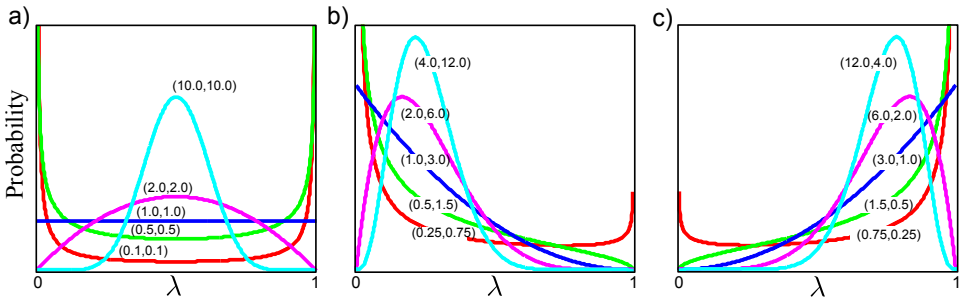


Figure 3.2 Beta distribution. The beta distribution is defined on $[0,1]$ and has parameters (α,β) whose relative values determine the expected value so $\mathrm{E}[\lambda] = \alpha/(\alpha+\beta)$ (numbers in parentheses show the α,β for each curve). As the absolute values of (α,β) increase, the concentration around $\mathrm{E}[\lambda]$ increases. a) $\mathrm{E}[\lambda] = 0.5$ for each curve, concentration varies. b) $\mathrm{E}[\lambda] = 0.25$. c) $\mathrm{E}[\lambda] = 0.75$.

The beta distribution has two parameters $\alpha,\beta \in [0,\infty]$ which both take positive values and affect the shape of the curve as indicated in figure 3.2. Mathematically, the beta distribution has the form

Problem 3.2
Problem 3.3
Problem 3.4

$$Pr(\lambda) = \frac{\Gamma[\alpha+\beta]}{\Gamma[\alpha]\Gamma[\beta]}\lambda^{\alpha-1}(1-\lambda)^{\beta-1}, \tag{3.4}$$

where $\Gamma[\bullet]$ is the *gamma function*[1]. For short, we abbreviate this to

Problem 3.5

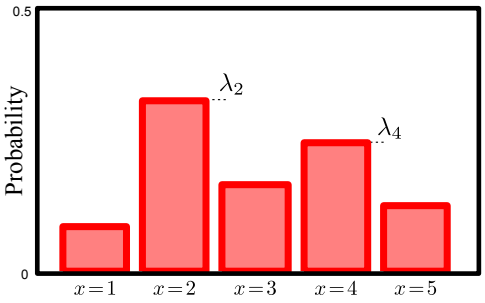


Figure 3.3 The categorical distribution is a discrete distribution with K possible outcomes, $x \in \{1, 2 \ldots, K\}$ and K parameters $\lambda_1, \lambda_2 \ldots, \lambda_K$ where $\lambda_k \geq 0$ and $\sum_k \lambda_k = 1$. Each parameter represents the probability of observing one of the outcomes, so that the probability of observing $x = k$ is given by λ_k. When the number of possible outcomes K is 2, the categorical reduces to the Bernoulli distribution.

$$Pr(\lambda) = \text{Beta}_\lambda[\alpha, \beta]. \tag{3.5}$$

3.3 Categorical distribution

The *categorical distribution* (figure 3.3) is a discrete distribution that determines the probability of observing one of K possible outcomes. Hence, the Bernoulli distribution is a special case of the categorical distribution when there are only two outcomes. In machine vision the intensity data at a pixel is usually quantized into discrete levels and so can be modeled with a categorical distribution. The state of the world may also take one of several discrete values. For example an image of a vehicle might be classified into {car,motorbike,van,truck} and our uncertainty over this state could be described by a categorical distribution.

The probabilities of observing the K outcomes are held in a $K \times 1$ parameter vector $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \ldots, \lambda_K]$, where $\lambda_k \in [0, 1]$ and $\sum_{k=1}^{K} \lambda_k = 1$. The categorical distribution can be visualized as a normalized histogram with K bins and can be written as

$$Pr(x = k) = \lambda_k. \tag{3.6}$$

For short, we use the notation

$$Pr(x) = \text{Cat}_x[\boldsymbol{\lambda}]. \tag{3.7}$$

Alternatively, we can think of the data as taking values $\mathbf{x} \in \{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_K\}$ where $\mathbf{e}_k$ is the k^{th} unit vector; all elements of $\mathbf{e}_k$ are zero except the k^{th}, which is one. Here we can write

$$Pr(\mathbf{x} = \mathbf{e}_k) = \prod_{j=1}^{K} \lambda_j^{x_j} = \lambda_k, \tag{3.8}$$

where x_j is the j^{th} element of $\mathbf{x}$.

[1] The gamma function is defined as $\Gamma[z] = \int_0^\infty t^{z-1} e^{-t} dt$ and is closely related to factorials, so that for positive integers $\Gamma[z] = (z-1)!$ and $\Gamma[z+1] = z\Gamma[z]$.

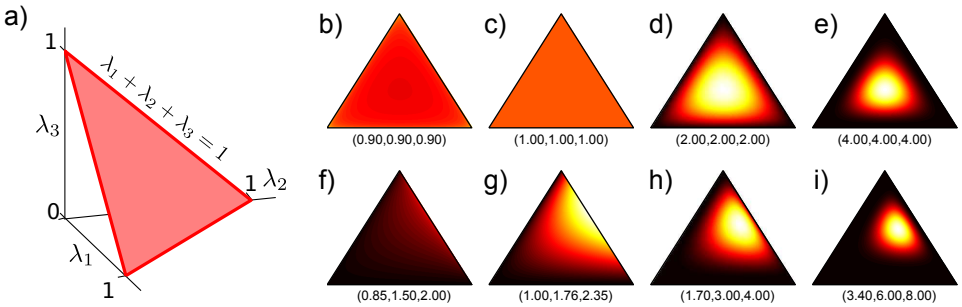


Figure 3.4 The Dirichlet distribution in K dimensions is defined on values $\lambda_1, \lambda_2, \ldots, \lambda_K$ such that $\sum_k \lambda_k = 1$ and $\lambda_k \in [0,1] \;\; \forall \;\; k \in \{1 \ldots K\}$. a) For K=3, this corresponds to a triangular section of the plane $\sum_k \lambda_k = 1$. In K dimensions, the Dirichlet is defined by K positive parameters $\alpha_{1\ldots K}$. The ratio of the parameters determines the expected value for the distribution. The absolute values determine the concentration: the distribution is highly peaked around the expected value at high parameter values but pushed away from the expected value at low parameter values. b-e) Ratio of parameters is equal, absolute values increase. f-i) Ratio of parameters favors $\alpha_3 > \alpha_2 > \alpha_1$, absolute values increase.

3.4 Dirichlet distribution

The *Dirichlet distribution* (figure 3.4) is defined over K continuous values $\lambda_1 \ldots \lambda_K$ where $\lambda_k \in [0,1]$ and $\sum_{k=1}^{K} \lambda_k = 1$. Hence it is suitable for defining a distribution over the parameters of the categorical distribution.

In K dimensions the Dirichlet distribution has K parameters $\alpha_1 \ldots \alpha_K$ each of which can take any positive value. The relative values of the parameters determine the expected values $\mathrm{E}[\lambda_1] \ldots \mathrm{E}[\lambda_k]$. The absolute values determine the concentration around the expected value. We write

$$Pr(\lambda_{1\ldots K}) = \frac{\Gamma[\sum_{k=1}^{K} \alpha_k]}{\prod_{k=1}^{K} \Gamma[\alpha_k]} \prod_{k=1}^{K} \lambda_k^{\alpha_k - 1}, \tag{3.9}$$

or for short

$$Pr(\lambda_{1\ldots K}) = \mathrm{Dir}_{\lambda_{1\ldots K}}[\alpha_{1\ldots K}]. \tag{3.10}$$

Just as the Bernoulli distribution was a special case of the categorical distribution with two possible outcomes, so the beta distribution is a special case of the Dirichlet distribution where the dimensionality is two.

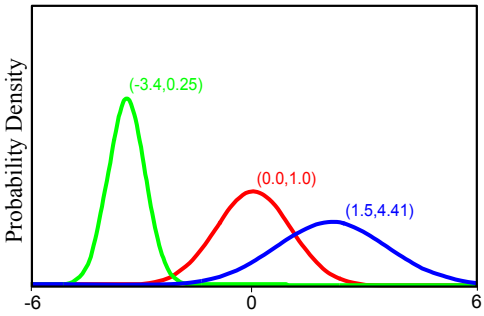


Figure 3.5 The univariate normal distribution is defined on $x \in \mathbb{R}$ and has two parameters $\{\mu, \sigma^2\}$. The mean parameter μ determines the expected value and the variance σ^2 determines the concentration about the mean so that as σ^2 increases, the distribution becomes wider and flatter.

3.5 Univariate normal distribution

The *univariate normal* or *Gaussian distribution* (figure 3.5) is defined on continuous values $x \in [-\infty, \infty]$. In vision, it is common to ignore the fact that the intensity of a pixel is quantized and model it with the continuous normal distribution. The world state may also be described by the normal distribution. For example, the distance to an object could be represented in this way.

Problem 3.6
Problem 3.7

The normal distribution has two parameters, the mean μ and the variance σ^2. The parameter μ can take any value and determines the position of the peak. The parameter σ^2 takes only positive values and determines the width of the distribution. The normal distribution is defined as

$$Pr(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-0.5(x-\mu)^2/\sigma^2\right], \tag{3.11}$$

and we will abbreviate this by writing

$$Pr(x) = \text{Norm}_x[\mu, \sigma^2]. \tag{3.12}$$

3.6 Normal-scaled inverse gamma distribution

The *normal-scaled inverse gamma distribution* (figure 3.6) is defined over a pair of continuous values μ, σ^2, the first of which can take any value and the second of which is constrained to be positive. As such it can define a distribution over the mean and variance parameters of the normal distribution.

Problem 3.8

The normal-scaled inverse gamma has four parameters $\alpha, \beta, \gamma, \delta$ where α, β, and γ are positive real numbers but δ can take any value. It has pdf:

$$Pr(\mu, \sigma^2) = \frac{\sqrt{\gamma}}{\sigma\sqrt{2\pi}} \frac{\beta^\alpha}{\Gamma[\alpha]} \left(\frac{1}{\sigma^2}\right)^{\alpha+1} \exp\left[-\frac{2\beta + \gamma(\delta-\mu)^2}{2\sigma^2}\right], \tag{3.13}$$

or for short

$$Pr(\mu, \sigma^2) = \text{NormInvGam}_{\mu,\sigma^2}[\alpha, \beta, \gamma, \delta]. \tag{3.14}$$

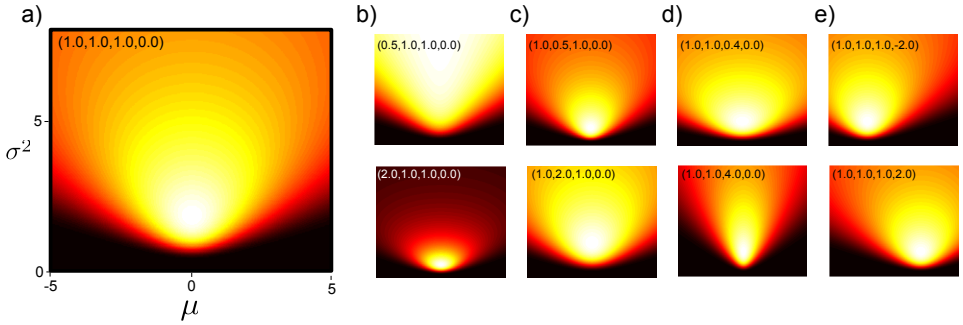


Figure 3.6 The normal-scaled inverse gamma distribution defines a probability distribution over bivariate continuous values μ, σ^2 where $\mu \in [-\infty, \infty]$ and $\sigma^2 \in [0, \infty]$. a) Distribution with parameters $[\alpha, \beta, \gamma, \delta] = [1, 1, 1, 0]$. b) Varying α. c) Varying β. d) Varying γ. e) Varying δ.

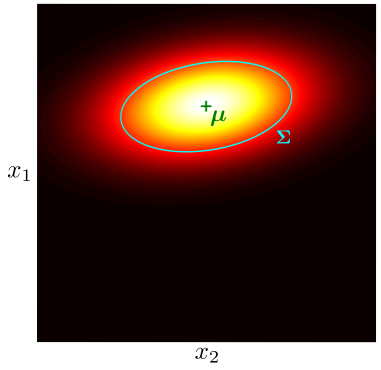


Figure 3.7 The multivariate normal distribution models D-dimensional variables $\mathbf{x} = [x_1 \ldots x_D]^T$ where each dimension x_d is continuous and real. It is defined by a $D \times 1$ vector $\boldsymbol{\mu}$ defining the mean of the distribution and a $D \times D$ covariance matrix $\boldsymbol{\Sigma}$ which determines the shape. The iso-contours of the distribution are ellipsoids where the center of the ellipsoid is determined by $\boldsymbol{\mu}$ and the shape by $\boldsymbol{\Sigma}$. This figure depicts a bivariate distribution, where the covariance is illustrated by drawing one of these ellipsoids.

3.7 Multivariate normal distribution

The *multivariate normal* or Gaussian distribution models D-dimensional variables $\mathbf{x}$ where each of the D elements $x_1 \ldots x_D$ is continuous and lies in the range $[-\infty, +\infty]$ (figure 3.7). As such the univariate normal distribution is a special case of the multivariate normal where the number of elements D is one. In machine vision the multivariate normal might model the joint distribution of the intensities of D pixels within a region of the image. The state of the world might also be described by this distribution. For example, the multivariate normal might describe the joint uncertainty in the 3D position (x, y, z) of an object in the scene.

The multivariate normal distribution has two parameters: the mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$. The mean $\boldsymbol{\mu}$ is a $D \times 1$ vector that describes the mean of the distribution. The covariance $\boldsymbol{\Sigma}$ is a symmetric $D \times D$ positive definite matrix so that $\mathbf{z}^T\boldsymbol{\Sigma}\mathbf{z}$ is positive for any real vector $\mathbf{z}$. The probability density function has the following form

$$Pr(\mathbf{x}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-0.5(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right], \tag{3.15}$$

or for short

$$Pr(\mathbf{x}) = \text{Norm}_{\mathbf{x}}\left[\boldsymbol{\mu}, \boldsymbol{\Sigma}\right]. \tag{3.16}$$

The multivariate normal distribution will be used extensively throughout this book, and we devote the whole of chapter 5 to describing its properties.

3.8 Normal inverse Wishart distribution

The *normal inverse Wishart distribution* defines a distribution over a $D \times 1$ vector $\boldsymbol{\mu}$ and a $D \times D$ positive definite matrix $\boldsymbol{\Sigma}$. As such it is suitable for describing uncertainty in the parameters of a multivariate normal distribution. The normal inverse Wishart has four parameters $\alpha, \boldsymbol{\Psi}, \gamma, \boldsymbol{\delta}$, where α and γ are positive scalars, $\boldsymbol{\delta}$ is a $D \times 1$ vector and $\boldsymbol{\Psi}$ is a positive definite $D \times D$ matrix

$$Pr(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{\gamma^{D/2}|\boldsymbol{\Psi}|^{\alpha/2}\exp\left[-0.5\left(\text{Tr}[\boldsymbol{\Psi}\boldsymbol{\Sigma}^{-1}] + \gamma(\boldsymbol{\mu}-\boldsymbol{\delta})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}-\boldsymbol{\delta})\right)\right]}{2^{\alpha D/2}(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{(\alpha+D+2)/2}\Gamma_D[\alpha/2]}, \tag{3.17}$$

where $\Gamma_D[\bullet]$ is the multivariate gamma function and $\text{Tr}[\boldsymbol{\Psi}]$ returns the trace of the matrix $\boldsymbol{\Psi}$ (see appendix C.2.4). For short we will write:

$$Pr(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \text{NorIWis}_{\boldsymbol{\mu},\boldsymbol{\Sigma}}\left[\alpha, \boldsymbol{\Psi}, \gamma, \boldsymbol{\delta}\right]. \tag{3.18}$$

The mathematical form of the normal inverse Wishart distribution is rather opaque. However, it is just a function that produces a positive value for any valid mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, such that when we integrate over all possible values of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, the answer is one. It is hard to visualize the normal inverse Wishart, but easy to draw samples and examine them: each sample is the mean and covariance of a normal distribution (figure 3.8).

3.9 Conjugacy

We have argued that the beta distribution can represent probabilities over the parameters of the Bernoulli. Similarly the Dirichlet defines a distribution over the parameters of the categorical, and there are analogous relationships between the normal-scaled inverse gamma and univariate normal and the normal inverse Wishart and the multivariate normal.

These pairs were carefully chosen because they have a special relationship: in each case, the former distribution is *conjugate* to the latter: the beta is *conjugate* to the Bernoulli and the Dirichlet is conjugate to the categorical and so on. When

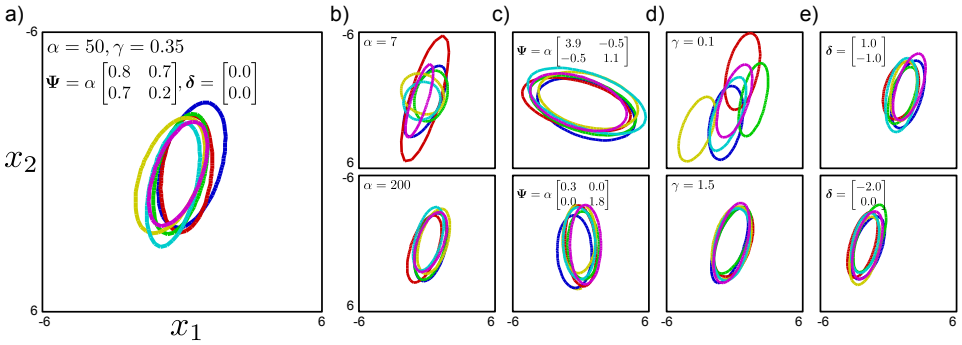


Figure 3.8 Sampling from 2D normal inverse Wishart distribution. a) Each sample consists of a mean vector and covariance matrix, here visualized with 2D ellipses illustrating the iso-contour of the associated Gaussian at a fixed Mahalanobis distance (i.e., a fixed proportion of the way through the density from the mean). b) Changing α modifies the dispersion of covariances observed. c) Changing $\boldsymbol{\Psi}$ modifies the average covariance. d) Changing γ modifies the dispersion of mean vectors observed. e) Changing $\boldsymbol{\delta}$ modifies the average value of the mean vectors.

we multiply a distribution with its conjugate, the result is proportional to a new distribution which has the same form as the conjugate. For example

Problem 3.9
Problem 3.10
Problem 3.11
Problem 3.12

$$\text{Bern}_x[\lambda] \cdot \text{Beta}_\lambda[\alpha, \beta] = \kappa(x, \alpha, \beta) \cdot \text{Beta}_\lambda\left[\tilde{\alpha}, \tilde{\beta}\right], \tag{3.19}$$

where κ is a scaling factor that is constant with respect to the variable of interest, λ. It is important to realize that this was not necessarily the case: if we had picked any distribution other than the beta, then this product would not have retained the same form. For this case, the relationship in equation 3.19 is easy to prove

$$\begin{aligned}
\text{Bern}_x[\lambda] \cdot \text{Beta}_\lambda[\alpha, \beta] &= \lambda^x(1-\lambda)^{1-x}\frac{\Gamma[\alpha+\beta]}{\Gamma[\alpha]\Gamma[\beta]}\lambda^{\alpha-1}(1-\lambda)^{\beta-1} \\
&= \frac{\Gamma[\alpha+\beta]}{\Gamma[\alpha]\Gamma[\beta]}\lambda^{x+\alpha-1}(1-\lambda)^{1-x+\beta-1} \\
&= \frac{\Gamma[\alpha+\beta]}{\Gamma[\alpha]\Gamma[\beta]}\frac{\Gamma[x+\alpha]\Gamma[1-x+\beta]}{\Gamma[x+\alpha+1-x+\beta]}\text{Beta}_\lambda[x+\alpha, 1-x+\beta] \\
&= \kappa(x, \alpha, \beta) \cdot \text{Beta}_\lambda\left[\tilde{\alpha}, \tilde{\beta}\right].
\end{aligned} \tag{3.20}$$

where in the third line we have both multiplied and divided by the constant associated with $\text{Beta}_\lambda[\tilde{\alpha}, \tilde{\beta}]$.

The conjugate relationship is important because we take products of distributions during both learning (fitting distributions) and evaluating the model (assessing probability of new data under fitted distribution). The conjugate relationship means that these products can both be computed neatly in closed form.

Summary

We use probability distributions to describe both the world state and the image data. We have presented four distributions (Bernoulli, categorical, univariate normal, multivariate normal) that are suited to this purpose. We also presented four other distributions (beta, Dirichlet, normal-scaled inverse gamma, and normal inverse Wishart) that can be used to describe the uncertainty in parameters of the first; they can hence describe the uncertainty in the fitted model. These four pairs of distributions have a special relationship: each distribution from the second set is conjugate to one from the first set. As we shall see, the conjugate relationship makes it easier to fit these distributions to observed data and evaluate new data under the fitted model.

Notes

Throughout this book, I use rather esoteric terminology for discrete distributions. I distinguish between the *binomial distribution* (probability of getting M successes in N binary trials) and the *Bernoulli distribution* (the binary trial itself or probability of getting a success or failure in one trial) and talk exclusively about the latter distribution. I take a similar approach to discrete variables which can take K values. The *multinomial distribution* assigns a probability to observing the values {1,2,..., K} with frequency $\{M_1, M_2, \ldots, M_K\}$ given N trials. The *categorical distribution* is a special case of this with $N = 1$. Most other authors do not make this distinction and would term this 'multinomial' as well.

A more complete list of common probability distributions and details of their properties are given in Appendix B of Bishop (2006). Further information about conjugacy can be found in Chapter 2 of Bishop (2006) or any textbook on Bayesian methods, such as that of Gelman *et al.* (2004). Much more information about the normal distribution is provided in chapter 5 of this book.

Problems

Problem 3.1 Consider a variable x which is Bernoulli distributed with parameter λ. Show that the mean $\mathrm{E}[x]$ is λ and the variance $\mathrm{E}[(x - \mathrm{E}[x])^2]$ is $\lambda(1 - \lambda)$.

Problem 3.2 Calculate an expression for the mode (position of the peak) of the beta distribution with $\alpha, \beta > 1$ in terms of the parameters α and β.

Problem 3.3 The mean and variance of the beta distribution are given by the expressions

$$\begin{aligned} E[\lambda] = \mu &= \frac{\alpha}{\alpha+\beta} \\ E[(\lambda-\mu)^2] = \sigma^2 &= \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}. \end{aligned}$$

We may wish to choose the parameters α and β so that the distribution has a particular mean μ and variance σ^2. Derive suitable expressions for α and β in terms of μ and σ^2.

Problem 3.4 All of the distributions in this chapter are members of the *exponential family* and can be written in the form

$$Pr(x|\boldsymbol{\theta}) = a[\mathbf{x}] \exp[\mathbf{b}[\boldsymbol{\theta}]^T c[\mathbf{x}] - d[\boldsymbol{\theta}]],$$

where $a[\mathbf{x}]$ and $\mathbf{c}[\mathbf{x}]$ are functions of the data and $\mathbf{b}[\boldsymbol{\theta}]$ and $d[\boldsymbol{\theta}]$ are functions of the parameters. Find the functions $a[\mathbf{x}], \mathbf{b}[\boldsymbol{\theta}], \mathbf{c}[\mathbf{x}]$ and $d[\boldsymbol{\theta}]$ that allow the Beta distribution to be represented in the generalized form of the exponential family.

Problem 3.5 Use integration by parts to prove that if

$$\Gamma[z] = \int_0^\infty t^{z-1} e^{-t} dt,$$

then

$$\Gamma[z+1] = z\Gamma[z].$$

Problem 3.6 Consider a restricted family of univariate normal distributions where the variance is always 1, so that

$$Pr(x|\mu) = \frac{1}{\sqrt{2\pi}} \exp\left[-0.5(x-\mu)^2\right].$$

Show that a normal distribution over the parameter μ

$$Pr(\mu) = \text{Norm}_\mu[\mu_p, \sigma_p^2]$$

has a conjugate relationship to the restricted normal distribution.

Problem 3.7 For the univariate normal distribution, find the functions $a[\mathbf{x}], \mathbf{b}[\boldsymbol{\theta}], \mathbf{c}[\mathbf{x}]$ and $d[\boldsymbol{\theta}]$ that allow it to be represented in the generalized form of the exponential family (see problem 3.4).

Problem 3.8 Calculate an expression for the mode (position of the peak in μ, σ^2 space) of the normal scaled inverse gamma distribution in terms of the parameters $\alpha, \beta, \gamma, \delta$.

Problem 3.9 Show that the more general form of the conjugate relation in which we multiply I Bernoulli distributions by the conjugate beta prior is given by

$$\prod_{i=1}^{I} \text{Bern}_{x_i}[\lambda] \cdot \text{Beta}_\lambda[\alpha, \beta] = \kappa \cdot \text{Beta}_\lambda[\tilde{\alpha}, \tilde{\beta}],$$

where

$$\begin{aligned}
\kappa &= \frac{\Gamma[\alpha+\beta]\Gamma[\alpha+\sum x_i]\Gamma[\beta+\sum(1-x_i)]}{\Gamma[\alpha+\beta+I]\Gamma[\alpha]\Gamma[\beta]} \\
\tilde{\alpha} &= \alpha + \sum x_i \\
\tilde{\beta} &= \beta + \sum(1-x_i).
\end{aligned}$$

Problem 3.10 Prove the conjugate relation

$$\prod_{i=1}^{I} \text{Cat}_{\mathbf{x}_i}[\lambda_{1\ldots K}] \cdot \text{Dir}_{\lambda_{1\ldots K}}[\alpha_{1\ldots K}] = \kappa \cdot \text{Dir}_{\lambda_{1\ldots K}}[\tilde{\alpha}_{1\ldots K}],$$

where

$$\begin{aligned}
\tilde{\kappa} &= \frac{\Gamma[\sum_{j=1}^K \alpha_j]}{\Gamma[I + \sum_{j=1}^K \alpha_j]} \cdot \frac{\prod_{j=1}^K \Gamma[\alpha_j + N_j]}{\prod_{j=1}^K \Gamma[\alpha_j]} \\
\tilde{\alpha}_{1\ldots K} &= [\alpha_1 + N_1, \alpha_2 + N_2, \ldots, \alpha_K + N_K].
\end{aligned}$$

and N_k is the total number of times that the variable took the value k.

Problem 3.11 Show that the conjugate relation between the normal and normal inverse gamma is given by

$$\prod_{i=1}^{I} \text{Norm}_{x_i}[\mu, \sigma^2] \cdot \text{NormInvGam}_{\mu,\sigma^2}[\alpha, \beta, \gamma, \delta] = \kappa \cdot \text{NormInvGam}_{\mu,\sigma^2}[\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}, \tilde{\delta}],$$

where

$$\begin{aligned}
\kappa &= \frac{1}{(2\pi)^{I/2}} \frac{\sqrt{\gamma}\beta^{\alpha}}{\sqrt{\tilde{\gamma}}\tilde{\beta}^{\tilde{\alpha}}} \frac{\Gamma[\tilde{\alpha}]}{\Gamma[\alpha]} \\
\tilde{\alpha} &= \alpha + I/2 \\
\tilde{\beta} &= \frac{\sum_i x_i^2}{2} + \beta + \frac{\gamma\delta^2}{2} - \frac{(\gamma\delta + \sum_i x_i)^2}{2(\gamma + I)} \\
\tilde{\gamma} &= \gamma + I \\
\tilde{\delta} &= \frac{(\gamma\delta + \sum_i x_i)}{\gamma + I}.
\end{aligned}$$

Problem 3.12 Show that the conjugate relationship between the multivariate normal and the normal inverse Wishart is given by

$$\prod_{i=1}^{I} \text{Norm}_{\mathbf{x}_i}[\boldsymbol{\mu}, \boldsymbol{\Sigma}] \cdot \text{NorIWis}_{\boldsymbol{\mu},\boldsymbol{\Sigma}}[\alpha, \boldsymbol{\Psi}, \gamma, \boldsymbol{\delta}] = \kappa \cdot \text{NorIWis}\left[\tilde{\alpha}, \tilde{\boldsymbol{\Psi}}, \tilde{\gamma}, \tilde{\boldsymbol{\delta}}\right],$$

where

$$\begin{aligned}
\kappa &= \frac{1}{\pi^{ID/2}} \frac{\boldsymbol{\Psi}^{\alpha/2}}{\tilde{\boldsymbol{\Psi}}^{\tilde{\alpha}/2}} \frac{\Gamma_D[\tilde{\alpha}/2]}{\Gamma_D[\alpha/2]} \frac{\gamma^{D/2}}{\tilde{\gamma}^{D/2}} \\
\tilde{\alpha} &= \alpha + I \\
\tilde{\boldsymbol{\Psi}} &= \boldsymbol{\Psi} + \gamma\boldsymbol{\delta}\boldsymbol{\delta}^T + \sum_{i=1}^{I} \mathbf{x}_i\mathbf{x}_i^T - \frac{1}{(\gamma + I)} \left(\gamma\boldsymbol{\delta} + \sum_{i=1}^{I} \mathbf{x}_i\right) \left(\gamma\boldsymbol{\delta} + \sum_{i=1}^{I} \mathbf{x}_i\right)^T \\
\tilde{\gamma} &= \gamma + I \\
\tilde{\boldsymbol{\delta}} &= \frac{\gamma\boldsymbol{\delta} + \sum_{i=1}^{I} \mathbf{x}_i}{\gamma + I}.
\end{aligned}$$

You may need to use the relation $\text{Tr}\left[\mathbf{z}\mathbf{z}^T\mathbf{A}^{-1}\right] = \mathbf{z}^T\mathbf{A}^{-1}\mathbf{z}$.

Chapter 4

Fitting probability models

This chapter concerns fitting probability models to data $\{\mathbf{x}_i\}_{i=1}^I$. This process is referred to as *learning* because we learn about the parameters $\boldsymbol{\theta}$ of the model.[1] It also concerns calculating the probability of a new datum $\mathbf{x}^*$ under the resulting model. This is known as evaluating the *predictive distribution*. We consider three methods: *maximum likelihood*, *maximum a posteriori*, and the *Bayesian approach*.

4.1 Maximum likelihood

As the name suggests, the maximum likelihood (ML) method finds the set of parameters $\hat{\boldsymbol{\theta}}$ under which the data $\{\mathbf{x}_i\}_{i=1}^I$ are most likely. To calculate the likelihood function $Pr(\mathbf{x}_i|\boldsymbol{\theta})$ at a single data point $\mathbf{x}_i$, we simply evaluate the probability density function at $\mathbf{x}_i$. Assuming each data point was drawn independently from the distribution, the likelihood function $Pr(\mathbf{x}_{1\ldots I}|\boldsymbol{\theta})$ for a set of points is the product of the individual likelihoods. Hence, the ML estimate of the parameters is

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}}{\text{argmax}}\left[Pr(\mathbf{x}_{1\ldots I}|\boldsymbol{\theta})\right] \\ &= \underset{\boldsymbol{\theta}}{\text{argmax}}\left[\prod_{i=1}^{I} Pr(\mathbf{x}_i|\boldsymbol{\theta})\right], \end{aligned} \tag{4.1}$$

where $\text{argmax}_{\boldsymbol{\theta}}\, f[\boldsymbol{\theta}]$ returns the value of $\boldsymbol{\theta}$ that maximizes the argument $f[\boldsymbol{\theta}]$.

To evaluate the predictive distribution for a new data point $\mathbf{x}^*$ (compute the probability that $\mathbf{x}^*$ belongs to the fitted model), we simply evaluate the probability density function $Pr(\mathbf{x}^*|\hat{\boldsymbol{\theta}})$ using the ML fitted parameters $\hat{\boldsymbol{\theta}}$.

[1] Here we adopt the notation $\boldsymbol{\theta}$ to represent a generic set of parameters when we have not specified the particular probability model.

4.2 Maximum a posteriori

In maximum a posteriori (MAP) fitting, we introduce *prior* information about the parameters $\boldsymbol{\theta}$. From previous experience we may know something about the possible parameter values. For example, in a time-sequence the values of the parameters at time t tell us a lot about the possible values at time $t+1$, and this information would be encoded in the prior distribution.

As the name suggests, maximum a posteriori estimation maximizes the posterior probability $Pr(\boldsymbol{\theta}|\mathbf{x}_{1\ldots I})$ of the parameters

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\left[Pr(\boldsymbol{\theta}|\mathbf{x}_{1\ldots I})\right] \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\left[\frac{Pr(\mathbf{x}_{1\ldots I}|\boldsymbol{\theta})Pr(\boldsymbol{\theta})}{Pr(\mathbf{x}_{1\ldots I})}\right] \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\left[\frac{\prod_{i=1}^{I} Pr(\mathbf{x}_i|\boldsymbol{\theta})Pr(\boldsymbol{\theta})}{Pr(\mathbf{x}_{1\ldots I})}\right], \end{aligned} \tag{4.2}$$

where we have used Bayes' rule between the first two lines and subsequently assumed independence. In fact, we can discard the denominator as it is constant with respect to the parameters and so does not affect the position of the maximum, and we get

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\left[\prod_{i=1}^{I} Pr(\mathbf{x}_i|\boldsymbol{\theta})Pr(\boldsymbol{\theta})\right]. \tag{4.3}$$

Comparing this to the maximum likelihood criterion (equation 4.1) we see that it is identical except for the additional prior term; maximum likelihood is a special case of maximum a posteriori where the prior is uninformative.

The predictive density (probability of a new datum $\mathbf{x}^*$ under the fitted model) is again calculated by evaluating the pdf $Pr(\mathbf{x}^*|\hat{\boldsymbol{\theta}})$ using the new parameters.

4.3 The Bayesian approach

In the Bayesian approach we stop trying to estimate single fixed values (*point estimates*) of the parameters $\boldsymbol{\theta}$ and admit what is obvious; there may be many values of the parameters that are compatible with the data. We compute a probability distribution $Pr(\boldsymbol{\theta}|\mathbf{x}_{1\ldots I})$ over the parameters $\boldsymbol{\theta}$ based on data $\{\mathbf{x}_i\}_{i=1}^{I}$ using Bayes' rule so that

$$Pr(\boldsymbol{\theta}|\mathbf{x}_{1\ldots I}) = \frac{\prod_{i=1}^{I} Pr(\mathbf{x}_i|\boldsymbol{\theta})Pr(\boldsymbol{\theta})}{Pr(\mathbf{x}_{1\ldots I})}. \tag{4.4}$$

Evaluating the predictive distribution is more difficult for the Bayesian case since we have not estimated a single model but have instead found a probability distribution over possible models. Hence, we calculate

$$Pr(\mathbf{x}^*|\mathbf{x}_{1\ldots I}) = \int Pr(\mathbf{x}^*|\boldsymbol{\theta})Pr(\boldsymbol{\theta}|\mathbf{x}_{1\ldots I})\, d\boldsymbol{\theta}, \tag{4.5}$$

which can be interpreted as follows: the term $Pr(\mathbf{x}^*|\boldsymbol{\theta})$ is the prediction for a given value of $\boldsymbol{\theta}$. So, the integral can be thought of as a weighted sum of the predictions given by different parameters $\boldsymbol{\theta}$, where the weighting is determined by the posterior probability distribution $Pr(\boldsymbol{\theta}|\mathbf{x}_{1\ldots I})$ over the parameters (representing our confidence that different parameters are correct).

The predictive density calculations for the Bayesian, MAP and ML cases can be unified if we consider the ML and MAP estimates to be special probability distributions over the parameters where all of the density is at $\hat{\boldsymbol{\theta}}$. More formally, we can consider them as *delta functions* centered at $\hat{\boldsymbol{\theta}}$. A delta function $\delta[z]$ is a function that integrates to one, and that returns zero everywhere except at $z = 0$. We can now write

$$\begin{aligned} Pr(\mathbf{x}^*|\mathbf{x}_{1\ldots I}) &= \int Pr(\mathbf{x}^*|\boldsymbol{\theta})\delta[\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}]\, d\boldsymbol{\theta} \\ &= Pr(\mathbf{x}^*|\hat{\boldsymbol{\theta}}), \end{aligned} \tag{4.6}$$

which is exactly the calculation we originally prescribed: we simply evaluate the probability of the data under the model with the estimated parameters.

4.4 Worked example 1: univariate normal

To illustrate the above ideas, we will consider fitting a univariate normal model to scalar data $\{x_i\}_{i=1}^I$. Recall that the univariate normal model has pdf

$$Pr(x|\mu, \sigma^2) = \text{Norm}_x[\mu, \sigma^2] = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-0.5\frac{(x-\mu)^2}{\sigma^2}\right], \tag{4.7}$$

and has two parameters, the mean μ and the variance σ^2. Let us generate I independent data points $x_{1\ldots I}$ from a univariate normal with $\mu = 1$ and $\sigma^2 = 1$. Our goal is to re-estimate these parameters from the data.

4.4.1 Maximum likelihood estimation

The likelihood $Pr(x_{1\ldots I}|\mu, \sigma^2)$ of the parameters $\{\mu, \sigma^2\}$ for observed data $\{x_i\}_{i=1}^I$ is computed by evaluating the pdf for each data point separately and taking the product: Algorithm 4.1

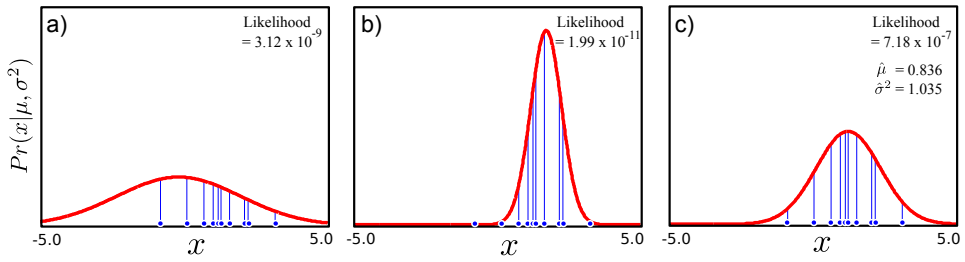


Figure 4.1 Maximum likelihood fitting. The likelihood of the parameters for a single datapoint is the height of the pdf evaluated at that point (blue vertical lines). The likelihood of a set of independently sampled data is the product of the individual likelihoods. a) The likelihood for this normal distribution is low because the large variance means the height of the pdf is low everywhere. b) The likelihood for this normal distribution is even lower as the left-most datum is very unlikely under the model. c) The maximum likelihood solution is the set of parameters for which the likelihood is maximized.

$$\begin{aligned} Pr(x_{1\ldots I}|\mu,\sigma^2) &= \prod_{i=1}^{I} Pr(x_i|\mu,\sigma^2) \\ &= \prod_{i=1}^{I} \text{Norm}_{x_i}[\mu,\sigma^2] \\ &= \frac{1}{(2\pi\sigma^2)^{I/2}} \exp\left[-0.5\sum_{i=1}^{I}\frac{(x_i-\mu)^2}{\sigma^2}\right]. \end{aligned} \tag{4.8}$$

Obviously, the likelihood for some sets of parameters $\{\mu, \sigma^2\}$ will be higher than others (figure 4.1) and it is possible to visualize this as a 2D function of the mean μ and variance σ^2 (figure 4.2). The maximum likelihood solution $\hat{\mu}, \hat{\sigma}$ will occur at the peak of this surface so that

$$\hat{\mu}, \hat{\sigma}^2 = \underset{\mu,\sigma^2}{\text{argmax}}\left[Pr(x_{1\ldots I}|\mu,\sigma^2)\right]. \tag{4.9}$$

In principle we can maximize this by taking the derivative of equation 4.8 with respect to μ and σ^2, equating the result to zero and solving. In practice, however, the resulting equations are messy. To simplify things, we work instead with the logarithm of this expression (the log likelihood, L). Since the logarithm is a monotonic function (figure 4.3), the position of the maximum in the transformed function remains the same. Algebraically, the logarithm turns the product of the likelihoods of the individual data points into a sum and so decouples the contribution of each. The ML parameters can now be calculated as

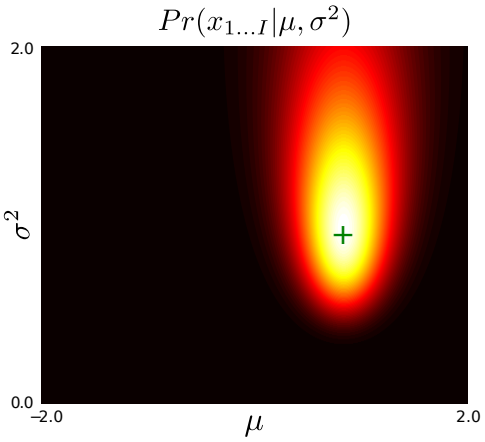


Figure 4.2 The likelihood function for a fixed set of observed data is a function of the mean μ and variance σ^2 parameters. The plot shows that there are many parameter settings which might plausibly be responsible for the ten data points from figure 4.1. A sensible choice for the "best" parameter setting is the maximum likelihood solution (green cross), which corresponds to the maximum of this function.

$$\begin{aligned}\hat{\mu}, \hat{\sigma}^2 &= \underset{\mu,\sigma^2}{\text{argmax}} \left[\sum_{i=1}^{I} \log\left[\text{Norm}_{x_i}[\mu, \sigma^2]\right]\right] \\ &= \underset{\mu,\sigma^2}{\text{argmax}} \left[-0.5I\log[2\pi] - 0.5I\log\sigma^2 - 0.5\sum_{i=1}^{I}\frac{(x_i-\mu)^2}{\sigma^2}\right].\end{aligned} \tag{4.10}$$

To maximize, we differentiate this *log likelihood* $\mathbf{L}$ with respect to μ and equate the result to zero

$$\begin{aligned}\frac{\partial L}{\partial \mu} &= \sum_{i=1}^{I}\frac{(x_i-\mu)}{\sigma^2} \\ &= \frac{\sum_{i=1}^{I} x_i}{\sigma^2} - \frac{I\mu}{\sigma^2} = 0\end{aligned} \tag{4.11}$$

and re-arranging, we see that

$$\hat{\mu} = \frac{\sum_{i=1}^{I} x_i}{I}. \tag{4.12}$$

By a similar process, the expression for the variance can be shown to be

Problem 4.1

$$\hat{\sigma}^2 = \sum_{i=1}^{I}\frac{(x_i-\hat{\mu})^2}{I}. \tag{4.13}$$

These expressions are hardly surprising, but the same idea can be used to estimate parameters in other distributions where the results are less familiar.

Figure 4.1 shows a set of data points and three possible fits to the data. The mean of the maximum likelihood fit is the mean of the data. The ML fit is neither too narrow (giving very low probabilities to the furthest data points from the mean) nor too wide (resulting in a flat distribution and giving low probability to all points).

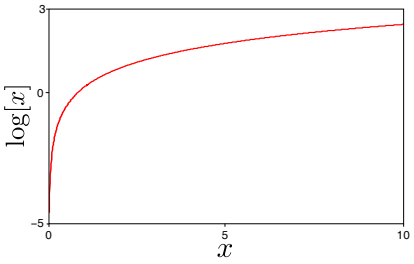


Figure 4.3 The logarithm is a monotonic transformation: if one point is higher than another then it will also be higher after transformation by the logarithmic function. It follows that if we transform the surface in figure 4.2 through the logarithmic function, the maximum will remain in the same position.

Least squares fitting

As an aside, we note that many texts discuss fitting in terms of *least squares*. Consider fitting just the mean parameter μ of the normal distribution using maximum likelihood. Manipulating the cost function so that

$$\begin{aligned}\hat{\mu} &= \underset{\mu}{\text{argmax}}\left[-0.5I\log[2\pi] - 0.5I\log\sigma^2 - 0.5\sum_{i=1}^{I}\frac{(x_i-\mu)^2}{\sigma^2}\right] \\ &= \underset{\mu}{\text{argmax}}\left[-\sum_{i=1}^{I}(x_i-\mu)^2\right] \\ &= \underset{\mu}{\text{argmin}}\left[\sum_{i=1}^{I}(x_i-\mu)^2\right], \end{aligned} \tag{4.14}$$

leads to a formulation where we minimize the sum of squares. In other words, least squares fitting is equivalent to fitting the mean parameter of a normal distribution using the maximum likelihood method.

4.4.2 Maximum a posteriori estimation

Algorithm 4.2

Returning to the main thread, we will now demonstrate maximum a posteriori fitting of the parameters of the normal distribution. The cost function becomes

$$\begin{aligned}\hat{\mu},\hat{\sigma}^2 &= \underset{\mu,\sigma^2}{\text{argmax}}\left[\prod_{i=1}^{I} Pr(x_i|\mu,\sigma^2)Pr(\mu,\sigma^2)\right] \\ &= \underset{\mu,\sigma^2}{\text{argmax}}\left[\prod_{i=1}^{I}\text{Norm}_{x_i}[\mu,\sigma^2]\text{NormInvGam}_{\mu,\sigma^2}[\alpha,\beta,\gamma,\delta]\right], \end{aligned} \tag{4.15}$$

where we have chosen normal inverse gamma prior with parameters $\alpha, \beta, \gamma, \delta$ (figure 4.4) as this is conjugate to the normal distribution. The expression for the prior is

$$Pr(\mu,\sigma^2) = \frac{\sqrt{\gamma}}{\sigma\sqrt{2\pi}}\frac{\beta^\alpha}{\Gamma(\alpha)}\left(\frac{1}{\sigma^2}\right)^{\alpha+1}\exp\left[-\frac{2\beta+\gamma(\delta-\mu)^2}{2\sigma^2}\right]. \tag{4.16}$$

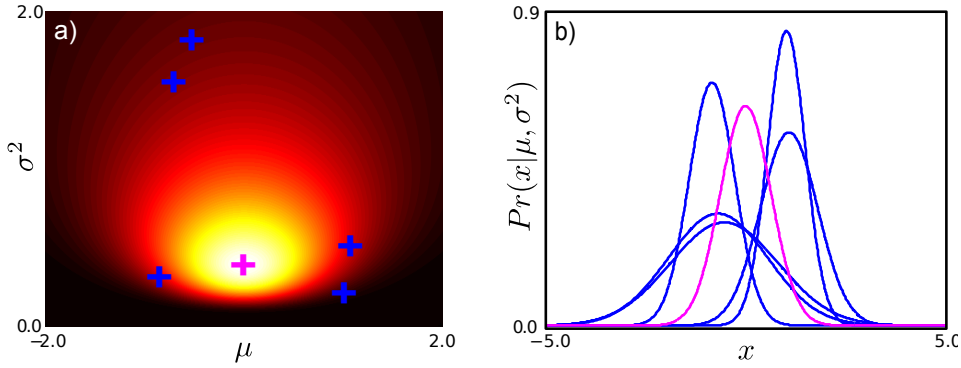


Figure 4.4 Prior over normal parameters. a) A normal inverse gamma with $\alpha, \beta, \gamma = 1$ and $\delta = 0$ gives a broad prior distribution over univariate normal parameters. The magenta cross indicates the peak of this prior distribution. The blue crosses are five samples randomly drawn from the distribution. b) The peak and the samples can be visualized by plotting the normal distributions that they represent.

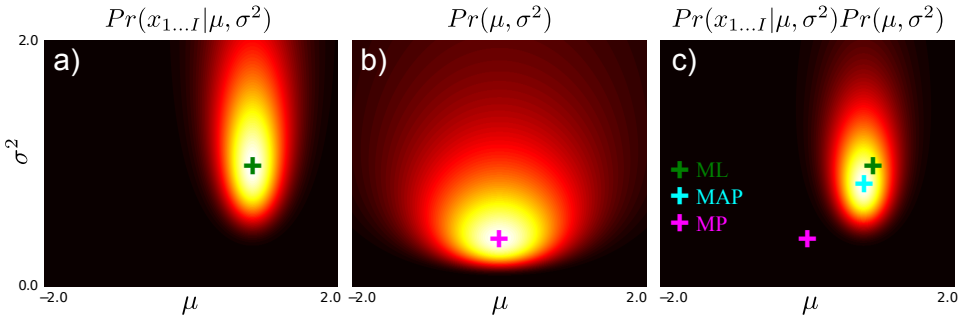


Figure 4.5 MAP inference for normal parameters. a) The likelihood function is multiplied by b) the prior probability to give a new function c) that is proportional to the posterior distribution. The maximum a posteriori (MAP) solution (cyan cross) is found at the peak of the posterior distribution. It lies between the maximum likelihood (ML) solution (green cross) and the maximum of the prior distribution (MP, magenta cross).

The posterior distribution is proportional to the product of the likelihood and the prior (figure 4.5), and has the highest density in regions that both agree with the data *and* were a priori plausible.

Like the maximum likelihood case, it is easier to maximize the logarithm of equation 4.15:

$$\hat{\mu}, \hat{\sigma}^2 = \underset{\mu,\sigma^2}{\text{argmax}} \left[\sum_{i=1}^{I} \log[\text{Norm}_{x_i}\left[\mu, \sigma^2\right]] + \log\left[\text{NormInvGam}_{\mu,\sigma^2}[\alpha, \beta, \gamma, \delta]\right] \right]. \tag{4.17}$$

Problem 4.2

To find the MAP parameters, we substitute in the expressions, differentiate with respect to μ and σ, equate to zero, and rearrange to give

$$\hat{\mu} = \frac{\sum_{i=1}^{I} x_i + \gamma\delta}{I + \gamma} \qquad \text{and} \qquad \hat{\sigma^2} = \frac{\sum_{i=1}^{I}(x_i - \hat{\mu})^2 + 2\beta + \gamma(\delta - \hat{\mu})^2}{I + 3 + 2\alpha}. \tag{4.18}$$

The formula for the mean can be more easily understood if we write it as

$$\hat{\mu} = \frac{I\overline{x} + \gamma\delta}{I + \gamma}. \tag{4.19}$$

This is a weighted sum of two terms. The first term is the data mean $\overline{x}$ and is weighted by the number of training examples I. The second term is δ, the value of μ favored by the prior, and is weighted by γ.

This gives some insight into the behavior of the MAP estimate (figure 4.6). With a large amount of data, the first term dominates, and the MAP estimate $\hat{\mu}$ is very close to the data mean (and the ML estimate). With intermediate amounts of data, $\hat{\mu}$ is a weighted sum of the prediction from the data and the prediction from the prior. With no data at all, the estimate is completely governed by the prior. The hyperparameter (parameter of the prior) γ controls the concentration of the prior with respect to μ and determines the extent of its influence. Similar conclusions can be drawn about the MAP estimate of the variance.

Where there is a single data point (figure 4.6e-f), the data tells us nothing about the variance and the maximum likelihood estimate $\hat{\sigma}^2$ is actually zero; the best fit is an infinitely thin and infinitely tall normal distribution centered on the one data point. This is unrealistic, not least because it accords the datum an infinite likelihood. However, MAP estimation is still valid as the prior ensures that sensible parameter values are chosen.

4.4.3 The Bayesian approach

Algorithm 4.3

In the Bayesian approach, we calculate a posterior distribution $Pr(\mu, \sigma^2|x_{1\ldots I})$ over possible parameter values using Bayes' rule,

$$\begin{aligned} Pr(\mu, \sigma^2|x_{1\ldots I}) &= \frac{\prod_{i=1}^{I} Pr(x_i|\mu, \sigma^2) Pr(\mu, \sigma^2)}{Pr(x_{1\ldots I})} \\ &= \frac{\prod_{i=1}^{I} \text{Norm}_{x_i}[\mu, \sigma^2]\text{NormInvGam}_{\mu,\sigma^2}[\alpha, \beta, \gamma, \delta]}{Pr(x_{1\ldots I})} \\ &= \frac{\kappa\text{NormInvGam}_{\mu,\sigma^2}[\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}, \tilde{\delta}]}{Pr(x_{1\ldots I})}, \end{aligned} \tag{4.20}$$

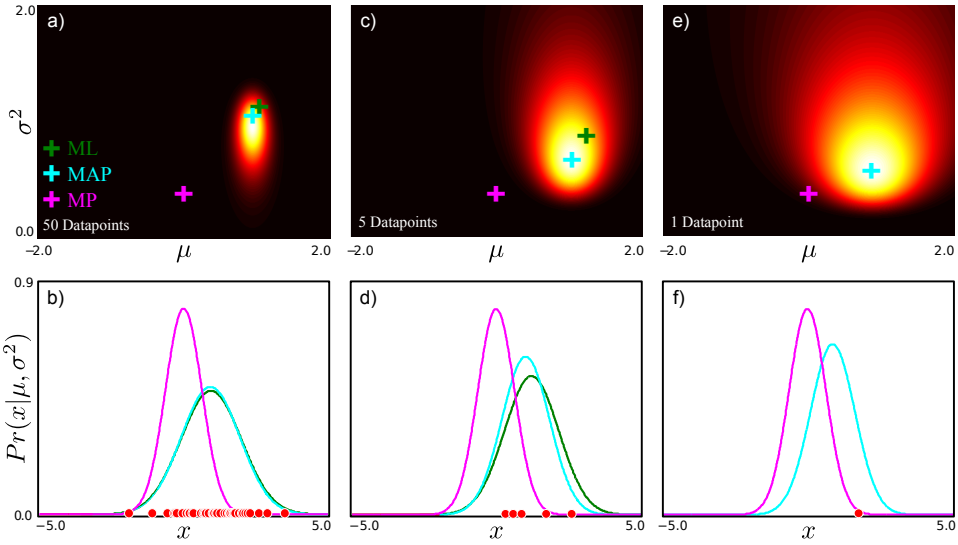


Figure 4.6 Maximum a posteriori estimation. a) MAP solution (cyan cross) lies between ML (green cross) and the peak of the prior (purple cross). b) Normal distributions corresponding to MAP solution, ML solution and peak of prior. c-d) With fewer data points, the prior has a greater effect on the final solution. e-f) With only one data point, the maximum likelihood solution cannot be computed (you cannot calculate the variance of a single point). However, the MAP solution can still be calculated.

where we have used the conjugate relationship between likelihood and prior (section 3.9) and κ is the associated constant. The product of the normal likelihood and normal inverse gamma prior creates a posterior over μ and σ^2, which is a new normal inverse gamma distribution and can be shown to have parameters

$$\begin{aligned}\tilde{\alpha} &= \alpha + I/2, \qquad \tilde{\gamma} = \gamma + I \qquad \tilde{\delta} = \frac{(\gamma\delta + \sum_i x_i)}{\gamma + I} \\ \tilde{\beta} &= \frac{\sum_i x_i^2}{2} + \beta + \frac{\gamma\delta^2}{2} - \frac{(\gamma\delta + \sum_i x_i)^2}{2(\gamma + I)}.\end{aligned} \tag{4.21}$$

Note that the posterior (left-hand side of equation 4.20) must be a valid probability distribution and sum to one, so the constant κ from the conjugate product and the denominator from the right-hand side must exactly cancel to give

$$Pr(\mu, \sigma^2|x_{1\ldots I}) \quad = \quad \text{NormInvGam}_{\mu,\sigma^2}[\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}, \tilde{\delta}]. \tag{4.22}$$

Now we see the major advantage of using a conjugate prior: we are guaranteed a closed form expression for the posterior distribution over the parameters.

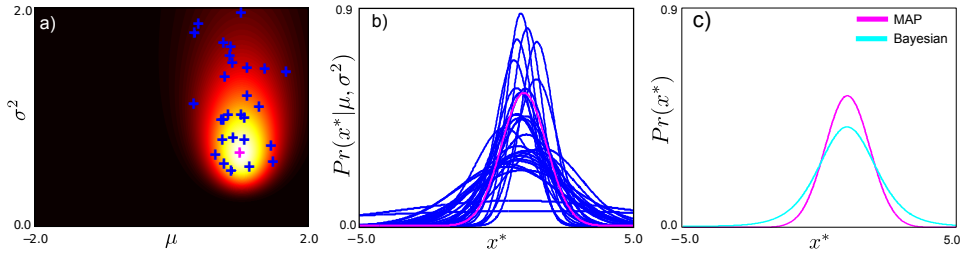


Figure 4.7 Bayesian predictions. a) Posterior probability distribution over parameters. b) Samples from posterior probability distribution correspond to normal distributions. c) The predictive distribution for the Bayesian case is the average of an infinite set of samples. Alternately, we can think of choosing the parameters from a uniform distribution and computing a weighted average where the weights correspond to the posterior distribution.

This posterior distribution represents the relative plausibility of various parameter settings μ and σ^2 having created the data. At the peak of the distribution is the MAP estimate, but there are many other plausible configurations (figure 4.6).

When data are plentiful (figure 4.6a), the parameters are well specified, and the probability distribution is concentrated. In this case, placing all of the probability mass at the MAP estimate is a good approximation to the posterior. However, when data are scarce (figure 4.6c), many possible parameters might have explained the data and the posterior is broad. In this case approximation with a point mass is inadequate.

Predictive density

For the maximum likelihood and MAP estimates, we evaluate the predictive density (probability that a new data point x^* belongs to the same model) by simply evaluating the normal pdf with the estimated parameters. For the Bayesian case, we compute a weighted average of the predictions for each possible parameter set, where the weighting is given by the posterior distribution over parameters (figures 4.6a-c and 4.7),

$$\begin{aligned} Pr(x^*|x_{1\ldots I}) &= \iint Pr(x^*|\mu,\sigma^2)Pr(\mu,\sigma^2|x_{1\ldots I})\, d\mu d\sigma \\ &= \iint \text{Norm}_{x^*}[\mu,\sigma^2]\text{NormInvGam}_{\mu,\sigma^2}[\tilde{\alpha},\tilde{\beta},\tilde{\gamma},\tilde{\delta}]\, d\mu d\sigma \\ &= \iint \kappa(x^*,\tilde{\alpha},\tilde{\beta},\tilde{\gamma},\tilde{\delta})\text{NormInvGam}_{\mu,\sigma^2}[\breve{\alpha},\breve{\beta},\breve{\gamma},\breve{\delta}]\, d\mu d\sigma. \end{aligned} \tag{4.23}$$

Here we have used the conjugate relation for a second time. The integral contains a constant with respect to μ and σ^2 multiplied by a probability distribution. Taking the constant outside the integral, we get

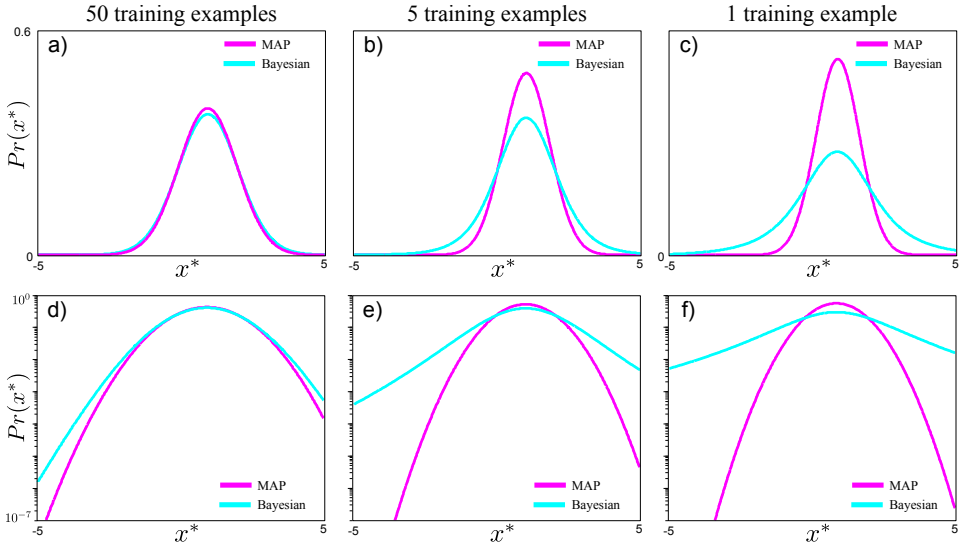


Figure 4.8 a-c) Predictive densities for MAP and Bayesian approaches with 50, 5, and 1 training examples, respectively. As the training data decreases, the Bayesian prediction becomes less certain but the MAP prediction is erroneously overconfident. d-f) This effect is even more clear on a log scale.

$$\begin{aligned} Pr(x^*|x_{1\ldots I}) &= \kappa(x^*, \tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}, \tilde{\delta}) \iint \text{NormInvGam}_{\mu,\sigma^2}[\breve{\alpha}, \breve{\beta}, \breve{\gamma}, \breve{\delta}]\, d\mu d\sigma \\ &= \kappa(x^*, \tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}, \tilde{\delta}), \end{aligned} \tag{4.24}$$

which follows because the integral of a pdf is one. It can be shown that the constant is given by

$$\kappa(x^*, \tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}, \tilde{\delta}) = \frac{1}{\sqrt{2\pi}} \frac{\sqrt{\tilde{\gamma}}\tilde{\beta}^{\tilde{\alpha}}}{\sqrt{\breve{\gamma}}\breve{\beta}^{\breve{\alpha}}} \frac{\Gamma[\breve{\alpha}]}{\Gamma[\tilde{\alpha}]}, \tag{4.25}$$

where

$$\begin{aligned} \breve{\alpha} &= \tilde{\alpha} + 1/2, \qquad & \breve{\gamma} &= \tilde{\gamma} + 1 \\ \breve{\beta} &= \frac{x^{*2}}{2} + \tilde{\beta} + \frac{\tilde{\gamma}\tilde{\delta}^2}{2} - \frac{(\tilde{\gamma}\tilde{\delta} + x^*)^2}{2(\tilde{\gamma}+1)}. \end{aligned} \tag{4.26}$$

Here, we see the second advantage of using the conjugate prior; it means that the integral can be computed and so we get a nice closed form expression for the predictive density.

Figure 4.8 shows the predictive distribution for the Bayesian and MAP cases, for varying amounts of training data. With plenty of training data, they are quite

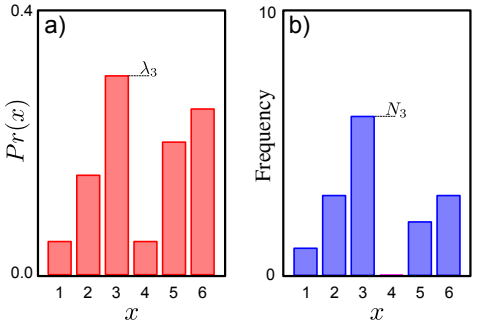


Figure 4.9 a) Categorical probability distribution over six discrete values with parameters $\{\lambda_k\}_{k=1}^{6}$ where $\sum_{k=1}^{6}\lambda_k = 1$. This could be the relative probability of a biased die landing on its six sides. b) Fifteen observations $\{x_i\}_{i=1}^{I}$ randomly sampled from this distribution. We denote the number of times category k was observed by N_k so that here the total observations $\sum_{k=1}^{6} N_k = 15$.

similar but as the amount of data decreases, the Bayesian predictive distribution has a significantly longer tail. This is typical of Bayesian solutions: they are more moderate (less certain) in their predictions. In the MAP case, erroneously committing to a single estimate of μ and σ^2 causes overconfidence in our future predictions.

4.5 Worked example 2: categorical distribution

As a second example, we consider discrete data $\{x_i\}_{i=1}^{I}$ where $x_i \in \{1, 2, \ldots, 6\}$ (figure 4.9). This could represent observed rolls of a die with unknown bias. We will describe the data using a categorical distribution (normalized histogram) where

$$Pr(x = k|\lambda_{1\ldots K}) = \lambda_k. \tag{4.27}$$

For the ML and MAP techniques, we estimate the six parameters $\{\lambda_k\}_{k=1}^{6}$. For the Bayesian approach, we compute a probability distribution over the parameters.

4.5.1 Maximum Likelihood

Algorithm 4.4

To find the maximum likelihood solution, we maximize the product of the likelihoods for each individual data point with respect to the parameters $\lambda_{1\ldots6}$.

$$\begin{aligned}
\hat{\lambda}_{1\ldots6} &= \underset{\lambda_{1\ldots6}}{\text{argmax}}\left[\prod_{i=1}^{I} Pr(x_i|\lambda_{1\ldots6})\right] \qquad \text{s.t.} \sum_k \lambda_k = 1 \\
&= \underset{\lambda_{1\ldots6}}{\text{argmax}}\left[\prod_{i=1}^{I} \text{Cat}_{x_i}[\lambda_{1\ldots6}]\right] \qquad \text{s.t.} \sum_k \lambda_k = 1 \\
&= \underset{\lambda_{1\ldots6}}{\text{argmax}}\left[\prod_{k=1}^{6} \lambda_k^{N_k}\right] \qquad \text{s.t.} \sum_k \lambda_k = 1,
\end{aligned} \tag{4.28}$$

where N_k is the total number of times we observed bin k in the training data. As before, it is easier to maximize the log probability, and we use the criterion

$$L = \sum_{k=1}^{6} N_k \log[\lambda_k] + \nu\left(\sum_{k=1}^{6}\lambda_k - 1\right), \tag{4.29}$$

where the second term uses the Lagrange multiplier ν to enforce the constraint on the parameters $\sum_{k=1}^{6}\lambda_k = 1$. We differentiate L with respect to λ_k and ν, set the derivatives equal to zero, and solve for λ_k to obtain

$$\hat{\lambda}_k = \frac{N_k}{\sum_{m=1}^{6} N_m}. \tag{4.30}$$

In other words, λ_k is the proportion of times that we observed bin k.

4.5.2 Maximum a posteriori

To find the maximum a posteriori solution we need to define a prior. We choose the Dirichlet distribution as it is conjugate to the categorical likelihood. This prior over the six categorical parameters is hard to visualize but samples can be drawn and examined (figure 4.10a-e). The MAP solution is given by

Algorithm 4.5

$$\begin{aligned}
\hat{\lambda}_{1\ldots6} &= \underset{\lambda_{1\ldots6}}{\text{argmax}}\left[\prod_{i=1}^{I} Pr(x_i|\lambda_{1\ldots6})Pr(\lambda_{1\ldots6})\right] \\
&= \underset{\lambda_{1\ldots6}}{\text{argmax}}\left[\prod_{i=1}^{I} \text{Cat}_{x_i}[\lambda_{1\ldots6}]\text{Dir}_{\lambda_{1\ldots6}}[\alpha_{1\ldots6}]\right] \\
&= \underset{\lambda_{1\ldots6}}{\text{argmax}}\left[\prod_{k=1}^{6}\lambda_k^{N_k}\prod_{k=1}^{6}\lambda_k^{\alpha_k-1}\right] \\
&= \underset{\lambda_{1\ldots6}}{\text{argmax}}\left[\prod_{k=1}^{6}\lambda_k^{N_k+\alpha_k-1}\right].
\end{aligned} \tag{4.31}$$

which is again subject to the constraint that $\sum_{k=1}^{6}\lambda_k = 1$. As in the maximum likelihood case, this constraint is enforced using a Lagrange multiplier. The MAP estimate of the parameters can be shown to be

Problem 4.4

$$\hat{\lambda}_k = \frac{N_k + \alpha_k - 1}{\sum_{m=1}^{6}(N_m + \alpha_m - 1)}, \tag{4.32}$$

where N_k is the number of times that observation k occurred in the training data. Note that if all the values α_k are set to one, the prior is uniform and this expression reverts to the maximum likelihood solution (equation 4.30).

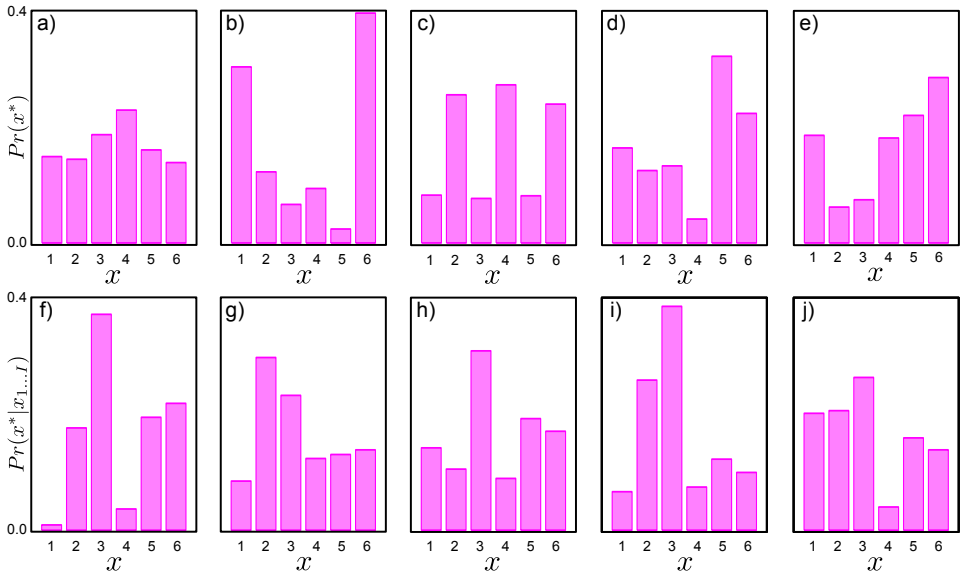


Figure 4.10 a-e) Five samples drawn from Dirichlet prior with hyperparameters $\alpha_{1...6} = 1$. This defines a uniform prior, so each sample looks like a random unstructured probability distribution. f-j) Five samples from Dirichlet posterior. The distribution favors histograms where bin three is larger and bin four is small as suggested by the data.

4.5.3 Bayesian Approach

Algorithm 4.6

In the Bayesian approach we calculate a posterior over the parameters

$$\begin{aligned} Pr(\lambda_1 \ldots \lambda_6 | x_{1...I}) &= \frac{\prod_{i=1}^{I} Pr(x_i|\lambda_{1...6}) Pr(\lambda_{1...6})}{Pr(x_{1...I})} \\ &= \frac{\prod_{i=1}^{I} \text{Cat}_{x_i}[\lambda_{1...6}] \text{Dir}_{\lambda_{1...6}}[\alpha_{1...6}]}{Pr(x_{1...I})} \\ &= \frac{\kappa(\alpha_{1...6}, x_{1...I}) \text{Dir}_{\lambda_{1...6}}[\tilde{\alpha}_{1...6}]}{Pr(x_{1...I})} \\ &= \text{Dir}_{\lambda_{1...6}}[\tilde{\alpha}_{1...6}], \end{aligned} \tag{4.33}$$

where $\tilde{\alpha}_k = N_k + \alpha_k$. We have again exploited the conjugate relationship to yield a posterior distribution with the same form as the prior. The constant κ must again cancel with the denominator to ensure a valid probability distribution on the left hand side. Samples from this distribution are shown in figure 4.10f-j.

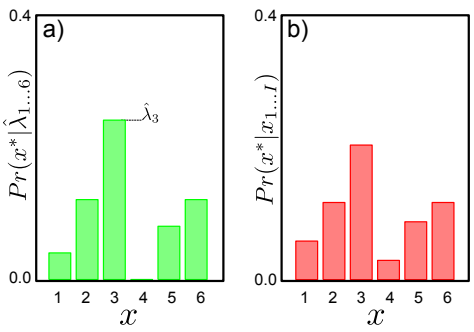


Figure 4.11 Predictive distributions with $\alpha_{1...6} = 1$ for a) maximum likelihood / maximum a posteriori approaches and b) Bayesian approach. The ML/MAP approaches predict the same distribution that exactly follows the data frequencies. The Bayesian approach predicts a more moderate distribution and allots some probability to the case $x = 4$ despite having seen no training examples in this category.

Predictive Density

For the ML and MAP estimates we evaluate the predictive density (probability that a new data point x^* belongs to the same model) by simply evaluating the categorical pdf with the estimated parameters. With the uniform prior ($\alpha_{1...6} = 1$) the MAP and ML predictions are identical (figure 4.11a) and both are exactly proportional to the frequencies of the observed data.

For the Bayesian case, we compute a weighted average of the predictions for each possible parameter set, where the weighting is given by the posterior distribution over parameters so that

$$\begin{aligned} Pr(x^*|x_{1...I}) &= \int Pr(x^*|\lambda_{1...6})Pr(\lambda_{1...6}|x_{1...I})\, d\lambda_{1...6} \\ &= \int \text{Cat}_{x^*}[\lambda_{1...6}]\text{Dir}_{\lambda_{1...6}}[\tilde{\alpha}_{1...6}]\, d\lambda_{1...6} \\ &= \int \kappa(x^*, \tilde{\alpha}_{1...6})\text{Dir}_{\lambda_{1...6}}[\breve{\alpha}_{1...6}]\, d\lambda_{1...6} \\ &= \kappa(x^*, \tilde{\alpha}_{1...6}). \end{aligned} \tag{4.34}$$

Here, we have again exploited the conjugate relationship to yield a constant multiplied by a probability distribution and the integral is simply the constant as the integral of the pdf is one. For this case, it can be shown that

$$Pr(x^* = k|x_{1...I}) = \kappa(x^*, \tilde{\alpha}_{1...6}) = \frac{N_k + \alpha_k}{\sum_{j=1}^{6}(N_j + \alpha_j)}. \tag{4.35}$$

This is illustrated in figure 4.11b. It is notable that once more the Bayesian predictive density is less confident than the ML/MAP solutions. In particular, it does not allot zero probability to observing $x^* = 4$ despite the fact that this value was never observed in the training data. This is sensible; just because we have not drawn a 4 in 15 observations does not imply that it is inconceivable that we will ever see one. We may have just been unlucky. The Bayesian approach takes this into account and allots this category a small amount of probability.

Summary

We presented three ways to fit a probability distribution to data and to predict the probability of new points. Of the three methods discussed, the Bayesian approach is the most desirable. Here it is not necessary to find a point estimate of the (uncertain) parameters, and so errors are not introduced because this point estimate is inaccurate.

However, the Bayesian approach is only tractable when we have a conjugate prior, which makes it easy to calculate the posterior distribution over the parameters $Pr(\boldsymbol{\theta}|\mathbf{x}_{1\ldots I})$ and also to evaluate the integral in the predictive density. When this is not the case, we will usually have to rely on maximum a posteriori estimates. Maximum likelihood estimates can be thought of as a special case of maximum a posteriori estimates in which the prior is uninformative.

Notes

For more information about the Bayesian approach to fitting distributions consult chapter 3 of Gelman *et al.* (2004). More information about Bayesian model selection (problem 4.6), including an impassioned argument for its superiority as a method of hypothesis testing can be found in Mackay (2003).

Problems

Problem 4.1 Show that the maximum likelihood solution for the variance σ^2 of the normal distribution is given by

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

Problem 4.2 Show that the MAP solution for the mean μ and variance σ^2 of the normal distribution are given by

$$\hat{\mu} = \frac{\sum_{i=1}^{I} x_i + \gamma\delta}{I+\gamma} \qquad \text{and} \qquad \hat{\sigma^2} = \frac{\sum_{i=1}^{I}(x_i - \hat{\mu})^2 + 2\beta + \gamma(\delta - \hat{\mu})^2}{I + 3 + 2\alpha},$$

when we use the conjugate normal-scaled inverse gamma prior

$$Pr(\mu, \sigma^2) = \frac{\sqrt{\gamma}}{\sigma\sqrt{2\pi}} \frac{\beta^\alpha}{\Gamma[\alpha]} \left(\frac{1}{\sigma^2}\right)^{\alpha+1} \exp\left[-\frac{2\beta + \gamma(\delta-\mu)^2}{2\sigma^2}\right].$$

Problem 4.3 Taking equation 4.29 as a starting point, show that the maximum likelihood parameters for the categorical distribution are given by

$$\hat{\lambda}_k = \frac{N_k}{\sum_{m=1}^{6} N_m},$$

where N_k is the number of times that category K was observed in the training data.

Problem 4.4 Show that the MAP estimate for the parameters $\{\lambda\}_{k=1}^{K}$ of the categorical distribution is given by

$$\hat{\lambda}_k = \frac{N_k + \alpha_k - 1}{\sum_{m=1}^{6}(N_m + \alpha_m - 1)},$$

under the assumption of a Dirichlet prior with hyperparameters $\{\alpha_k\}_{k=1}^{K}$. The terms N_k again indicate the number of times that category k was observed in the training data.

Problem 4.5 The denominator of Bayes' rule

$$Pr(x_{1\ldots I}) = \int \prod_{i=1}^{I} Pr(x_i|\theta) Pr(\theta)\, d\theta$$

is known as the *evidence*. It is a measure of how well the distribution fits *regardless* of the particular values of the parameters. Find an expression for the evidence term for (i) the normal distribution and (ii) the categorical distribution assuming conjugate priors in each case.

Problem 4.6 The evidence term can be used to compare models. Consider two sets of data $\mathcal{S}_1 = \{0.1, -0.5, 0.2, 0.7\}$ and $\mathcal{S}_2 = \{1.1, 2.0, 1.4, 2.3\}$. Let us pose the question of whether these two data sets came from the same normal distribution or from two different normal distributions.

Let model M_1 denote the case where all of the data comes from the one normal distribution. The evidence for this model is

$$Pr(\mathcal{S}_1 \cup \mathcal{S}_2|M_1) = \int \prod_{i \in \mathcal{S}_1 \cup \mathcal{S}_2} Pr(x_i|\boldsymbol{\theta}) Pr(\boldsymbol{\theta}) \, d\boldsymbol{\theta},$$

where $\boldsymbol{\theta} = \{\mu, \sigma^2\}$ contains the parameters of this normal distribution. Similarly, we will let M_2 denote the case where the two sets of data belong to different normal distributions

$$Pr(\mathcal{S}_1 \cup \mathcal{S}_2|M_2) = \int \prod_{i \in \mathcal{S}_1} Pr(x_i|\boldsymbol{\theta}_1) Pr(\boldsymbol{\theta}_1) \, d\boldsymbol{\theta}_1 \int \prod_{i \in \mathcal{S}_2} Pr(x_i|\boldsymbol{\theta}_2) Pr(\boldsymbol{\theta}_2) \, d\boldsymbol{\theta}_2,$$

where $\boldsymbol{\theta}_1 = \{\mu_1, \sigma_1^2\}$ and $\boldsymbol{\theta}_2 = \{\mu_2, \sigma_2^2\}$.

Now it is possible to compare the probability of the data under each of these two models using Bayes' rule

$$Pr(M_1|\mathcal{S}_1 \cup \mathcal{S}_2) = \frac{Pr(\mathcal{S}_1 \cup \mathcal{S}_2|M_1) Pr(M_1)}{\sum_{n=1}^{2} Pr(\mathcal{S}_1 \cup \mathcal{S}_2|M_n) Pr(M_n)}$$

Use this expression to compute the posterior probability that the two datasets came from the same underlying normal distribution. You may assume normal-scaled inverse gamma priors over $\boldsymbol{\theta}$, $\boldsymbol{\theta}_1$, and $\boldsymbol{\theta}_2$ with parameters $\alpha = 1, \beta = 1, \gamma = 1, \delta = 0$.

Note that this is (roughly) a Bayesian version of the two-sample t-test, but it is much neater - we get a posterior probability distribution over the two hypotheses rather than the potentially misleading p value of the t-test. The process of comparing evidence terms in this way is known as *Bayesian model selection* or *the evidence framework*. It is rather clever in that two normal distributions fitted with maximum likelihood will *always* explain the data better than one; the additional parameters simply make the model more flexible. However because we have marginalized these parameters away here, it is valid to compare these models in the Bayesian case.

Problem 4.7 In the Bernoulli distribution, the likelihood $Pr(x_{1\ldots I}|\lambda)$ of the data $\{x_i\}_{i=1}^I$ where $x_i \in \{0, 1\}$ given the parameter λ is

$$Pr(x_{1\ldots I}|\lambda) = \prod_{i=1}^{I} \lambda^{x_i} (1-\lambda)^{1-x_i}.$$

Find an expression for the maximum likelihood estimate of the parameter λ.

Problem 4.8 Find an expression for the MAP estimate of the Bernoulli parameter λ (see problem 4.7) assuming a beta distributed prior

$$Pr(\lambda) = \text{Beta}_\lambda[\alpha, \beta].$$

Problem 4.9 Now consider the Bayesian approach to fitting Bernoulli data, using a beta distributed prior. Find expressions for (i) the posterior probability distribution over the Bernoulli parameters given observed data $\{x_i\}_{i=1}^I$ and (ii) the predictive distribution for new data $\mathbf{x}^*$.

Problem 4.10 Staying with the Bernoulli distribution, consider observing data $0, 0, 0, 0$ from four trials. Assuming a uniform beta prior ($\alpha = 1, \beta = 1$), compute the predictive distribution using the (i) maximum likelihood, (ii) maximum a posteriori and (iii) Bayesian approaches. Comment on the results.

Chapter 5

The normal distribution

The most common representation for uncertainty in machine vision is the multivariate normal distribution. We devote this chapter to exploring its main properties, which will be used extensively throughout the rest of the book.

Recall from chapter 3 that the multivariate normal distribution has two parameters: the mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$. The mean $\boldsymbol{\mu}$ is a $D\times 1$ vector that describes the position of the distribution. The covariance $\boldsymbol{\Sigma}$ is a symmetric $D\times D$ positive definite matrix (implying that $\mathbf{z}^T\boldsymbol{\Sigma}\mathbf{z}$ is positive for any real vector $\mathbf{z}$) and describes the shape of the distribution. The probability density function is

$$Pr(\mathbf{x}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-0.5(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right], \tag{5.1}$$

or for short

$$Pr(\mathbf{x}) = \text{Norm}_{\mathbf{x}}\left[\boldsymbol{\mu}, \boldsymbol{\Sigma}\right]. \tag{5.2}$$

5.1 Types of covariance matrix

Covariance matrices in multivariate normals take three forms, termed *spherical*, *diagonal*, and *full* covariances. For the two dimensional (bivariate) case, these are

$$\boldsymbol{\Sigma}_{spher} = \begin{bmatrix}\sigma^2 & 0\\ 0 & \sigma^2\end{bmatrix} \quad \boldsymbol{\Sigma}_{diag} = \begin{bmatrix}\sigma_1^2 & 0\\ 0 & \sigma_2^2\end{bmatrix} \quad \boldsymbol{\Sigma}_{full} = \begin{bmatrix}\sigma_{11}^2 & \sigma_{12}^2\\ \sigma_{21}^2 & \sigma_{22}^2\end{bmatrix}. \tag{5.3}$$

The spherical covariance matrix is a positive multiple of the identity matrix and so has the same value on all of the diagonal elements and zeros elsewhere. In the diagonal covariance matrix, each value on the diagonal has a different positive value. The full covariance matrix can have non-zero elements everywhere although the matrix is still constrained to be symmetric and positive definite so for the 2D example, $\sigma_{12}^2 = \sigma_{21}^2$.

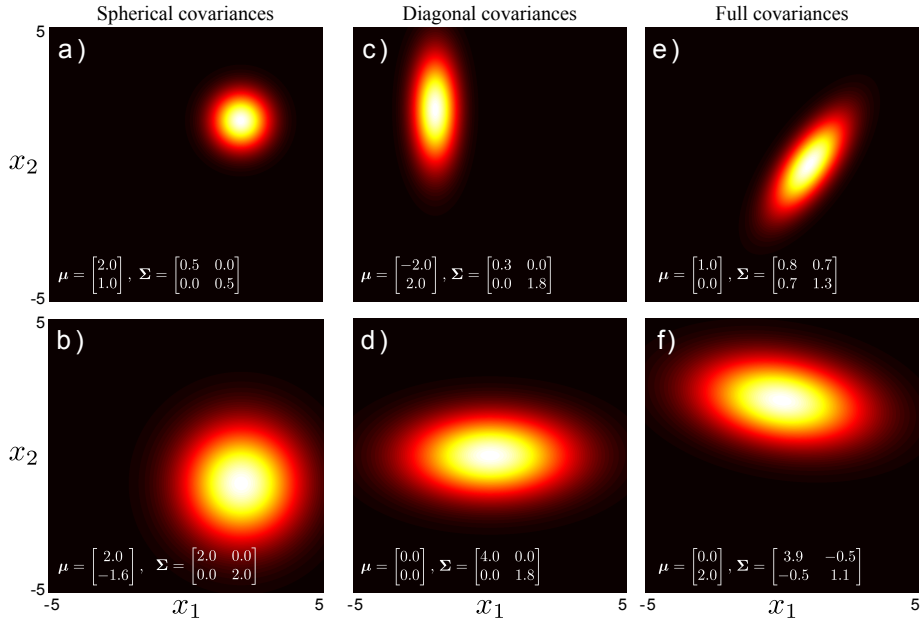


Figure 5.1 Covariance matrices take three forms. a-b) Spherical covariance matrices are multiples of the identity. The variables are independent and the iso-probability surfaces are hyperspheres. c-d) Diagonal covariance matrices permit different non-zero entries on the diagonal, but have zero entries elsewhere. The variables are independent, but scaled differently and the iso-probability surfaces are hyper-ellipsoids (ellipses in 2D) whose principal axes are aligned to the coordinate axes. e-f) Full covariance matrices are symmetric and positive definite. Variables are dependent and iso-probability surfaces are ellipsoids that are not aligned in any special way.

For the bivariate case (figure 5.1), spherical covariances produce circular iso-density contours. Diagonal covariances produce ellipsoidal iso-contours that are aligned with the coordinate axes. Full covariances also produce ellipsoidal iso-density contours, but these may now take an arbitrary orientation. More generally, in D dimensions, spherical covariances produce iso-contours that are D-spheres, diagonal covariances produce iso-contours that are D-dimensional ellipsoids aligned with the coordinate axes, and full covariances produce iso-contour that are n-dimensional ellipsoids in general position.

When the covariance is spherical or diagonal, the individual variables are independent. For example, for the bivariate diagonal case with zero mean, we have

$$\begin{aligned} Pr(x_1, x_2) &= \frac{1}{2\pi\sqrt{|\boldsymbol{\Sigma}|}} \exp\left[-0.5 \begin{pmatrix} x_1 & x_2 \end{pmatrix} \boldsymbol{\Sigma}^{-1} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right] \\ &= \frac{1}{2\pi\sigma_1\sigma_2} \exp\left[-0.5 \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} \sigma_1^{-2} & 0 \\ 0 & \sigma_2^{-2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right] \end{aligned}$$

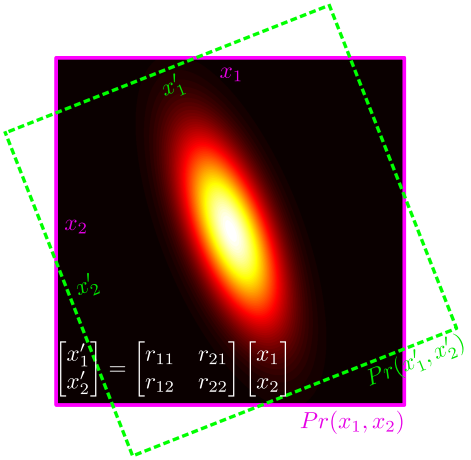


Figure 5.2 Decomposition of full covariance. For every bivariate normal distribution in variables x_1 and x_2 with full covariance matrix, there exists a coordinate system with variables x_1' and x_2' where the covariance is diagonal: the ellipsoidal iso-contours align with the coordinate axes x_1' and x_2' in this canonical coordinate frame. The two frames of reference are related by the rotation matrix $\mathbf{R}$ which maps (x_1', x_2') to (x_1, x_2). From this it follows (see text) that any covariance matrix $\boldsymbol{\Sigma}$ can be broken down into the product $\mathbf{R}^T\boldsymbol{\Sigma}'_{diag}\mathbf{R}$ of a rotation matrix $\mathbf{R}$ and a diagonal covariance matrix $\boldsymbol{\Sigma}'_{diag}$.

$$\begin{aligned} &= \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left[-\frac{x_1^2}{2\sigma_1^2}\right] \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left[-\frac{x_2^2}{2\sigma_2^2}\right] \\ &= Pr(x_1)Pr(x_2). \end{aligned} \tag{5.4}$$

5.2 Decomposition of covariance

We can use the foregoing geometrical intuitions to decompose the full covariance matrix $\boldsymbol{\Sigma}_{full}$. Given a normal distribution with mean zero and a full covariance matrix, we know that the iso-contours take an ellipsoidal form with the major and minor axes at arbitrary orientations.

Now consider viewing the distribution in a new coordinate frame where the axes *are* aligned with the axes of the normal (figure 5.2): in this new frame of reference, the covariance matrix $\boldsymbol{\Sigma}'_{diag}$ will be diagonal. We denote the data vector in the new coordinate system by $\mathbf{x}' = [x_1', x_2']^T$ where the frames of reference are related by $\mathbf{x}' = \mathbf{R}\mathbf{x}$. We can write the probability distribution over $\mathbf{x}'$ as

$$Pr(\mathbf{x}') = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}'_{diag}|^{1/2}} \exp\left[-0.5\mathbf{x}'^T\boldsymbol{\Sigma}'^{-1}_{diag}\mathbf{x}'\right]. \tag{5.5}$$

We now convert back to the original axes by substituting in $\mathbf{x}' = \mathbf{R}\mathbf{x}$ to get

$$\begin{aligned} Pr(\mathbf{x}) &= \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}'_{diag}|^{1/2}} \exp\left[-0.5(\mathbf{R}\mathbf{x})^T\boldsymbol{\Sigma}'^{-1}_{diag}\mathbf{R}\mathbf{x}\right] \\ &= \frac{1}{(2\pi)^{D/2}|\mathbf{R}^T\boldsymbol{\Sigma}'_{diag}\mathbf{R}|^{1/2}} \exp\left[-0.5\mathbf{x}^T(\mathbf{R}^T\boldsymbol{\Sigma}'_{diag}\mathbf{R})^{-1}\mathbf{x}\right] \end{aligned} \tag{5.6}$$

where we have used $|\mathbf{R}^T\boldsymbol{\Sigma}'\mathbf{R}| = |\mathbf{R}^T|.|\boldsymbol{\Sigma}'|.|\mathbf{R}| = 1.|\boldsymbol{\Sigma}'|.1 = |\boldsymbol{\Sigma}'|$. Equation 5.6 is a multivariate normal with covariance

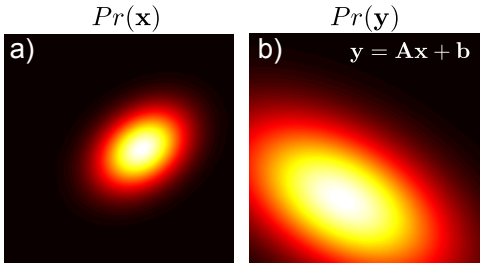


Figure 5.3 Transformation of normal variables. a) If $\mathbf{x}$ has a multivariate normal pdf and we apply a linear transformation to create new variable $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b}$, then b) the distribution of $\mathbf{y}$ is also multivariate normal. The mean and covariance of $\mathbf{y}$ depend on the original mean and covariance of $\mathbf{x}$ and the parameters $\mathbf{A}$ and $\mathbf{b}$.

$$\boldsymbol{\Sigma}_{full} = \mathbf{R}^T \boldsymbol{\Sigma}'_{diag} \mathbf{R}. \tag{5.7}$$

We conclude that full covariance matrices are expressible as a product of this form involving a rotation matrix $\mathbf{R}$ and a diagonal covariance matrix $\boldsymbol{\Sigma}'_{diag}$. Having understood this, it is possible to retrieve these elements from an arbitrary valid covariance matrix $\boldsymbol{\Sigma}_{full}$ by decomposing it in this way using the singular value decomposition.

The matrix $\mathbf{R}$ contains the principal directions of the ellipsoid in its columns. The values on the diagonal of $\boldsymbol{\Sigma}'_{diag}$ encode the variance (and hence the width of the distribution) along each of these axes. Hence we can use the results of the eigen-decomposition to answer questions about which directions in space are most and least certain.

5.3 Linear transformations of variables

Problem 5.1
Problem 5.2

The form of the multivariate normal is preserved under linear transformations $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b}$ (figure 5.3). If the original distribution was

$$Pr(\mathbf{x}) = \text{Norm}_{\mathbf{x}}\left[\boldsymbol{\mu}, \boldsymbol{\Sigma}\right], \tag{5.8}$$

then the transformed variable $\mathbf{y}$ is distributed as:

$$Pr(\mathbf{y}) = \text{Norm}_{\mathbf{y}}\left[\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^T\right]. \tag{5.9}$$

This relationship provides a simple method to draw samples from a normal distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$. We first draw a sample $\mathbf{x}$ from a standard normal distribution (with mean $\boldsymbol{\mu} = \mathbf{0}$ and covariance $\boldsymbol{\Sigma} = \mathbf{I}$) and then apply the transform $\mathbf{y} = \boldsymbol{\Sigma}^{1/2}\mathbf{x} + \boldsymbol{\mu}$.

5.4 Marginal distributions

Problem 5.3

If we marginalize over any subset of random variables in a multivariate normal

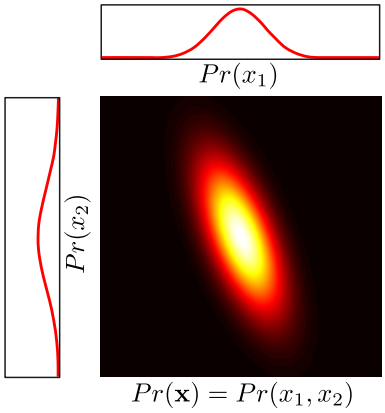


Figure 5.4 The marginal distribution of any subset of variables in a normal distribution is also normally distributed. In other words, if we sum over the distribution in any direction, the remaining quantity is also normally distributed. To find the mean and the covariance of the new distribution, we can simply extract the relevant entries from the original mean and covariance matrix.

distribution, the remaining distribution is also normally distributed (figure 5.4). If we partition the original random variable into two parts $\mathbf{x} = [\mathbf{x}_1^T, \mathbf{x}_2^T]^T$ so that

$$Pr(\mathbf{x}) = Pr\left(\begin{bmatrix}\mathbf{x}_1\\ \mathbf{x}_2\end{bmatrix}\right) = \text{Norm}_{\mathbf{x}}\left[\begin{bmatrix}\boldsymbol{\mu}_1\\ \boldsymbol{\mu}_2\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{21}^T\\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22}\end{bmatrix}\right], \tag{5.10}$$

then

$$\begin{aligned} Pr(\mathbf{x}_1) &= \text{Norm}_{\mathbf{x}_1}\left[\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11}\right] \\ Pr(\mathbf{x}_2) &= \text{Norm}_{\mathbf{x}_2}\left[\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22}\right]. \end{aligned} \tag{5.11}$$

So, to find the mean and covariance of the marginal distribution of a subset of variables, we extract the relevant entries from the original mean and covariance.

5.5 Conditional distributions

If the variable $\mathbf{x}$ is distributed as a multivariate normal, then the conditional distribution of a subset of variables $\mathbf{x}_1$ given known values for the remaining variables $\mathbf{x}_2$ is also distributed as a multivariate normal (figure 5.5). If

Problem 5.4
Problem 5.5
Problem 5.6

$$Pr(\mathbf{x}) = Pr\left(\begin{bmatrix}\mathbf{x}_1\\ \mathbf{x}_2\end{bmatrix}\right) = \text{Norm}_{\mathbf{x}}\left[\begin{bmatrix}\boldsymbol{\mu}_1\\ \boldsymbol{\mu}_2\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{21}^T\\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22}\end{bmatrix}\right], \tag{5.12}$$

then the conditional distributions are

$$\begin{aligned} Pr(\mathbf{x}_1|\mathbf{x}_2 = \mathbf{x}_2^*) &= \text{Norm}_{\mathbf{x}_1}\left[\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{21}^T\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{x}_2^* - \boldsymbol{\mu}_2), \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{21}^T\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}\right] \\ Pr(\mathbf{x}_2|\mathbf{x}_1 = \mathbf{x}_1^*) &= \text{Norm}_{\mathbf{x}_2}\left[\boldsymbol{\mu}_2 + \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}(\mathbf{x}_1^* - \boldsymbol{\mu}_1), \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{21}^T\right]. \end{aligned} \tag{5.13}$$

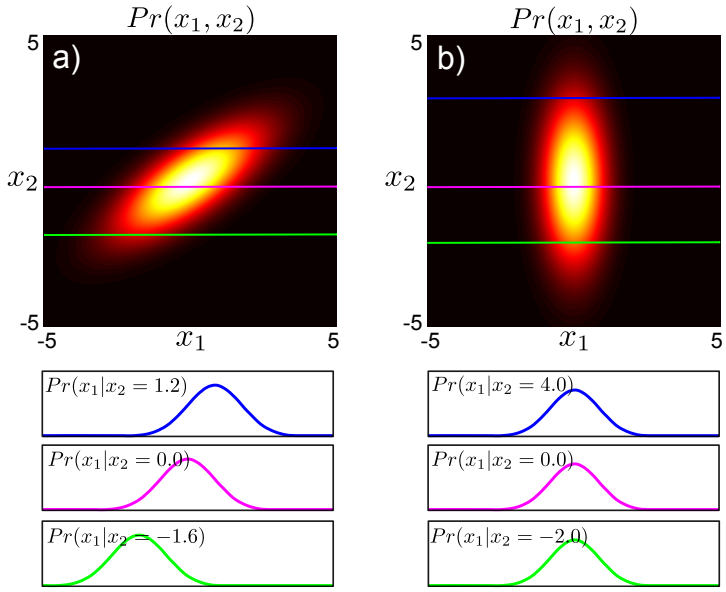


Figure 5.5 Conditional distributions of multivariate normal. a) If we take any multivariate normal distribution, fix a subset of the variables, and look at the distribution of the remaining variables, this distribution will also take the form of a normal. The mean of this new normal depends on the values that we fixed the subset to, but the covariance is always the same. b) If the original multivariate normal has spherical or diagonal covariance, both the mean and covariance of the resulting normal distributions are the same, regardless of the value we conditioned on: these forms of covariance matrix imply independence between the constituent variables.

5.6 Product of two normals

Problem 5.7
Problem 5.8

The product of two normal distributions is proportional to a third normal distribution (figure 5.6). If the two original distributions have means $\mathbf{a}$ and $\mathbf{b}$ and covariances $\mathbf{A}$ and $\mathbf{B}$, respectively, then we find that

$$\begin{aligned}\text{Norm}_{\mathbf{x}}[\mathbf{a}, \mathbf{A}]\text{Norm}_{\mathbf{x}}[\mathbf{b}, \mathbf{B}] &= \\ \kappa \cdot \text{Norm}_{\mathbf{x}}\left[\left(\mathbf{A}^{-1}+\mathbf{B}^{-1}\right)^{-1}\left(\mathbf{A}^{-1}\mathbf{a}+\mathbf{B}^{-1}\mathbf{b}\right), \left(\mathbf{A}^{-1}+\mathbf{B}^{-1}\right)^{-1}\right],\end{aligned} \tag{5.14}$$

Problem 5.9

where the constant κ is itself a normal distribution,

$$\kappa = \text{Norm}_{\mathbf{a}}[\mathbf{b}, \mathbf{A}+\mathbf{B}] = \text{Norm}_{\mathbf{b}}[\mathbf{a}, \mathbf{A}+\mathbf{B}]. \tag{5.15}$$

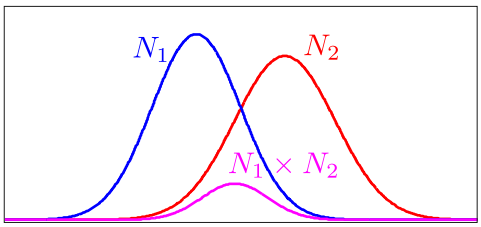


Figure 5.6 The product of any two normals N_1 and N_2 is proportional to a third normal distribution, with a mean between the two original means and a variance that is smaller than either of the original distributions.

5.6.1 Self-conjugacy

The preceding property can be used to demonstrate that the normal distribution is *self-conjugate* with respect to its mean $\boldsymbol{\mu}$. Consider taking a product of a normal distribution over data $\mathbf{x}$ and a second normal distribution over the mean vector $\boldsymbol{\mu}$ of the first distribution. It is easy to show from equation 5.14 that

$$\begin{aligned}\text{Norm}_{\mathbf{x}}[\boldsymbol{\mu}, \boldsymbol{\Sigma}]\text{Norm}_{\boldsymbol{\mu}}[\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p] &= \text{Norm}_{\boldsymbol{\mu}}[\mathbf{x}, \boldsymbol{\Sigma}]\text{Norm}_{\boldsymbol{\mu}}[\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p] \\ &= \kappa \cdot \text{Norm}_{\boldsymbol{\mu}}[\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}}], \end{aligned} \tag{5.16}$$

which is the definition of conjugacy (see section 3.9). The new parameters $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\Sigma}}$ are determined from equation 5.14. This analysis assumes that the variance $\boldsymbol{\Sigma}$ is being treated as a fixed quantity. If we also treat this as uncertain, then we must use a normal inverse Wishart prior.

5.7 Change of variable

Consider a normal distribution in variable $\mathbf{x}$ whose mean is a linear function $\mathbf{A}\mathbf{y}+\mathbf{b}$ of a second variable $\mathbf{y}$. We can re-express this in terms of a normal distribution in $\mathbf{y}$ which is a linear function $\mathbf{A}'\mathbf{x}+\mathbf{b}'$ of $\mathbf{x}$ so that

$$\text{Norm}_{\mathbf{x}}[\mathbf{A}\mathbf{y}+\mathbf{b}, \boldsymbol{\Sigma}] = \kappa \cdot \text{Norm}_{\mathbf{y}}[\mathbf{A}'\mathbf{x}+\mathbf{b}', \boldsymbol{\Sigma}'], \tag{5.17}$$

where κ is a constant and the new parameters are given by

Problem 5.10

$$\begin{aligned}\boldsymbol{\Sigma}' &= (\mathbf{A}^T\boldsymbol{\Sigma}^{-1}\mathbf{A})^{-1} \\ \mathbf{A}' &= (\mathbf{A}^T\boldsymbol{\Sigma}^{-1}\mathbf{A})^{-1}\mathbf{A}^T\boldsymbol{\Sigma}^{-1} \\ \mathbf{b}' &= -(\mathbf{A}^T\boldsymbol{\Sigma}^{-1}\mathbf{A})^{-1}\mathbf{A}^T\boldsymbol{\Sigma}^{-1}\mathbf{b}. \end{aligned} \tag{5.18}$$

This relationship is mathematically opaque, but it is easy to understand visually when x and y are scalars (figure 5.7). It is often used in the context of Bayes' rule where our goal is to move from $Pr(\mathbf{x}|\mathbf{y})$ to $Pr(\mathbf{y}|\mathbf{x})$.

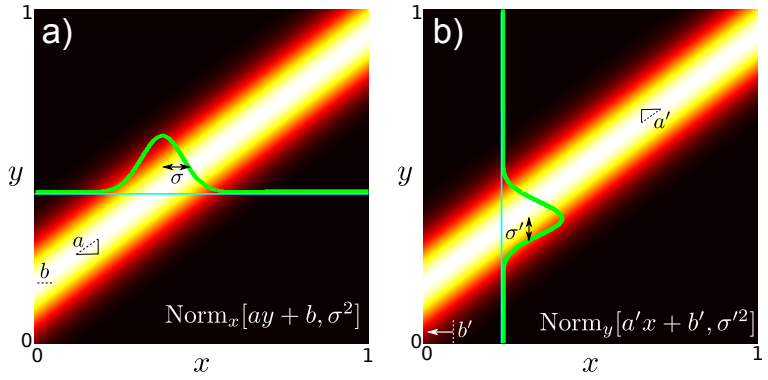


Figure 5.7 a) Consider a normal distribution in x whose variance σ^2 is constant, but whose mean is a linear function $ay + b$ of a second variable y. b) This is mathematically equivalent to a constant κ times a normal distribution in y whose variance σ'^2 is constant and whose mean is a linear function $a'x + b'$ of x.

Summary

In this chapter we have presented a number of properties of the multivariate normal distribution. The most important of these relate to the marginal and conditional distributions: when we marginalize or take the conditional distribution of a normal with respect to a subset of variables, the result is another normal. These properties are exploited in many vision algorithms.

Notes

The normal distribution has further interesting properties which are not discussed because they are not relevant for this book. For example, the convolution of a normal distribution with a second normal distribution produces a function that is proportional to a third normal, and the Fourier transform of a normal profile creates a normal profile in frequency space. For a different treatment of this topic the interested reader can consult chapter 2 of Bishop (2006).

Problems

Problem 5.1 Consider a multivariate normal distribution in variable $\mathbf{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$. Show that if we make the linear transformation $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b}$ then the transformed variable $\mathbf{y}$ is distributed as:

$$Pr(\mathbf{y}) = \text{Norm}_{\mathbf{y}}\left[\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^T\right].$$

Problem 5.2 Show that we can convert a normal distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ to a new distribution with mean $\mathbf{0}$ and covariance $\mathbf{I}$ using the linear transformation $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b}$ where

$$\begin{aligned}\mathbf{A} &= \boldsymbol{\Sigma}^{-1/2} \\ \mathbf{b} &= -\boldsymbol{\Sigma}^{-1/2}\boldsymbol{\mu}.\end{aligned}$$

This is known as the *whitening* transform.

Problem 5.3 Show that for multivariate normal distribution

$$Pr(\mathbf{x}) = Pr\left(\begin{bmatrix}\mathbf{x}_1 \\ \mathbf{x}_2\end{bmatrix}\right) = \text{Norm}_{\mathbf{x}}\left[\begin{bmatrix}\boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{21}^T \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22}\end{bmatrix}\right],$$

the marginal distribution in $\mathbf{x}_1$ is

$$Pr(\mathbf{x}_1) = \text{Norm}_{\mathbf{x}_1}\left[\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11}\right].$$

Hint: apply the transformation $\mathbf{y} = [\mathbf{I}, \mathbf{0}]\mathbf{x}$.

Problem 5.4 The Schur complement identity states that inverse of a matrix in terms of its sub-blocks is

$$\begin{bmatrix}\mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D}\end{bmatrix}^{-1} = \begin{bmatrix}(\mathbf{A}-\mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1} & -(\mathbf{A}-\mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1}\mathbf{B}\mathbf{D}^{-1} \\ -\mathbf{D}^{-1}\mathbf{C}(\mathbf{A}-\mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1} & \mathbf{D}^{-1}+\mathbf{D}^{-1}\mathbf{C}(\mathbf{A}-\mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1}\mathbf{B}\mathbf{D}^{-1}\end{bmatrix}.$$

Show that this relation is true.

Problem 5.5 Prove the conditional distribution property for the normal distribution: if

$$Pr(\mathbf{x}) = Pr\left(\begin{bmatrix}\mathbf{x}_1 \\ \mathbf{x}_2\end{bmatrix}\right) = \text{Norm}_{\mathbf{x}}\left[\begin{bmatrix}\boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12}^T \\ \boldsymbol{\Sigma}_{12} & \boldsymbol{\Sigma}_{22}\end{bmatrix}\right],$$

then

$$Pr(\mathbf{x}_1|\mathbf{x}_2) = \text{Norm}_{\mathbf{x}_1}\left[\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}^T\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{x}_2 - \boldsymbol{\mu}_2), \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}^T\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{12}\right].$$

Hint: use Schur's complement.

Problem 5.6 Use the conditional probability relation for the normal distribution to show that the conditional distribution $Pr(x_1|x_2 = k)$ is the same for all k when the covariance is diagonal and the variables are independent (see figure 5.5b).

Problem 5.7 Show that

$$\text{Norm}_{\mathbf{x}}[\mathbf{a}, \mathbf{A}]\text{Norm}_{\mathbf{x}}[\mathbf{b}, \mathbf{B}] \propto \text{Norm}_{\mathbf{x}}[(\mathbf{A}^{-1} + \mathbf{B}^{-1})^{-1}(\mathbf{A}^{-1}\mathbf{a} + \mathbf{B}^{-1}\mathbf{b}), (\mathbf{A}^{-1} + \mathbf{B}^{-1})^{-1}].$$

Problem 5.8 For the 1D case, show that when we take the product of the two normal distributions with means μ_1, μ_2 and variances σ_1^2, σ_2^2, the new mean lies between the original two means and the new variance is smaller than either of the original variances.

Problem 5.9 Show that the constant of proportionality κ in the product relation in problem 5.7 is also a normal distribution where

$$\kappa = \text{Norm}_{\mathbf{a}}[\mathbf{b}, \mathbf{A} + \mathbf{B}].$$

Problem 5.10 Prove the change of variable relation. Show that

$$\text{Norm}_{\mathbf{x}}[\mathbf{A}\mathbf{y} + \mathbf{b}, \boldsymbol{\Sigma}] = \kappa \cdot \text{Norm}_{\mathbf{y}}[\mathbf{A}'\mathbf{x} + \mathbf{b}', \boldsymbol{\Sigma}'],$$

and derive expressions for κ, $\mathbf{A}'$, $\mathbf{b}'$ and $\boldsymbol{\Sigma}'$. Hint: write out the terms in the original exponential, extract quadratic and linear terms in $\mathbf{y}$, and complete the square.

Part II

Machine learning for machine vision

Part II: Machine learning for machine vision

In the second part of this book (chapters 6-9), we treat vision as a machine learning problem and disregard everything we know about the creation of the image. For example, we will not exploit our understanding of perspective projection or light transport. Instead, we treat vision as pattern recognition; we interpret new image data based on prior experience of images in which the contents were known. We divide this process into two parts: in *learning* we model the relationship between the image data and the scene content. In *inference*, we exploit this relationship to predict the contents of new images.

To abandon useful knowledge about image creation may seem odd, but the logic is twofold. First, these same learning and inference techniques will also underpin our algorithms when image formation is taken into account. Second, it is possible to achieve a great deal with a pure learning approach to vision. For many tasks, knowledge of the image formation process is genuinely unnecessary.

The structure of part II is as follows: in chapter 6 we present a taxonomy of models that relate the measured image data and the actual scene content. In particular, we distinguish between *generative* models and *discriminative* models. For generative models, we build a probability model of the data and parameterize it by the scene content. For discriminative models, we build a probability model of the scene content and parameterize it by the data. In the subsequent three chapters, we elaborate our discussion of these models.

In chapter 7 we consider generative models. In particular, we discuss how to use *hidden variables* to construct complex probability densities over visual data. As examples, we consider mixtures of Gaussians, t-distributions, and factor analyzers. Together, these three models allow us to build densities that are multi-modal, robust, and suitable for modeling high dimensional data.

In chapter 8 we consider *regression* models: we aim to estimate a continuous quantity from continuous data. For example, we might want to predict the joint angles from an image of the human body. We start with linear regression and move to more complex nonlinear methods such as Gaussian process regression and relevance vector regression. In chapter 9 we consider *classification* models: here we want to predict a discrete quantity from continuous data. For example, we might want to assign a label to a region of the image to indicate whether or not a face is present. We start with logistic regression and work toward more sophisticated methods such as Gaussian process classification, boosting, and classification trees.

Chapter 6

Learning and inference in vision

At an abstract level, the goal of computer vision problems is to use the observed image data to infer something about the world. For example, we might observe adjacent frames of a video sequence and infer the camera motion, or we might observe a facial image and infer the identity.

The aim of this chapter is to describe a mathematical framework for solving this type of problem and to organize the resulting models into useful subgroups, which will be explored in subsequent chapters.

6.1 Computer vision problems

In vision problems, we take visual data $\mathbf{x}$ and use them to infer the state of the world $\mathbf{w}$. The world state $\mathbf{w}$ may be continuous (e.g., the 3D pose of a body model) or discrete (e.g., the presence or absence of a particular object). When the state is continuous, we call the inference process *regression*. When the state is discrete, we call it *classification*.

Unfortunately, the measurements $\mathbf{x}$ may be compatible with more than one world state $\mathbf{w}$. The measurement process is noisy and there is inherent ambiguity in visual data: a lump of coal viewed under bright light may produce the same luminance measurements as white paper in dim light. Similarly, a small object seen close-up may produce the same image as a larger object that is further away.

In the face of such ambiguity, the best that we can do is to return the *posterior probability distribution* $Pr(\mathbf{w}|\mathbf{x})$ over possible states $\mathbf{w}$. This describes everything we know about the state after observing the visual data. So, a more precise description of an abstract vision problem is that we wish to take observations $\mathbf{x}$ and return the whole posterior probability distribution $Pr(\mathbf{w}|\mathbf{x})$ over world states.

In practice, computing the posterior is not always tractable; we often have to settle for returning the world state $\hat{\mathbf{w}}$ at the peak of the posterior (the maximum a posteriori solution). Alternatively, we might draw samples from the posterior and use the collection of samples as an approximation to the full distribution.

6.1.1 Components of the solution

To solve a vision problem of this kind, we need three components.

- We need a *model* that mathematically relates the visual data $\mathbf{x}$ and the world state $\mathbf{w}$. The model specifies a family of possible relationships between $\mathbf{x}$ and $\mathbf{w}$ and the particular relationship is determined by the model parameters $\boldsymbol{\theta}$.
- We need a *learning algorithm* that allows us to fit the parameters $\boldsymbol{\theta}$ using paired training examples $\{\mathbf{x}_i, \mathbf{w}_i\}$ where we know both the measurements and the underlying state.
- We need an *inference algorithm* that takes a new observation $\mathbf{x}$ and uses the model to return the posterior $Pr(\mathbf{w}|\mathbf{x}, \boldsymbol{\theta})$ over the world state $\mathbf{w}$. Alternately, it might return the MAP solution or draw samples from the posterior.

The rest of this book is structured around these components: each chapter focusses on one model or one family of models, and discusses the associated learning and inference algorithms.

6.2 Types of model

The first and most important component of the solution is the model. Models relating the data $\mathbf{x}$ to the world $\mathbf{w}$ fall into one of two categories. We either:

1. model the contingency of the world state on the data $Pr(\mathbf{w}|\mathbf{x})$ or
2. model the contingency of the data on the world state $Pr(\mathbf{x}|\mathbf{w})$.

The first type of model is termed *discriminative*. The second is termed *generative*; here, we construct a probability model over the data and this can be used to generate (confabulate) new observations. Let us consider these two types of model in turn and discuss learning and inference in each.

6.2.1 Model contingency of world on data (discriminative)

To model $Pr(\mathbf{w}|\mathbf{x})$, we choose an appropriate form for the distribution $Pr(\mathbf{w})$ over the world state $\mathbf{w}$ and then make the distribution parameters a function of the data $\mathbf{x}$. So if the world state was continuous, we might model $Pr(\mathbf{w})$ with a normal distribution and make the mean $\boldsymbol{\mu}$ a function of the data $\mathbf{x}$.

The value that this function returns also depends on a set of parameters, $\boldsymbol{\theta}$. Since the distribution over the state depends on both the data and these parameters, we write it as $Pr(\mathbf{w}|\mathbf{x}, \boldsymbol{\theta})$ and refer to it as the *posterior distribution*.

The goal of the learning algorithm is to fit the parameters $\boldsymbol{\theta}$ using paired training data $\{\mathbf{x}_i, \mathbf{w}_i\}_{i=1}^{I}$. This can be done using the maximum likelihood (ML), maximum a posteriori (MAP), or Bayesian approaches (chapter 4).

The goal of inference is to find a distribution over the possible world states $\mathbf{w}$ for a new observation $\mathbf{x}$. In this case, this is easy: we have already directly

constructed an expression for the posterior distribution $Pr(\mathbf{w}|\mathbf{x}, \boldsymbol{\theta})$, and we simply evaluate it with the new data.

6.2.2 Model contingency of data on world (generative)

To model $Pr(\mathbf{x}|\mathbf{w})$, we choose the form for the distribution $Pr(\mathbf{x})$ over the data and make the distribution parameters a function of the world state $\mathbf{w}$. For example, if the data were discrete and multi-valued then we might use a categorical distribution and make the parameter vector $\boldsymbol{\lambda}$ a function of the world state $\mathbf{w}$.

The value that this function returns also depends on a set of parameters $\boldsymbol{\theta}$. Since the distribution $Pr(\mathbf{x})$ now depends on both the world state and these parameters, we write it as $Pr(\mathbf{x}|\mathbf{w}, \boldsymbol{\theta})$ and refer to it as the *likelihood*. The goal of learning is to fit the parameters $\boldsymbol{\theta}$ using paired training examples $\{\mathbf{x}_i, \mathbf{w}_i\}_{i=1}^{I}$.

In inference, we aim to compute the posterior distribution $Pr(\mathbf{w}|\mathbf{x})$. To this end we specify a prior $Pr(\mathbf{w})$ over the world state and then use Bayes' rule,

$$Pr(\mathbf{w}|\mathbf{x}) = \frac{Pr(\mathbf{x}|\mathbf{w})Pr(\mathbf{w})}{\int Pr(\mathbf{x}|\mathbf{w})Pr(\mathbf{w})d\mathbf{w}}. \tag{6.1}$$

Here we have modeled both the likelihood $Pr(\mathbf{x}|\mathbf{w})$ and the prior $Pr(\mathbf{w})$ and multiplied these together in the numerator of Bayes' rule. However, notice that we could have equivalently modeled the joint distribution $Pr(\mathbf{x}, \mathbf{w}) = Pr(\mathbf{x}|\mathbf{w})Pr(\mathbf{w})$ directly. Sometimes generative models are presented in this form (see section 7.9.5).

Summary

We've seen that there are two distinct approaches to modeling the relationship between the world state $\mathbf{w}$ and the data $\mathbf{x}$, corresponding to modeling the posterior $Pr(\mathbf{w}|\mathbf{x})$, or the likelihood $Pr(\mathbf{x}|\mathbf{w})$.

The two model types result in different approaches to inference. For the discriminative model, we describe the posterior $Pr(\mathbf{w}|\mathbf{x})$ directly and there is no need for further work. For the generative model, we compute the posterior using Bayes' rule. This sometimes results in complex inference algorithms.

To make these ideas concrete, we now consider two toy examples. For each case, we will investigate using both generative and discriminative models. At this stage, we won't present the details of the learning and inference algorithms; these are presented in subsequent chapters anyway. The goal here is to introduce the main types of model used in computer vision in their most simple form.

6.3 Example 1: regression

Consider the situation where we make a univariate continuous measurement x and use this to predict a univariate continuous state w. For example, we might predict the distance to a car in a road scene based on the number of pixels in its silhouette.

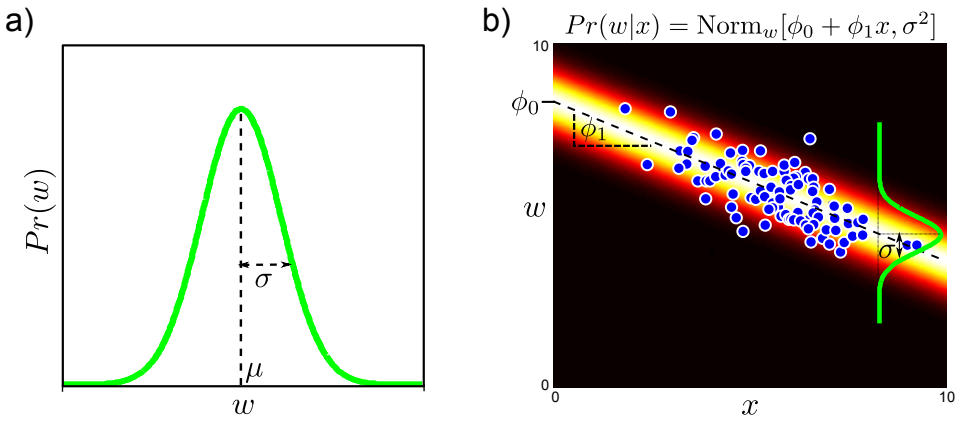


Figure 6.1 Regression by modeling the posterior $Pr(w|x)$ (discriminative). a) We model the world state w as normally distributed. b) We make the normal parameters a function of the observations x: the mean is a linear function $\mu = \phi_0 + \phi_1 x$ of the observations, and the variance σ^2 is fixed. The learning algorithm fits the parameters $\boldsymbol{\theta} = \{\phi_0, \phi_1, \sigma^2\}$ to example training pairs $\{x_i, w_i\}_{i=1}^I$ (blue dots). In inference we take a new observation x and compute the posterior distribution $Pr(w|x)$ over the state.

6.3.1 Model contingency of world on data (discriminative)

Problem 6.5

We define a probability distribution over the world state w and make its parameters contingent on the data x. Since the world state is univariate and continuous, we chose the univariate normal. We fix the variance, σ^2 and make the mean μ a linear function $\phi_0 + \phi_1 x$ of the data. So we have

$$Pr(w|x, \boldsymbol{\theta}) = \text{Norm}_w\left[\phi_0 + \phi_1 x, \sigma^2\right], \tag{6.2}$$

where $\boldsymbol{\theta} = \{\phi_0, \phi_1, \sigma^2\}$ are the unknown parameters of the model (figure 6.1). This model is referred to as *linear regression*.

The learning algorithm estimates the model parameters $\boldsymbol{\theta}$ from paired training examples $\{x_i, w_i\}_{i=1}^I$. For example, in the MAP approach, we seek

$$\begin{aligned}
\hat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}}{\text{argmax}}\left[Pr(\boldsymbol{\theta}|w_{1\ldots I}, x_{1\ldots I})\right] \\
&= \underset{\boldsymbol{\theta}}{\text{argmax}}\left[Pr(w_{1\ldots I}|x_{1\ldots I}, \boldsymbol{\theta})Pr(\boldsymbol{\theta})\right] \\
&= \underset{\boldsymbol{\theta}}{\text{argmax}}\left[\prod_{i=1}^{I} Pr(w_i|x_i, \boldsymbol{\theta})Pr(\boldsymbol{\theta})\right],
\end{aligned} \tag{6.3}$$

where we have assumed that the I training pairs $\{x_i, w_i\}_{i=1}^I$ are independent, and defined a suitable prior $Pr(\boldsymbol{\theta})$.

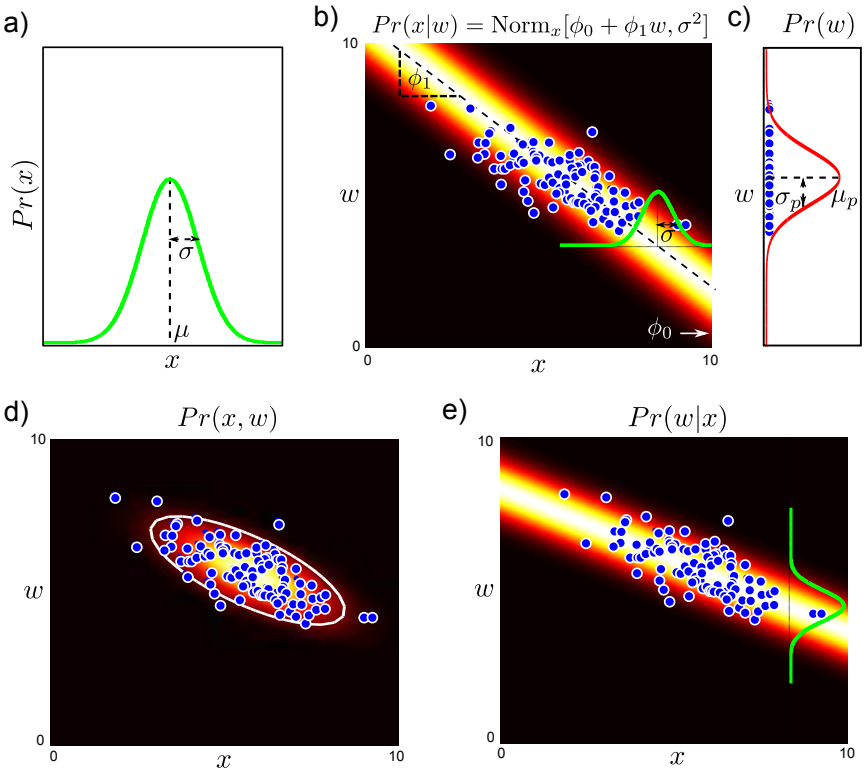


Figure 6.2 Regression by modeling likelihood $Pr(x|w)$ (generative). a) We represent the data x with a normal distribution. b) We make the normal parameters functions of the world state w. Here the mean is a linear function $\mu = \phi_0 + \phi_1 w$ of the world state and the variance σ^2 is fixed. The learning algorithm fits the parameters $\boldsymbol{\theta} = \{\phi_0, \phi_1, \sigma^2\}$ to example training pairs $\{x_i, w_i\}_{i=1}^{I}$ (blue dots). c) We also learn a prior distribution over the world state w (here modeled as a normal distribution with parameters $\boldsymbol{\theta}_p = \{\mu_p, \sigma_p\}$). In inference we take a new datum x and compute the posterior $Pr(w|x)$ over the state. d) This can be done by computing the joint distribution $Pr(x, w) = Pr(x|w)Pr(w)$ (weighting each row of (b) by the appropriate value from the prior) and e) normalizing the columns $Pr(w|x) = Pr(x, w)/Pr(x)$. Together these operations implement Bayes' rule: $Pr(w|x) = Pr(x|w)Pr(w)/Pr(x)$.

We also need an *inference algorithm* that takes visual data x and returns the posterior distribution $Pr(w|x, \boldsymbol{\theta})$. Here this is very simple: we simply evaluate equation 6.2 using the data x and the learned parameters $\hat{\boldsymbol{\theta}}$.

6.3.2 Model the contingency of data on world (generative)

In the generative formulation, we choose a probability distribution over the data x and make its parameters contingent on the world state w. Since the data are univariate and continuous, we will model the data as a normal distribution with fixed variance, σ^2 and a mean μ that is a linear function $\phi_0 + \phi_1 w$ of the world state (figure 6.2) so that

$$Pr(x|w, \boldsymbol{\theta}) = \text{Norm}_x\left[\phi_0 + \phi_1 w, \sigma^2\right]. \tag{6.4}$$

We also need a prior $Pr(w)$ over the world states which might also be normal so

$$Pr(w) = \text{Norm}_w[\mu_p, \sigma_p^2]. \tag{6.5}$$

The learning algorithm fits the parameters $\boldsymbol{\theta} = \{\phi_0, \phi_1, \sigma^2\}$ using paired training data $\{x_i, w_i\}_{i=1}^I$, and fits the parameters $\boldsymbol{\theta}_p = \{\mu_p, \sigma_p^2\}$ using the world states $\{w_i\}_{i=1}^I$. The inference algorithm takes a new datum x and returns the posterior $Pr(w|x)$ over the world state w using Bayes' rule

$$Pr(w|x) = \frac{Pr(x|w)Pr(w)}{Pr(x)} = \frac{Pr(x,w)}{Pr(x)}. \tag{6.6}$$

In this case, the posterior can be computed in closed form and is again normally distributed with fixed variance and a mean that is proportional to the data x.

Discussion

We have presented two models that can be used to estimate the world state w from an observed data example x, based on modeling the posterior $Pr(w|x)$ and the likelihood $Pr(x|w)$, respectively.

The models were carefully chosen so that they predict exactly the same posterior $P(w|x)$ over the world state (compare figures 6.1b and 6.2e). This is only the case with maximum likelihood learning: in the MAP approach we would have placed priors on the parameters, and because each model is parameterized differently, they would, in general, have different effects.

6.4 Example 2: binary classification

As a second example, we will consider the case where the observed measurement x is univariate and continuous, but the world state w is discrete and can take one of two values. For example, we might wish to classify a pixel as belonging to a skin or non-skin region based on observing just the red channel.

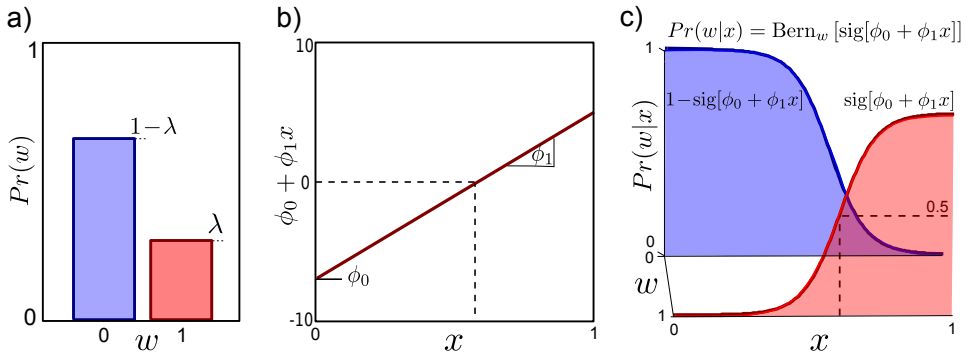


Figure 6.3 Classification by modeling posterior $Pr(w|x)$ (discriminative). a) We represent the world state w as a Bernoulli distribution. We make the Bernoulli parameter λ a function of the observations x. b) To this end we form a linear function $\phi_0 + \phi_1 x$ of the observations. c) The Bernoulli parameter $\lambda = \text{sig}[\phi_0 + \phi_1 x]$ is formed by passing the linear function through the logistic sigmoid $\text{sig}[\bullet]$ to constrain the value to lie between 0 and 1, giving the characteristic sigmoid shape (red curve). In learning we fit parameters $\boldsymbol{\theta} = \{\phi_0, \phi_1\}$ using example training pairs $\{x_i, w_i\}_{i=1}^I$. In inference we take a new datum x and evaluate the posterior $Pr(w|x)$ over the state.

6.4.1 Model contingency of world on data (discriminative)

We define a probability distribution over the world state $w \in \{0, 1\}$ and make its parameters contingent on the data x. Since the world state is discrete and binary, we will use a Bernoulli distribution. This has a single parameter λ, which determines the probability of success so that $Pr(w = 1) = \lambda$.

We make λ a function of the data x, but in doing so we must ensure the constraint $0 \leq \lambda \leq 1$ is obeyed. To this end, we form linear function $\phi_0 + \phi_1 x$ of the data x, which returns a value in the range $[-\infty \ \infty]$. We then pass the result through a function $\text{sig}[\bullet]$ that maps $[-\infty \ \infty]$ to $[0 \ 1]$, so that

$$Pr(w|x) = \text{Bern}_w[\text{sig}[\phi_0 + \phi_1 x]] = \text{Bern}_w\left[\frac{1}{1+\exp[-\phi_0 - \phi_1 x]}\right]. \tag{6.7}$$

This produces a sigmoidal dependence of the distribution parameter λ on the data x (figure 6.3). The function $\text{sig}[\bullet]$ is called the *logistic sigmoid*. This model is confusingly termed *logistic regression* despite being used here for classification.

In learning, we aim to fit the parameters $\boldsymbol{\theta} = \{\phi_0, \phi_1\}$ from paired training examples $\{x_i, w_i\}_{i=1}^I$. In inference, we simply substitute in the observed data value x into equation 6.7 to retrieve the posterior distribution $Pr(w|x)$ over the state.

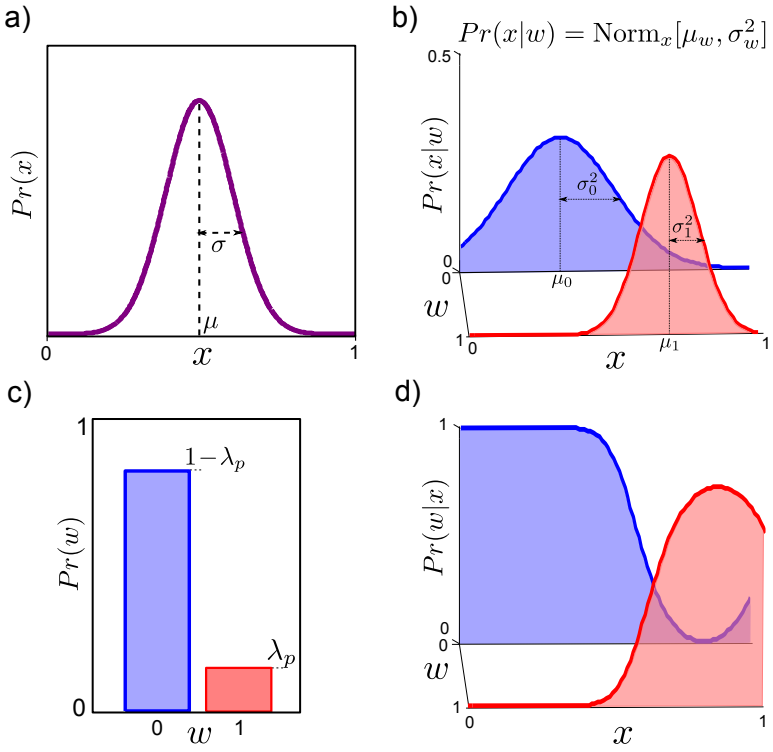


Figure 6.4 Classification by modeling the likelihood $Pr(x|w)$ (generative). a) We choose a normal distribution to represent the data x. b) We make the parameters $\{\mu, \sigma^2\}$ of this normal a function of the world state w. In practice, this means using one set of mean and variance parameters when the world state $w = 0$ and another when $w = 1$. The learning algorithm fits the parameters $\boldsymbol{\theta} = \{\mu_0, \mu_1, \sigma_0^2, \sigma_1^2\}$ to example training pairs $\{x_i, w_i\}_{i=1}^I$. c) We also model the prior probability of the world state w with a Bernoulli distribution with parameter λ_p. d) In inference we take a new datum x and compute the posterior $Pr(w|x)$ over the state using Bayes' rule.

6.4.2 Model contingency of data on world (generative)

Algorithm 6.1

We choose a probability distribution over the data x and make its parameters contingent on the world state w. Since the data are univariate and continuous, we will choose a univariate normal and allow the variance σ^2 and the mean μ to be functions of the binary world state w (figure 6.4) so that the likelihood is

Problem 6.7
Problem 6.8

$$Pr(x|w, \boldsymbol{\theta}) = \text{Norm}_x\left[\mu_w, \sigma_w^2\right]. \tag{6.8}$$

In practice this means that we have one set of parameters $\{\mu_0, \sigma_0^2\}$ when the state of the world is $w = 0$ and a different set $\{\mu_1, \sigma_1^2\}$ when the state is $w = 1$ so

$$\begin{aligned} Pr(x|w=0) &= \text{Norm}_x\left[\mu_0, \sigma_0^2\right] \\ Pr(x|w=1) &= \text{Norm}_x\left[\mu_1, \sigma_1^2\right]. \end{aligned} \tag{6.9}$$

These are referred to as *class conditional density functions* as they model the density of the data for each class separately.

We also define a prior distribution $Pr(w)$ over the world state,

$$Pr(w) = \text{Bern}_w[\lambda_p], \tag{6.10}$$

where λ_p is the prior probability of observing the state $w = 1$.

In learning, we fit the parameters $\boldsymbol{\theta} = \{\mu_0, \sigma_0^2, \mu_1, \sigma_1^2, \lambda_p\}$ using paired training data $\{x_i, w_i\}_{i=1}^I$. In practice, this consists of fitting the parameters μ_0 and σ_0^2 of the first class conditional density function $Pr(x|w=0)$ from just the data x where the state w was 0, and the parameters μ_1 and σ_1^2 of $P(x|w=1)$ from the data x where the state was 1. We learn the prior parameter λ_p from the training world states $\{w_i\}_{i=1}^I$.

The inference algorithm takes new datum x and returns the posterior distribution $Pr(w|x, \boldsymbol{\theta})$ over the world state w using Bayes' rule,

$$Pr(w|x) = \frac{Pr(x|w)Pr(w)}{\sum_{w=0}^{1} Pr(x|w)Pr(w)}. \tag{6.11}$$

This is very easy to compute; we evaluate the two class conditional density functions, weight each by the appropriate prior and normalize so that these two values sum to one.

Discussion

For binary classification, there is an asymmetry between the world state, which is discrete, and the measurements, which are continuous. Consequently, the generative and discriminative models look quite different, and the posteriors over the world state w as a function of the data x have different shapes (compare figure 6.3c with figure 6.4d). For the discriminative model, this function is by definition sigmoidal, but for the generative case it has a more complex form that was implicitly defined by the normal likelihoods. In general, choosing to model $Pr(w|x)$ or $P(x|w)$ will affect the expressiveness of the final model.

6.5 Which type of model should we use?

We have established that there are two different types of model that relate the world state and the observed data. But when should we use each type of model? There is no definitive answer to this question, but some considerations are:

Problem 6.9

- Inference is generally simpler with *discriminative* models. They directly model the conditional probability distribution of the world $Pr(\mathbf{w}|\mathbf{x})$ given

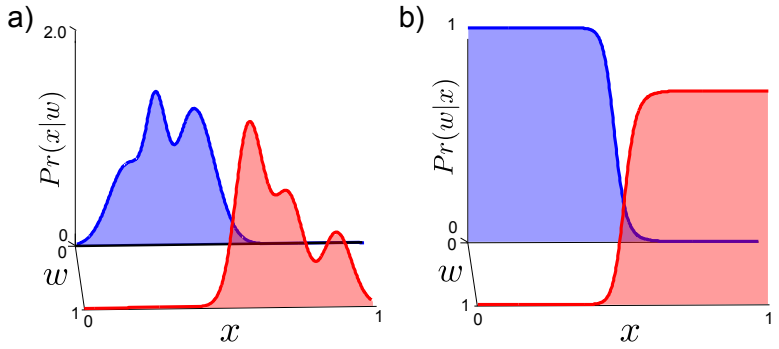


Figure 6.5 Generative vs. discriminative models. a) Generative approach: we separately model the probability $Pr(x|w)$ for each class. This may require a complex model with many parameters. b) Posterior probability distribution $Pr(w|x)$ computed via Bayes' rule with a uniform prior. Notice that the complicated structure of the individual class conditional density functions isn't reflected in the posterior: here, it would have been more efficient to take a discriminative approach and model this posterior directly.

the data. In contrast, generative models calculate this probability via Bayes' rule, and sometimes this requires a computationally expensive algorithm.

- *Generative methods* build probability models $Pr(\mathbf{x}|\mathbf{w})$ over the data whereas *discriminative models* just build a probability model $Pr(\mathbf{w}|\mathbf{x})$ over the world state. The data (usually an image) are generally of much higher dimension than the world state (some aspect of a scene), and modeling it is costly. Moreover, there may be many aspects of the data which do not influence the state; we might devote parameters to describing whether data configuration 1 is more likely than data configuration 2 although they both imply the same world state (figure 6.5).

- Modeling the likelihood $Pr(\mathbf{x}|\mathbf{w})$ mirrors the actual way that the data were created; the state of the world did create the observed data through some physical process (usually light being emitted from a source, interacting with the object and being captured by a camera). If we wish to build information about the generation process into our model, then this approach is desirable. For example, we can account for phenomena such as perspective projection and occlusion. Using the other approaches, it is harder to exploit this knowledge: essentially we have to re-learn these phenomena from the data.

- Sometimes parts of the training or test data vector $\mathbf{x}$ may be missing. Here, generative models are preferred. They model the joint distribution over all of the data dimensions and can effectively interpolate the missing elements.

- A fundamental property of the generative approach is that it allows incorporation of expert knowledge in the form of a prior. It is harder to impose prior knowledge in a principled way in discriminative models.

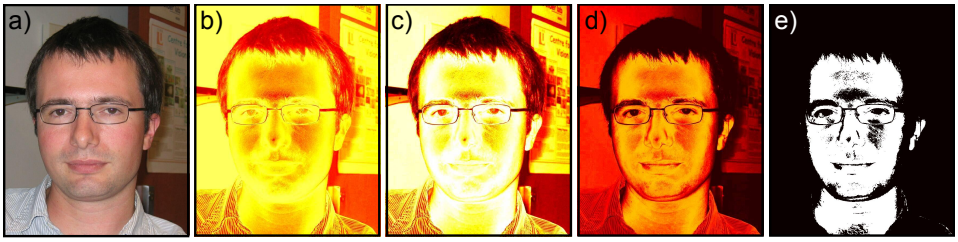


Figure 6.6 Skin detection. For each pixel we aim to infer a label $w \in \{0,1\}$ denoting the absence or presence of skin based on the RGB triple $\mathbf{x}$. Here we modeled the class conditional density functions $Pr(\mathbf{x}|w)$ as normal distributions. a) Original image. b) Log likelihood (log of data assessed under class-conditional density function) for non-skin. c) Log likelihood for skin. d) Posterior probability of belonging to skin class. e) Thresholded posterior probability $Pr(w|\mathbf{x}) > 0.5$ gives estimate of w.

It is notable that generative models are more common in vision applications. Consequently, most of the chapters in the rest of the book concern generative models.

6.6 Applications

The focus of this chapter, and indeed most of the chapters of this book, is on the models themselves and the learning and inference algorithms. From this point forward, we will devote a section at the end of each chapter to discussing practical applications of the relevant models in computer vision. For this chapter, only one of the models can actually be implemented based on the information presented so far. This is the generative classification model from section 6.4.2. Consequently, we will focus the applications on variations of this model and return to the other models in subsequent chapters.

6.6.1 Skin detection

The goal of skin-detection algorithms is to infer a label $w \in \{0,1\}$ denoting the presence or absence of skin at a pixel, based on the RGB measurements $\mathbf{x} = [x^R, x^G, x^B]$ at that pixel. This is a useful precursor to segmenting a face or hand, or it may be used as the basis of a crude method for detecting prurient content in web images. Taking a generative approach, we describe the likelihoods as

$$Pr(\mathbf{x}|w=k) = \text{Norm}_{\mathbf{x}}[\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k] \tag{6.12}$$

and the prior probability over states as

$$Pr(w) = \text{Bern}_w[\lambda]. \tag{6.13}$$

In the learning algorithm, we estimate the parameters $\boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_0, \boldsymbol{\Sigma}_1$ from training data pairs $\{w_i, \mathbf{x}_i\}_{i=1}^I$ where the pixels have been labeled by hand. In particular we learn $\boldsymbol{\mu}_0$ and $\boldsymbol{\Sigma}_0$ from the subset of the training data where $w_i = 0$ and $\boldsymbol{\mu}_1$ and $\boldsymbol{\Sigma}_1$ from the subset where $w_i = 1$. The prior parameter is learned from the world states $\{w_i\}_{i=1}^I$ alone. In each case, this involves fitting a probability distribution to data using one of the techniques discussed in chapter 4.

To classify a new data point $\mathbf{x}$ as skin or non-skin we apply Bayes' rule

$$Pr(w=1|\mathbf{x}) = \frac{Pr(\mathbf{x}|w=1)Pr(w=1)}{\sum_{k=0}^{1} Pr(\mathbf{x}|w=k)Pr(w=k)}, \tag{6.14}$$

and denote this pixel as skin if $Pr(w=1|\mathbf{x}) > 0.5$. Figure 6.6 shows the result of applying this model at each pixel independently in the image. Note that the classification is not perfect: there is genuinely an overlap between the skin- and non-skin distributions and this inevitably results in misclassified pixels. The results could be improved by exploiting the fact that skin areas tend to be contiguous regions without small holes. To do this, we must somehow connect together all of the per-pixel classifiers. This is the topic of chapters 11 and 12.

We briefly note that the RGB data are naturally discrete with $x^R, x^G, x^B \in \{0, 1, \ldots, 255\}$, and we could alternatively have based our skin detection model on this assumption. For example, modeling the three color channels independently, the likelihoods become

$$Pr(\mathbf{x}|w=k) = \text{Cat}_{x^R}[\boldsymbol{\lambda}_k^R]\text{Cat}_{x^G}[\boldsymbol{\lambda}_k^G]\text{Cat}_{x^B}[\boldsymbol{\lambda}_k^B]. \tag{6.15}$$

We refer to the assumption that the elements of the data vector are independent as *naïve Bayes*. Of course, it is not necessarily valid in the real world. To model the joint distribution of the R,G, and B components, we might combine them to form one variable with 256^3 entries and model this with a single categorical distribution. Unfortunately, this means we must learn 256^3 parameters for each categorical distribution, and so it is more practical to quantize each channel to fewer levels (say 8) before combining them together.

6.6.2 Background subtraction

Problem 6.10

A second application of the generative classification model is for background subtraction. Here, the goal is to infer a binary label $w_n \in \{0, 1\}$ which indicates whether the n^{th} pixel in the image is part of a known background ($w = 0$) or whether a foreground object is occluding it ($w = 1$). As for the skin detection model, this is based on its RGB pixel data $\mathbf{x}_n$ at that pixel.

It is usual to have training data $\{\mathbf{x}_{in}\}_{i=1,n=1}^{I,N}$ that consists of a number of empty scenes where all pixels are known to be background. However, it is not typical to have examples of the foreground objects which are highly variable in appearance.

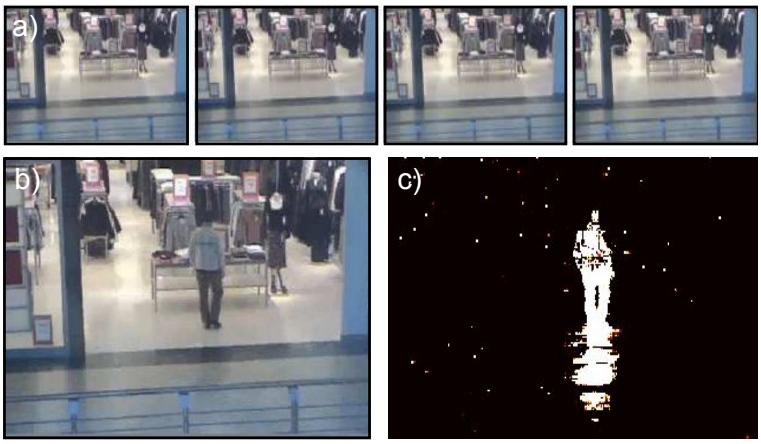


Figure 6.7 Background subtraction. For each pixel we aim to infer a label $w \in \{0, 1\}$ denoting the absence or presence of a foreground object. a) We learn a class conditional density model $Pr(\mathbf{x}|w)$ for the background from training examples of an empty scene. The foreground model is treated as uniform. b) For a new image, we then compute the posterior distribution using Bayes' rule. c) Posterior probability of being foreground $Pr(w = 1|\mathbf{x})$. Images from CAVIAR database.

For this reason, we model the class conditional distribution of the background as a normal distribution

$$Pr(\mathbf{x}_n|w = 0) = \text{Norm}_{\mathbf{x}_n}[\boldsymbol{\mu}_{n0}, \boldsymbol{\Sigma}_{n0}], \tag{6.16}$$

but model the foreground class as a uniform distribution

$$Pr(\mathbf{x}_n|w = 1) = \begin{cases} 1/255^3 & 0 < x_n^R, x_n^G, x_n^B < 255 \\ 0 & \text{otherwise} \end{cases}, \tag{6.17}$$

and again model the prior as a Bernoulli variable.

To compute the posterior distribution we once more apply Bayes' rule. Typical results are shown in figure 6.7, which illustrates a common problem with this method: shadows are often misclassified as foreground. A simple way to remedy this is to classify pixels based on the hue alone.

In some situations we need a more complex distribution to describe the background. For example, consider an outdoor scene in which trees are blowing in the wind (figure 6.8). Certain pixels may have bimodal distributions where one part of the foliage intermittently moves in front of another. It is clear that the unimodal normal likelihood cannot provide a good description of this data, and the resulting background segmentation result will be poor. We devote part of the next chapter to methods for describing more complex probability distributions of this type.

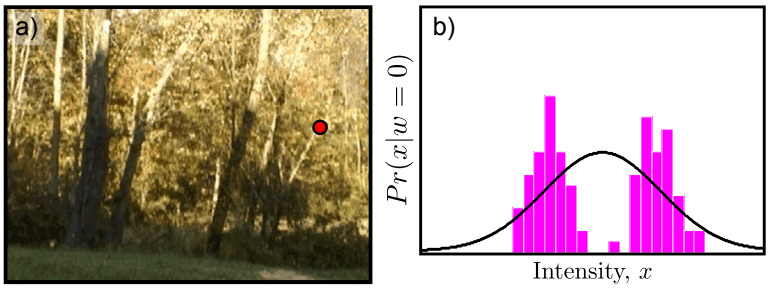


Figure 6.8 Background subtraction in deforming scene. a) The foliage is blowing in the wind in the training images. b) The distribution of RGB values at the pixel indicated by the circle in (a) is now bimodal and not well described by a normal density function (red channel only shown). Images from video by Terry Boult.

Summary

In this chapter, we have provided an overview of how abstract vision problems can be solved using machine learning techniques. We have illustrated these ideas with some simple examples. We did not provide the implementation level details of the learning and inference algorithms; these are presented in subsequent chapters.

	Model $Pr(w\|x)$	Model $Pr(x\|w)$
Regression $x \in [-\infty, \infty], w \in [-\infty, \infty]$	Linear regression	Linear regression
Classification $x \in [-\infty, \infty], w \in \{0, 1\}$	Logistic regression	Probability density function

Table 6.1: Example models in this chapter. These can be categorized into those that are based on modeling probability density functions, those that are based on linear regression, and those that are based on logistic regression.

The examples in this chapter are summarized in table 6.1, where it can be seen that there are three distinct types of model. First, there are those that depend on building probability density functions (describing the class conditional density functions $Pr(x|w = k)$). In the following chapter, we investigate building complex probability density models. The second type of model is based on linear regression, and chapter 8 investigates a family of related algorithms. Finally, the third type of model discussed in this chapter was logistic regression. We will elaborate on the logistic regression model in chapter 9.

Notes

The goal of this chapter was to give a very compact view of learning and inference in vision. Alternative views of this material which are not particularly aimed at vision can be found in Bishop (2006) and Duda *et al.* (2001) and many other texts.

Skin Detection: Reviews of skin detection can be found in Kakumanu *et al.* (2007) and Vezhnevets *et al.* (2003). Pixel-based skin-segmentation algorithms have been variously used as the basis for face detection (Hsu *et al.* 2002), hand gesture analysis (Zhu *et al.* 2000) and filtering of pornographic images (Jones & Rehg 2002).

There are two main issues that affect the quality of the final results: the representation of the pixel color and the classification algorithm. With regard to the latter issue, various generative approaches have been investigated, including methods based on normal distributions (Hsu *et al.* 2002), mixtures of normal distributions (Jones & Rehg 2002) and categorical distributions (Jones & Rehg 2002) as well as discriminative methods such as the multi-layer perceptron (Phung *et al.* 2005). There are several detailed empirical studies that compare the efficacy of the color representation and classification algorithm (Phung *et al.* 2005; Brand & Mason 2000; Schmugge *et al.* 2007)

Background Subtraction: Reviews of background subtraction techniques can be found in Piccardi (2004), Bouwmans *et al.* (2010), and Elgammal (2011). Background subtraction is a common first step in many vision systems as it quickly identifies regions of the image that are of interest. Generative classification systems have been built based on normal distributions (Wren *et al.* 1997), mixtures of normal distribution (Stauffer & Grimson 1999), and kernel density functions (Elgammal *et al.* 2000). Several systems (Friedman & Russell 1997; Horprasert *et al.* 2000) have incorporated an explicit label in the model to identify shadows.

Most recent research in this area has addressed maintenance of the background model in changing environments. Many systems such as that of Stauffer & Grimson (1999) are adaptive and can incorporate new objects into the background model when the background changes. Other models compensate for lighting changes by exploiting the fact that all of the background pixels change together and describing this covariance with a subspace model (Oliver *et al.* 2000). It is also common now to abandon the per-pixel approach and to estimate the whole label field simultaneously using a technique based on Markov random fields (e.g., Sun *et al.* 2006).

Problems

Problem 6.1 Consider the following problems.

i Determining the gender of an image of a face.
ii Determining the pose of the human body given an image of the body.
iii Determining which suit a playing card belongs to based on an image of that card.
iv Determining whether two images of faces match (face verification).
v Determining the 3D position of a point given the positions to which it projects in two cameras at different positions in the world (stereo reconstruction).

For each case, try to describe the contents of the world state $\mathbf{w}$ and the data $\mathbf{x}$. Is each discrete or continuous? If discrete, then how many values can it take? Which are regression problems and which are classification problems?

Problem 6.2 Describe a classifier that relates univariate discrete data $x \in \{1 \ldots K\}$ to a univariate discrete world state $w \in \{1 \ldots M\}$ for both discriminative and generative model types.

Problem 6.3 Describe a regression model that relates univariate binary discrete data $x \in \{0,1\}$ to a univariate continuous world state $w \in [-\infty, \infty]$. Use a generative formulation in which $Pr(x|w)$ and $Pr(w)$ are modeled.

Problem 6.4 Describe a discriminative regression model that relates a continuous world state $w \in [0,1]$ to univariate continuous data $x \in [-\infty, \infty]$. Hint: Base your classifier on the Beta distribution. Ensure that the constraints on the parameters are obeyed.

Problem 6.5 Find expressions for the maximum likelihood estimates of the parameters in the discriminative linear regression model (section 6.3.1). In other words find the parameters $\{\phi_0, \phi_1, \sigma^2\}$ that satisfy

$$\begin{aligned}\hat{\phi}_0, \hat{\phi}_1, \hat{\sigma}^2 &= \underset{\phi_0,\phi_1,\sigma^2}{\text{argmax}}\left[\prod_{i=1}^{I} Pr(w_i|x_i, \phi_0, \phi_1, \sigma^2)\right] \\ &= \underset{\phi_0,\phi_1,\sigma^2}{\text{argmax}}\left[\sum_{i=1}^{I} \log\left[Pr(w_i|x_i, \phi_0, \phi_1, \sigma^2)\right]\right] \\ &= \underset{\phi_0,\phi_1,\sigma^2}{\text{argmax}}\left[\sum_{i=1}^{I} \log\left[\text{Norm}_w\left[\phi_0 + \phi_1 x, \sigma^2\right]\right]\right],\end{aligned}$$

where $\{w_i, x_i\}_{i=1}^{I}$ are paired training examples.

Problem 6.6 Consider a regression model that models the joint probability $Pr(x, w)$ between the world w and the data x as

$$Pr\left(\begin{bmatrix} w_i \\ x_i \end{bmatrix}\right) = \text{Norm}_{[w_i, x_i]^T}\left[\begin{bmatrix} \mu_w \\ \mu_x \end{bmatrix}, \begin{bmatrix} \sigma_{ww}^2 & \sigma_{xw}^2 \\ \sigma_{xw}^2 & \sigma_{xx} \end{bmatrix}\right].$$

Use the relation in section 5.5 to compute the posterior distribution $Pr(w_i|x_i)$. Show that it has the form

$$Pr(w_i|x_i) = \text{Norm}_{w_i}[\phi_0 + \phi_1 x, \sigma^2],$$

and compute expressions for ϕ_0 and ϕ_1 in terms of the training data $\{w_i, x_i\}_{i=1}^{I}$ by substituting in explicit maximum likelihood estimates of the parameters $\{\mu_w, \mu_x, \sigma_{ww}^2, \sigma_{xw}^2, \sigma_{xx}^2\}$.

Problem 6.7 For a two-class problem, the *decision boundary* is the locus of world values w where the posterior probability $Pr(w = 1|x)$ is equal to 0.5. In other words, it represents the boundary between regions that would be classified as $w = 0$ and $w = 1$. Consider the generative classifier from section 6.4.2. Show that with equal priors $Pr(w = 0) = Pr(w = 1) = 0.5$ points on the decision boundary (the locus of points where $Pr(w = 0|x) = Pr(w = 1|x)$) obey a constraint of the form

$$ax^2 + bx + c = 0,$$

where and $\{a, b, c\}$ are scalars. Does the shape of the decision boundary for the logistic regression model from section 6.4.1 have the same form?

Problem 6.8 Consider a generative classification model for 1D data with likelihood terms

$$\begin{aligned}Pr(x_i|w_i = 0) &= \text{Norm}_{x_i}\left[0, \sigma^2\right] \\ Pr(x_i|w_i = 1) &= \text{Norm}_{x_i}\left[0, 1.5\sigma^2\right].\end{aligned}$$

What is the decision boundary for this classifier with equal priors $Pr(w=0) = Pr(w=1) = 0.5$? Develop a discriminative classifier that can produce the same decision boundary. Hint: base your classifier on a quadratic rather than a linear function.

Problem 6.9 Consider a generative binary classifier for multivariate data based on multivariate normal likelihood terms

$$\begin{aligned} Pr(\mathbf{x}_i|w_i=0) &= \text{Norm}_{\mathbf{x}_i}\left[\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0\right] \\ Pr(\mathbf{x}_i|w_i=1) &= \text{Norm}_{\mathbf{x}_i}\left[\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1\right] \end{aligned}$$

and a discriminative classifier based on logistic regression for the same data

$$Pr(w_i|\mathbf{x}_i) = \text{Bern}_{w_i}\left[\text{sig}[\phi_0 + \boldsymbol{\phi}^T\mathbf{x}_i]\right].$$

where there is one entry in the gradient vector $\boldsymbol{\phi}$ for each entry of $\mathbf{x}_i$.

How many parameters does each model have as a function of the dimensionality of $\mathbf{x}_i$? What are the relative advantages and disadvantages of each model as the dimensionality increases?

Problem 6.10 One of the problems with the background subtraction method described is that it erroneously classifies shadows as foreground. Describe a model that could be used to classify pixels into three categories (foreground, background, and shadow).

Chapter 7

Modeling complex data densities

In the last chapter we showed that classification with generative models is based on building simple probability models. In particular, we build class-conditional probability distributions $Pr(\mathbf{x}|w = k)$ over the observed data $\mathbf{x}$ for each value of the world state w.

In chapter 3 we introduced several probability distributions that could be used for this purpose but these were quite limited in scope. For example, it is not realistic to assume that all of the complexities of visual data are well described by the normal distribution. In this chapter, we show how to construct complex probability density functions from elementary ones using the idea of a *hidden variable*.

As a representative problem we consider *face detection*; we observe a 60×60 RGB image patch and we would like to decide whether it contains a face or not. To this end, we concatenate the RGB values to form the 10800×1 vector $\mathbf{x}$. Our goal is to take the vector $\mathbf{x}$ and return a label $w \in \{0, 1\}$ indicating whether it contains background ($w = 0$) or a face ($w = 1$). In a real face detection system we would repeat this procedure for every possible sub-window of an image (figure 7.1).

We will start with a basic generative approach in which we describe the likelihood of the data in the presence/absence of a face with a normal distribution. We will then extend this model to address its weaknesses. We emphasize though that state-of-the-art face detection algorithms are *not* based on generative methods such as these; they are usually tackled using the discriminative methods of chapter 9. This application was selected for purely pedagogical reasons.

7.1 Normal classification model

We will take a generative approach to face detection; we will model the probability of the data $\mathbf{x}$ and parameterize this by the world state w. We will describe the data with a multivariate normal distribution so that

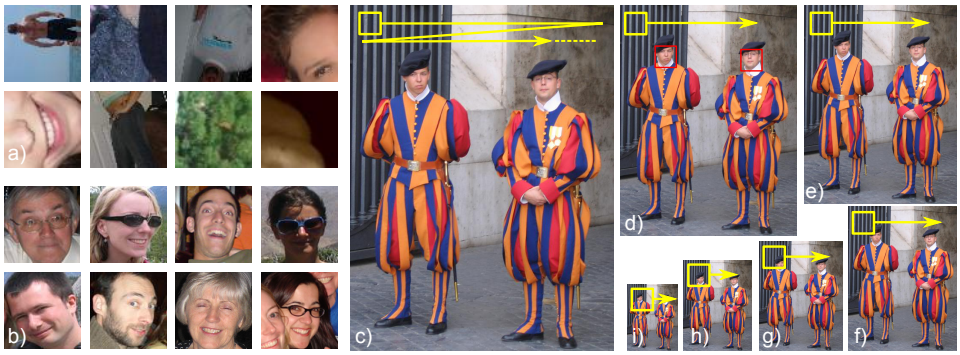


Figure 7.1 Face detection. Consider examining a small window of the image (here 60×60). We concatenate the RGB values in the window to make a data vector $\mathbf{x}$ of dimension 10800×1. The goal of face detection is to infer a label $w \in \{0, 1\}$ indicating whether the window contains (a) a background region ($w=0$) or (b) an aligned face ($w=1$). (c-i) We repeat this operation at every position and scale in the image by sweeping a fixed size window through a stack of resized images, estimating w at every point.

$$Pr(\mathbf{x}|w) = \text{Norm}_{\mathbf{x}}[\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w] \tag{7.1}$$

or treating the two possible values of the state w separately, we can explicitly write

$$\begin{aligned} Pr(\mathbf{x}|w=0) &= \text{Norm}_{\mathbf{x}}[\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0] \\ Pr(\mathbf{x}|w=1) &= \text{Norm}_{\mathbf{x}}[\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1]. \end{aligned} \tag{7.2}$$

These expressions are examples of *class conditional density functions*. They describe the density of the data $\mathbf{x}$ conditional on the value of the world state w.

The goal of learning is to estimate the parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1\}$ from example pairs of training data $\{\mathbf{x}_i, w_i\}_{i=1}^I$. Since parameters $\boldsymbol{\mu}_0$ and $\boldsymbol{\Sigma}_0$ are concerned exclusively with background regions (where $w=0$) we can learn them from the subset of training data $\mathcal{S}_0$ that belonged to the background. For example, using the maximum likelihood approach, we would seek

$$\begin{aligned} \hat{\boldsymbol{\mu}}_0, \hat{\boldsymbol{\Sigma}}_0 &= \underset{\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0}{\text{argmax}} \left[\prod_{i \in \mathcal{S}_0} Pr(\mathbf{x}_i|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \right] \\ &= \underset{\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0}{\text{argmax}} \left[\prod_{i \in \mathcal{S}_0} \text{Norm}_{\mathbf{x}_i}[\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0] \right]. \end{aligned} \tag{7.3}$$

Similarly, $\boldsymbol{\mu}_1$ and $\boldsymbol{\Sigma}_1$ are concerned exclusively with faces (where $w = 1$) and can be learned from the subset $\mathcal{S}_1$ of training data which contained faces. Figure 7.2

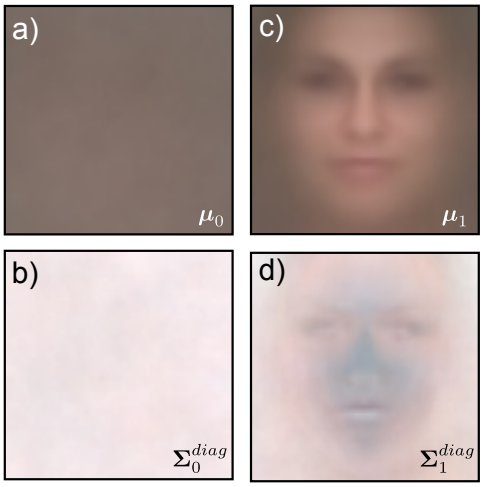


Figure 7.2 Class conditional density functions for normal model with diagonal covariance. Maximum likelihood fits based on 1000 training examples per class. a) Mean for background data $\boldsymbol{\mu}_0$ (reshaped from 10800×1 vector to 60×60 RGB image). b) Reshaped square root of diagonal covariance for background data $\boldsymbol{\Sigma}_0$. c) Mean for face data $\boldsymbol{\mu}_1$ d) Covariance for face data $\boldsymbol{\Sigma}_1$. The background model has little structure: the mean is uniform and the variance is high everywhere. The mean of the face model clearly captures class-specific information. The covariance of the face is larger at the edges of the image, which usually contain hair or background.

shows the maximum likelihood estimates of the parameters where we have used the diagonal form of the covariance matrix.

The goal of the inference algorithm is to take a new facial image $\mathbf{x}$ and assign a label w to it. To this end, we define a prior over the values of the world state $Pr(w) = \text{Bern}_w[\lambda]$ and apply Bayes' rule

$$Pr(w=1|\mathbf{x}) = \frac{Pr(\mathbf{x}|w=1)Pr(w=1)}{\sum_{k=0}^{1} Pr(\mathbf{x}|w=k)Pr(w=k)}. \tag{7.4}$$

All of these terms are simple to compute and so inference is very easy and will not be discussed further in this chapter.

7.1.1 Deficiencies of the multivariate normal model

Unfortunately, this model does not detect faces reliably. We will defer presenting experimental results until section 7.9.1, but for now please take it on trust that while this model achieves above-chance performance, it doesn't come close to producing a state-of-the-art result. This is hardly surprising: the success of this classifier hinges on fitting the data with a normal distribution. Unfortunately, this fit is poor for three reasons (figure 7.3).

- The normal distribution is unimodal; neither faces nor background regions are well represented by a pdf with a single peak.
- The normal distribution is not robust; a single outlier can dramatically affect the estimates of the mean and covariance.
- The normal distribution has too many parameters; here the data have $D = 10800$ dimensions. The full covariance matrix contains $D(D+1)/2$ parameters. With only 1000 training examples, we cannot even specify these parameters uniquely, so we were forced to use the diagonal form.

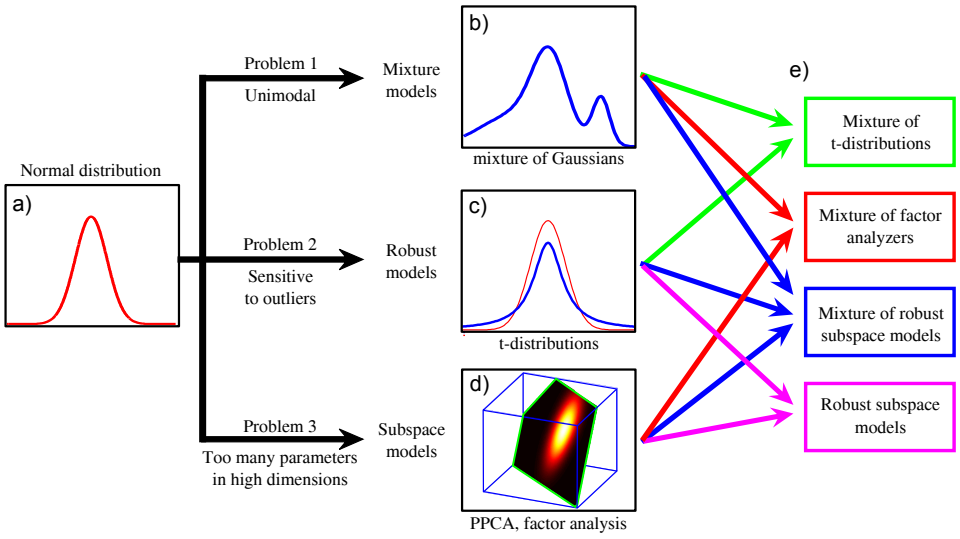


Figure 7.3 a) Problems with the multivariate normal density. b) Normal models are unimodal, but mixtures of Gaussians can model multi-modal distributions. c) Normal distributions are not robust to outliers, but t-distributions can cope with unusual observations. d) Normal models need many parameters in high dimensions but subspace models reduce this requirement. e) These solutions can be combined to form hybrid models addressing several of these problems at once.

We devote the rest of this chapter to tackling these problems. To make the density multi-modal, we introduce *mixture models*. To make the density robust, we replace the normal with the *t-distribution*. To cope with parameter estimation in high dimensions, we introduce *subspace models*.

The new models have much in common with each other. In each case we introduce a *hidden* or *latent variable* $\mathbf{h}_i$ associated with each observed data point $\mathbf{x}_i$. The hidden variable induces the more complex properties of the resulting pdf. Moreover, because the structure of the models is similar, we can use a common approach to learn the parameters.

In the following section, we present an abstract discussion of how hidden variables can be used to model complex pdfs. In section 7.3 we discuss how to learn the parameters of models with hidden variables. Then in sections 7.4, 7.5, and 7.6 we will introduce mixture models, t-distributions, and factor analysis, respectively.

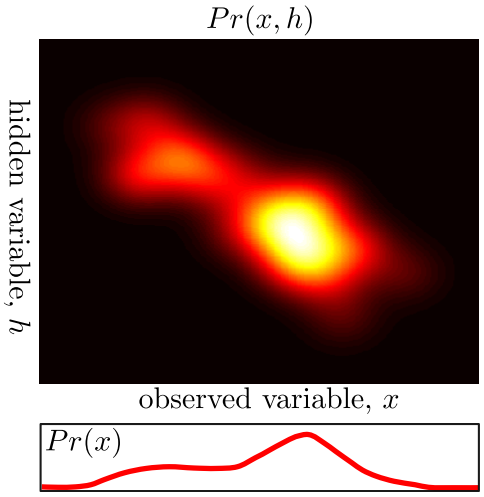


Figure 7.4 Using hidden variables to help model complex densities. One way to model the density $Pr(x)$ is to consider the joint probability distribution $Pr(x,h)$ between the observed data x and a hidden variable h. The density $Pr(x)$ can be considered as the marginalization of (integral over) this distribution with respect to the hidden variable h. As we manipulate the parameters $\boldsymbol{\theta}$ of this joint distribution, the marginal changes and the agreements with the observed data $\{x_i\}_{i=1}^{I}$ increases or decreases. Sometimes it is easier to fit the distribution in this indirect way than to directly manipulate $Pr(x)$.

7.2 Hidden variables

To model a complex probability density function over the variable $\mathbf{x}$, we will introduce a *hidden* or *latent* variable $\mathbf{h}$, which may be discrete or continuous. We will discuss the continuous formulation, but all of the important concepts transfer to the discrete case.

To exploit the hidden variables, we describe the final density $Pr(\mathbf{x})$ as the marginalization of the joint density $Pr(\mathbf{x},\mathbf{h})$ between $\mathbf{x}$ and $\mathbf{h}$ so that

$$Pr(\mathbf{x}) = \int Pr(\mathbf{x},\mathbf{h})\, d\mathbf{h}. \tag{7.5}$$

We now concentrate on describing the joint density $Pr(\mathbf{x},\mathbf{h})$. We can choose this so that it is relatively simple to model, but produces an expressive family of marginal distributions $Pr(\mathbf{x})$ when we integrate over $\mathbf{h}$ (see figure 7.4).

Whatever form we choose for the joint distribution, it will have some parameters $\boldsymbol{\theta}$, and so really we should write

$$Pr(\mathbf{x}|\boldsymbol{\theta}) = \int Pr(\mathbf{x},\mathbf{h}|\boldsymbol{\theta})\, d\mathbf{h}. \tag{7.6}$$

There are two possible approaches to fitting the model to training data $\{\mathbf{x}_i\}_{i=1}^{I}$ using the maximum likelihood method. We could directly maximize the log likelihood of the distribution $Pr(\mathbf{x})$ from the left hand side of equation 7.6 so that

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left[\sum_{i=1}^{I} \log\left[Pr(\mathbf{x}_i|\boldsymbol{\theta})\right] \right]. \tag{7.7}$$

This formulation has the advantage that we don't need to involve the hidden variables at all. However, in the models that we will consider, it will not result in a

neat closed form solution. Of course we could apply a brute force nonlinear optimization technique (appendix B) but there is an alternative approach: we use the *expectation maximization* algorithm, which works directly with the right-hand side of equation 7.6 and seeks

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left[\sum_{i=1}^{I} \log \left[\int Pr(\mathbf{x}_i, \mathbf{h}_i|\boldsymbol{\theta}) \, d\mathbf{h}_i \right] \right]. \tag{7.8}$$

7.3 Expectation maximization

In this section, we will present a brief description of the *expectation maximization (EM)* algorithm. The goal is to provide just enough information to use this technique for fitting models. We will return to a more detailed treatment in section 7.8.

The EM algorithm is a general-purpose tool for fitting parameters $\boldsymbol{\theta}$ in models of the form of equation 7.6 where

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left[\sum_{i=1}^{I} \log \left[\int Pr(\mathbf{x}_i, \mathbf{h}_i|\boldsymbol{\theta}) \, d\mathbf{h}_i \right] \right]. \tag{7.9}$$

The EM algorithm works by defining a lower bound $\mathcal{B}[\{q_i(\mathbf{h}_i)\}, \boldsymbol{\theta}]$ on the log likelihood in equation 7.9 and iteratively increasing this bound. The lower bound is simply a function that is parameterized by $\boldsymbol{\theta}$ and some other quantities and is guaranteed to always return a value that is less than or equal to the log likelihood $L[\boldsymbol{\theta}]$ for any given set of parameters $\boldsymbol{\theta}$ (figure 7.5).

For the EM algorithm, the particular lower bound chosen is

$$\begin{aligned} \mathcal{B}[\{q_i(\mathbf{h}_i)\}, \boldsymbol{\theta}] &= \sum_{i=1}^{I} \int q_i(\mathbf{h}_i) \log \left[\frac{Pr(\mathbf{x}_i, \mathbf{h}_i|\boldsymbol{\theta})}{q_i(\mathbf{h}_i)} \right] d\mathbf{h}_i \\ &\leq \sum_{i=1}^{I} \log \left[\int Pr(\mathbf{x}_i, \mathbf{h}_i|\boldsymbol{\theta}) \, d\mathbf{h}_i \right]. \end{aligned} \tag{7.10}$$

It is not obvious that this inequality is true, making this a valid lower bound; take this on trust for the moment and we will return to this in section 7.8.

In addition to the parameters $\boldsymbol{\theta}$, the lower bound $\mathcal{B}[\{q_i(\mathbf{h}_i)\}, \boldsymbol{\theta}]$ also depends on a set of I probability distributions $\{q_i(\mathbf{h}_i)\}_{i=1}^{I}$ over the hidden variables $\{\mathbf{h}_i\}_{i=1}^{I}$. When we vary these probability distributions, the value that the lower bound returns will change, but it will always remain less than or equal to the log likelihood.

The EM algorithm manipulates both the parameters $\boldsymbol{\theta}$ and the distributions $\{q_i(\mathbf{h}_i)\}_{i=1}^{I}$ to increase the lower bound. It alternates between:

- updating the probability distributions $\{q_i(\mathbf{h}_i)\}_{i=1}^{I}$ to improve the bound in equation 7.10. This is called the *expectation step* or *E-step*, and
- updating the parameters $\boldsymbol{\theta}$ to improve the bound in equation 7.10. This is called the *maximization step* or *M-step*.

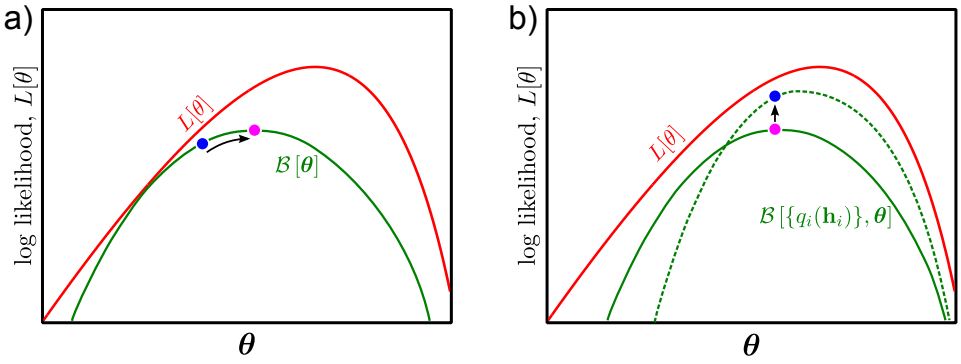


Figure 7.5 Manipulating the lower bound. a) Consider the log likelihood $L[\boldsymbol{\theta}]$ of the data $\{\mathbf{x}\}_{i=1}^{I}$ as a function of the model parameters $\boldsymbol{\theta}$ (red curve). In maximum likelihood learning our goal is to find the parameters $\boldsymbol{\theta}$ that maximize this function. A lower bound on the log likelihood is another function $\mathcal{B}[\boldsymbol{\theta}]$ of the parameters $\boldsymbol{\theta}$ that is everywhere lower or equal to the log likelihood (green curve). One way to improve the current estimate (blue dot) is to manipulate the parameters so that $\mathcal{B}[\boldsymbol{\theta}]$ increases (purple dot). This is the goal of the maximization step of the EM algorithm. b) The lower bound $\mathcal{B}[\{q_i(\mathbf{h}_i)\}, \boldsymbol{\theta}]$ also depends on a set of probability distributions $\{q_i(\mathbf{h}_i)\}_{i=1}^{I}$ over hidden variables $\{\mathbf{h}_i\}$. Manipulating these probability distributions changes the value that the lower bound returns for every $\boldsymbol{\theta}$ (e.g., green curve). So a second way to improve the current estimate (purple dot) is to change the distributions in such a way that the curve increases for the current parameters (blue dot). This is the goal of the expectation step of the EM algorithm.

In the E-step at iteration t+1 we set each distribution $q_i(\mathbf{h}_i)$ to be the posterior distributions $Pr(\mathbf{h}_i|\mathbf{x}_i, \boldsymbol{\theta})$ over that hidden variable given the associated data example and the current parameters $\boldsymbol{\theta}^{[t]}$. To compute these we use Bayes' rule

$$\hat{q}_i(\mathbf{h}_i) = Pr(\mathbf{h}_i|\mathbf{x}_i, \boldsymbol{\theta}^{[t]}) = \frac{Pr(\mathbf{x}_i|\mathbf{h}_i, \boldsymbol{\theta}^{[t]})Pr(\mathbf{h}_i|\boldsymbol{\theta}^{[t]})}{Pr(\mathbf{x}_i)}. \tag{7.11}$$

It can be shown that this choice maximizes the bound as much as possible.

In the M-step we directly maximize the bound (equation 7.10) with respect to the parameters $\boldsymbol{\theta}$. In practice we can simplify the expression for the bound to eliminate terms that do not depend on $\boldsymbol{\theta}$ and this yields

$$\hat{\boldsymbol{\theta}}^{[t+1]} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left[\sum_{i=1}^{I} \int \hat{q}_i(\mathbf{h}_i) \log \left[Pr(\mathbf{x}_i, \mathbf{h}_i|\boldsymbol{\theta}) \right] \, d\mathbf{h}_i \right]. \tag{7.12}$$

Each of these steps is guaranteed to improve the bound, and iterating them alternately is guaranteed to find at least a local maximum with respect to $\boldsymbol{\theta}$.

This is a practical description of the EM algorithm, but there is a lot missing: we have not demonstrated that equation 7.10 really is a bound on the log likelihood.

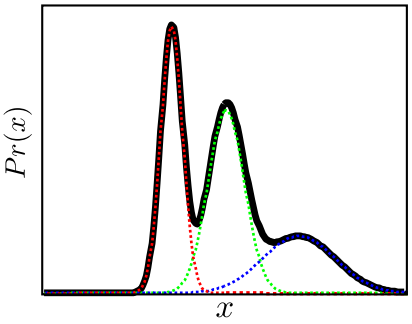


Figure 7.6 Mixture of Gaussians model in 1D. A complex multimodal probability density function (black solid curve) is created by taking a weighted sum or *mixture* of several constituent normal distributions with different means and variances (red, green and blue dashed curves). To ensure that the final distribution is a valid density, the weights must be positive and sum to one.

We have not shown that the posterior distribution $Pr(\mathbf{h}_i|\mathbf{x}_i, \boldsymbol{\theta}^{[t]})$ is the optimal choice for $q_i(\mathbf{h}_i)$ in the E-step (equation 7.11), and we have not demonstrated that the cost function for the M-step (equation 7.12) improves the bound. For now we will assume that these things are true and proceed with the main thrust of the chapter. We will return to these issues in section 7.8.

7.4 Mixture of Gaussians

The *mixture of Gaussians* (MoG) is a prototypical example of a model where learning is suited to the EM algorithm. The data are described as a weighted sum of K normal distributions

$$Pr(\mathbf{x}|\boldsymbol{\theta}) = \sum_{k=1}^{K} \lambda_k \text{Norm}_{\mathbf{x}}[\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k], \tag{7.13}$$

where $\boldsymbol{\mu}_{1\ldots K}$ and $\boldsymbol{\Sigma}_{1\ldots K}$ are the means and covariances of the normal distributions and $\lambda_{1\ldots K}$ are positive valued weights that sum to one. The mixtures of Gaussians model describes complex multi-modal probability densities by combining simpler constituent distributions (figure 7.6).

To learn the parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \lambda_k\}_{k=1}^K$ from training data $\{\mathbf{x}_i\}_{i=1}^I$ we could apply the straightforward maximum likelihood approach

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}}{\text{argmax}} \left[\sum_{i=1}^{I} \log\left[Pr(\mathbf{x}_i|\boldsymbol{\theta})\right]\right] \\ &= \underset{\boldsymbol{\theta}}{\text{argmax}} \left[\sum_{i=1}^{I} \log\left[\sum_{k=1}^{K} \lambda_k \text{Norm}_{\mathbf{x}_i}[\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k]\right]\right].\end{aligned} \tag{7.14}$$

Unfortunately, if we take the derivative with respect to the parameters $\boldsymbol{\theta}$ and equate the resulting expression to zero, it is not possible to solve the resulting equations in closed form. The sticking point is the summation inside the logarithm, which precludes a simple solution. Of course, we could use a nonlinear optimization approach, but this would be complex as we would have to maintain the constraints

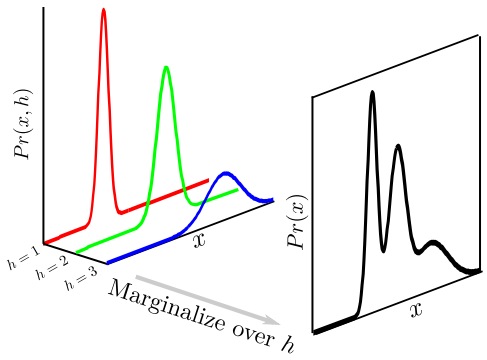


Figure 7.7 Mixture of Gaussians as a marginalization. The mixture of Gaussians can also be thought of in terms of a joint distribution $Pr(x,h)$ between the observed variable x and a discrete hidden variable h. To create the mixture density we marginalize over h. The hidden variable has a straightforward interpretation: it is the index of the constituent normal distribution.

on the parameters; the weights $\boldsymbol{\lambda}$ must sum to one and the covariances $\{\boldsymbol{\Sigma}_k\}_{k=1}^K$ must be positive definite. For a simpler approach, we express the observed density as a marginalization and use the EM algorithm to learn the parameters.

7.4.1 Mixture of Gaussians as a marginalization

The mixture of Gaussians model can be expressed as the marginalization of a joint probability distribution between the observed data $\mathbf{x}$ and a discrete hidden variable h that takes values $h \in \{1 \ldots K\}$ (figure 7.7). If we define

$$\begin{aligned} Pr(\mathbf{x}|h,\boldsymbol{\theta}) &= \text{Norm}_{\mathbf{x}}[\boldsymbol{\mu}_h, \boldsymbol{\Sigma}_h] \\ Pr(h|\boldsymbol{\theta}) &= \text{Cat}_h[\boldsymbol{\lambda}], \end{aligned} \tag{7.15}$$

where $\boldsymbol{\lambda} = [\lambda_1 \ldots \lambda_K]$ are the parameters of the categorical distribution, then we can recover the original density using

$$\begin{aligned} Pr(\mathbf{x}|\boldsymbol{\theta}) &= \sum_{k=1}^{K} Pr(\mathbf{x}, h=k|\boldsymbol{\theta}) \\ &= \sum_{k=1}^{K} Pr(\mathbf{x}|h=k,\boldsymbol{\theta}) Pr(h=k|\boldsymbol{\theta}) \\ &= \sum_{k=1}^{K} \lambda_k \text{Norm}_{\mathbf{x}}[\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k]. \end{aligned} \tag{7.16}$$

Interpreting the model in this way also provides a method to draw samples from a mixture of Gaussians: we sample from the joint distribution $Pr(\mathbf{x}, h)$, and then discard the hidden variable h to leave just a data sample $\mathbf{x}$. To sample from the joint distribution $Pr(\mathbf{x}, h)$ we first sample h from the categorical prior $Pr(h)$, then sample $\mathbf{x}$ from the normal distribution $Pr(\mathbf{x}|h)$ associated with the value of h. Notice that the hidden variable h has a clear interpretation in this procedure; it determines which of the constituent normal distributions is *responsible* for the observed data point $\mathbf{x}$.

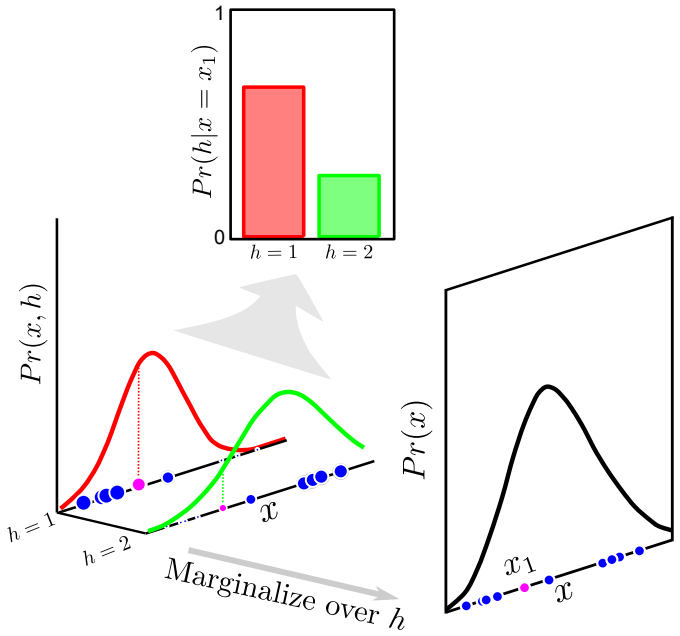


Figure 7.8 E-step for fitting the mixture of Gaussians model. For each of the I data points $\mathbf{x}_{1...I}$, we calculate the posterior distribution $Pr(h_i|\mathbf{x}_i)$ over the hidden variable h_i. The posterior probability $Pr(h_i = k|\mathbf{x}_i)$ that h_i takes value k can be understood as the responsibility of normal distribution k for data point x_i. For example, for data point x_1 (magenta circle), component 1 (red curve) is more than twice as likely to be responsible than component 2 (green curve). Note that in the joint distribution (left), the size of the projected data point indicates the responsibility.

7.4.2 Expectation maximization for fitting mixture models

Algorithm 7.1

To learn the MoG parameters $\boldsymbol{\theta} = \{\lambda_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$ from training data $\{\mathbf{x}_i\}_{i=1}^I$ we apply the EM algorithm. Following the recipe of section 7.3, we initialize the parameters randomly and alternate between performing the E- and M-steps.

In the E-step, we maximize the bound with respect to the distributions $q_i(h_i)$ by finding the posterior probability distribution $Pr(h_i|\mathbf{x}_i)$ of each hidden variable h_i given the observation $\mathbf{x}_i$ and the current parameter settings,

$$\begin{aligned} q_i(h_i) = Pr(h_i = k|\mathbf{x}_i, \boldsymbol{\theta}^{[t]}) &= \frac{Pr(\mathbf{x}_i|h_i = k, \boldsymbol{\theta}^{[t]})Pr(h_i = k, \boldsymbol{\theta}^{[t]})}{\sum_{j=1}^K Pr(\mathbf{x}_i|h_i = j, \boldsymbol{\theta}^{[t]})Pr(h_i = j, \boldsymbol{\theta}^{[t]})} \\ &= \frac{\lambda_k \text{Norm}_{\mathbf{x}_i}[\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k]}{\sum_{j=1}^K \lambda_j \text{Norm}_{\mathbf{x}_i}[\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j]} \\ &= r_{ik}. \end{aligned} \tag{7.17}$$

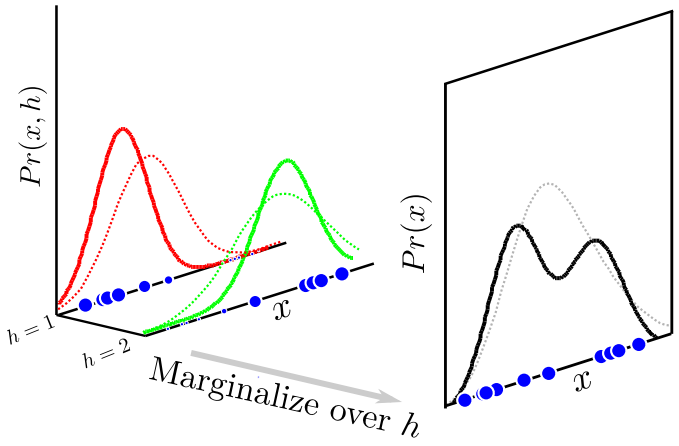


Figure 7.9 M-step for fitting the mixture of Gaussians model. For the k^{th} constituent Gaussian, we update the parameters $\{\boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$. The i^{th} data point $\mathbf{x}_i$ contributes to these updates according to the responsibility r_{ik} (indicated by size of point) assigned in the E-step; data points that are more associated with the k^{th} component have more effect on the parameters. Dashed and solid lines represent fit before and after update, respectively.

In other words we compute the probability $Pr(h_i = k|\mathbf{x}_i, \boldsymbol{\theta}^{[t]})$ that the k^{th} normal distribution was responsible for the i^{th} data point (figure 7.8). We denote this *responsibility* by r_{ik} for short.

In the M-step, we maximize the bound with respect to the parameters $\boldsymbol{\theta} = \{\lambda_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$ so that

$$\begin{aligned}\hat{\boldsymbol{\theta}}^{[t+1]} &= \underset{\boldsymbol{\theta}}{\text{argmax}} \left[\sum_{i=1}^{I}\sum_{k=1}^{K} \hat{q}_i(h_i = k)\log\left[Pr(\mathbf{x}_i, h_i = k|\boldsymbol{\theta})\right]\right] \\ &= \underset{\boldsymbol{\theta}}{\text{argmax}} \left[\sum_{i=1}^{I}\sum_{k=1}^{K} r_{ik}\log\left[\lambda_k \text{Norm}_{\mathbf{x}_i}[\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k]\right]\right]. \qquad (7.18)\end{aligned}$$

This maximization can be performed by taking the derivative of the expression with respect to the parameters, equating the result to zero and rearranging, taking care to enforce the constraint $\sum_k \lambda_k = 1$ using Lagrange multipliers. The procedure results in the update rules:

Problem 7.3

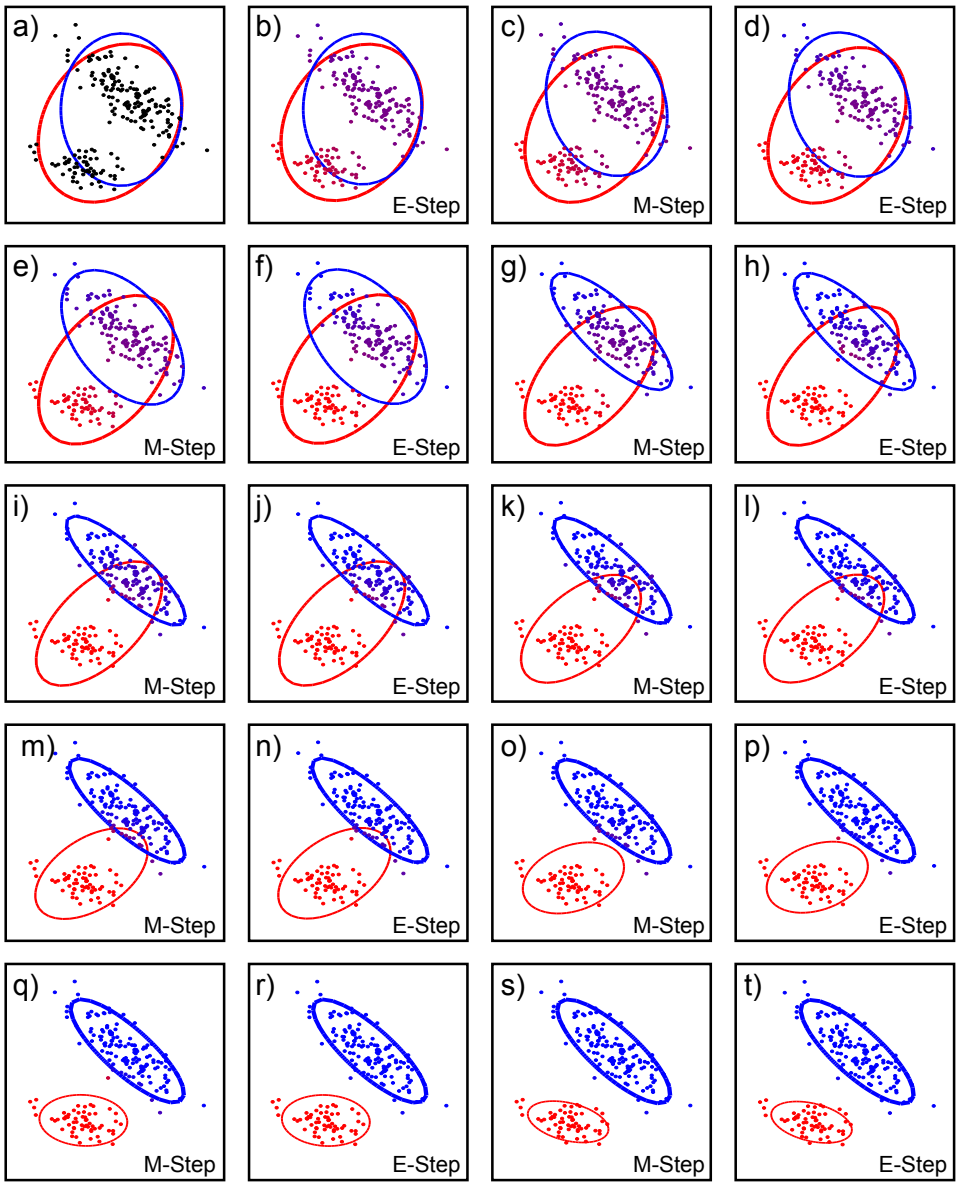


Figure 7.10 a) Initial model. b) E-step. For each data point the posterior probability that is was generated from each Gaussian is calculated (indicated by color of point). c) M-step. The mean, variance and weight of each Gaussian is updated based on these posterior probabilities. Ellipse shows Mahalanobis distance of two. Weight (thickness) of ellipse indicates weight of Gaussian. d-t) Further E-step and M-step iterations.

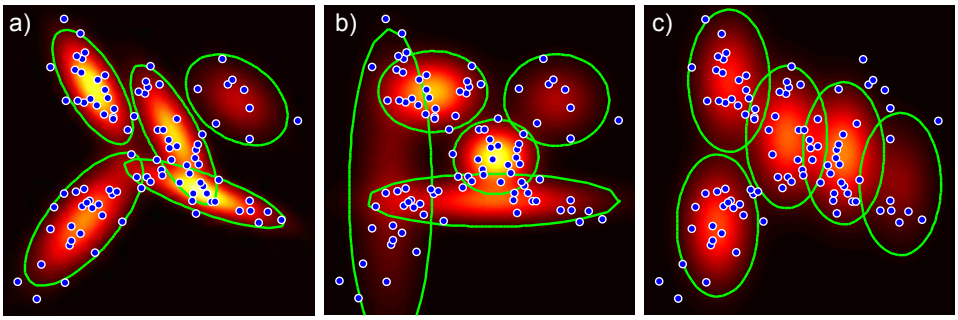


Figure 7.11 Covariance of components in mixture models. a) Full covariances. b) Diagonal covariances. c) Identical diagonal covariances.

$$\begin{aligned}
\lambda_k^{[t+1]} &= \frac{\sum_{i=1}^{I} r_{ik}}{\sum_{j=1}^{K}\sum_{i=1}^{I} r_{ij}} \\
\boldsymbol{\mu}_k^{[t+1]} &= \frac{\sum_{i=1}^{I} r_{ik}\mathbf{x}_i}{\sum_{i=1}^{I} r_{ik}} \\
\boldsymbol{\Sigma}_k^{[t+1]} &= \frac{\sum_{i=1}^{I} r_{ik}(\mathbf{x}_i - \boldsymbol{\mu}_k^{[t+1]})(\mathbf{x}_i - \boldsymbol{\mu}_k^{[t+1]})^T}{\sum_{i=1}^{I} r_{ik}}.
\end{aligned} \tag{7.19}$$

These update rules can be easily understood (figure 7.9): we update the weights $\{\lambda_k\}_{k=1}^K$ according to the relative total responsibility of each component for the data points. We update the cluster means $\{\boldsymbol{\mu}_k\}_{k=1}^K$ by computing the weighted mean over the data points where the weights are given by the responsibilities. If component k is mostly responsible for data point $\mathbf{x}_i$, then this data point has a high weight and affects the update more. The update rule for the covariances has a similar interpretation.

In practice the E- and M-steps are alternated until the bound on the data no longer increases and the parameters no longer change. The alternating E-steps and M-steps for a two dimensional example are shown in figure 7.10. Notice that the final fit identifies the two *clusters* in the data. The mixture of Gaussians is closely related to clustering techniques such as the *K-means* algorithm (section 13.4.4).

The EM approach to estimating mixture models has three attractive features.

1. Both steps of the algorithm can be computed in closed form without the need for an optimization procedure.
2. The solution guarantees that the constraints on the parameters are respected: the weighting parameters $\{\lambda_k\}_{k=1}^K$ are guaranteed to be positive and sum to one, and the covariance matrices $\{\boldsymbol{\Sigma}_k\}_{k=1}^K$ are guaranteed to be positive definite.

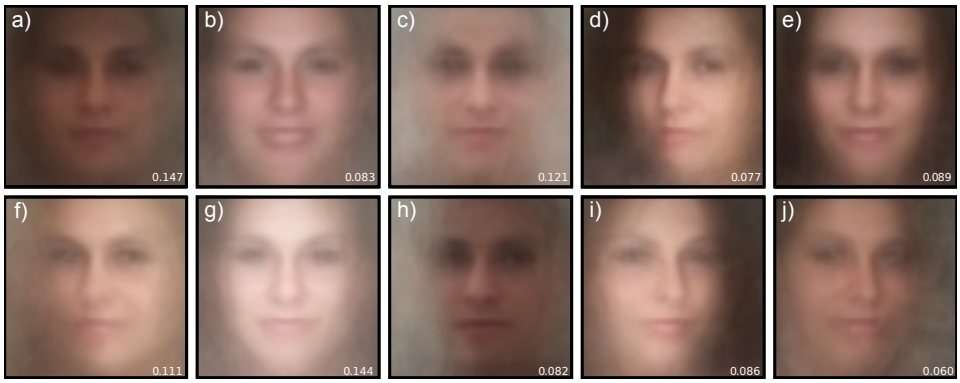


Figure 7.12 Mixtures of Gaussians model for face data. a-j) Mean vectors $\boldsymbol{\mu}_k$ for a mixture of ten Gaussians fitted to the face data set. The model has captured variation in the mean luminance and chromaticity of the face and other factors such as the pose and background color. Numbers indicate the weight of each component.

3. The method can cope with missing data. Imagine that some of the elements of training example $\mathbf{x}_i$ are missing. In the E-step, the remaining dimensions can still be used to establish a distribution over the hidden variable h. In the M-step, this data point would contribute only to the dimensions of $\{\boldsymbol{\mu}_k\}_{k=1}^K$ and $\{\boldsymbol{\Sigma}_k\}_{k=1}^K$where data were observed.

Figure 7.11 shows a mixture of five Gaussians that has been fit to a 2D data set. As for the basic multivariate normal model, it is possible to constrain the covariance matrices to be spherical or diagonal. We can also constrain the covariances to be the same for each component if desired. Figure 7.12 shows the mean vectors $\boldsymbol{\mu}_k$ for a ten component model with diagonal covariances fitted to the face data set. The clusters represent different illumination conditions as well as changes in pose, expression, and background color.

In fitting mixtures of Gaussians there are several things to consider. First, the EM algorithm does not guarantee to find a global solution to this non-convex optimization problem. Figure 7.13 shows three different solutions that were computed by starting the fitting algorithm with different initial random values for the parameters $\boldsymbol{\theta}$. The best we can do to circumvent this problem is to start fitting in different places and take the solution with the greatest log likelihood. Second, we must pre-specify the number of mixing components. Unfortunately, we cannot decide the number of components by comparing the log likelihood; models with more parameters will inevitably describe the data better. There are methods to tackle this problem, but they are beyond the scope of this volume.

Finally, although we presented a maximum likelihood approach here, it is important in practice to include priors over model parameters $Pr(\boldsymbol{\theta})$ to prevent the scenario where one of the Gaussians becomes exclusively associated with a single data point. Without a prior, the variance of this component becomes progressively

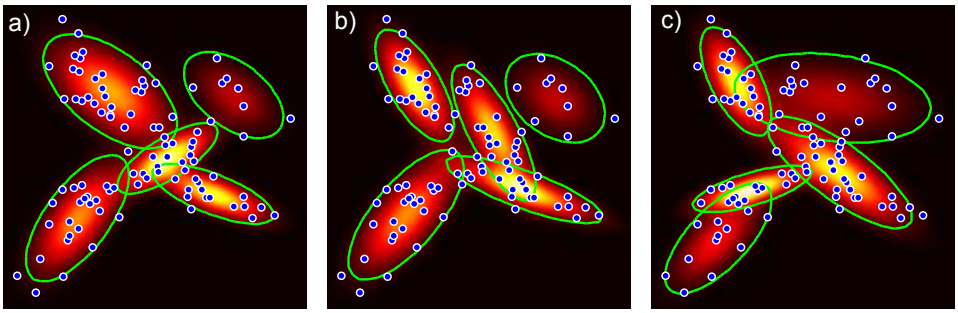


Figure 7.13 Local maxima. Repeated fitting of mixture of Gaussians model with different starting points results in different models as the fit converges to different local maxima. The log likelihoods are a) 98.76 b) 96.97 c) 94.35, respectively, indicating that (a) is the best fit.

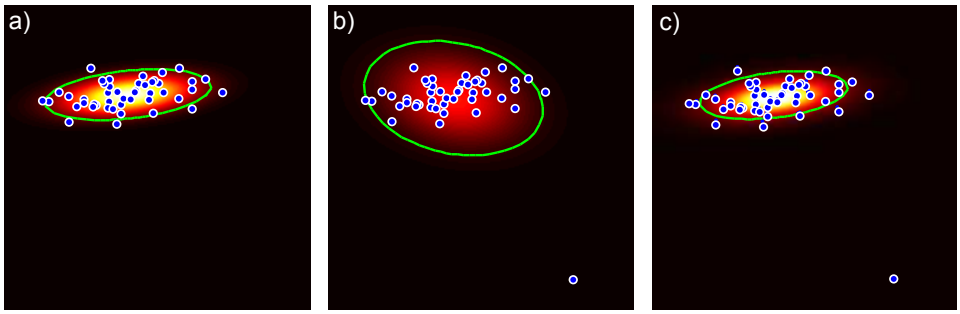


Figure 7.14 Motivation for t-distribution. a) The multivariate normal model fit to data. b) Adding a single outlier completely changes the fit. c) With the multivariate t-distribution the outlier does not have such a drastic effect.

smaller and the likelihood increases without bound.

7.5 The t-distribution

The second significant problem with using the normal distribution to describe visual data is that it is not robust: the height of the normal pdf falls off very rapidly as we move into the tails. The effect of this is that outliers (unusually extreme observations) drastically affect the estimated parameters (figure 7.14). The t-distribution is a closely related distribution in which the length of the tails is parameterized.

The univariate t-distribution (figure 7.15) has probability density function

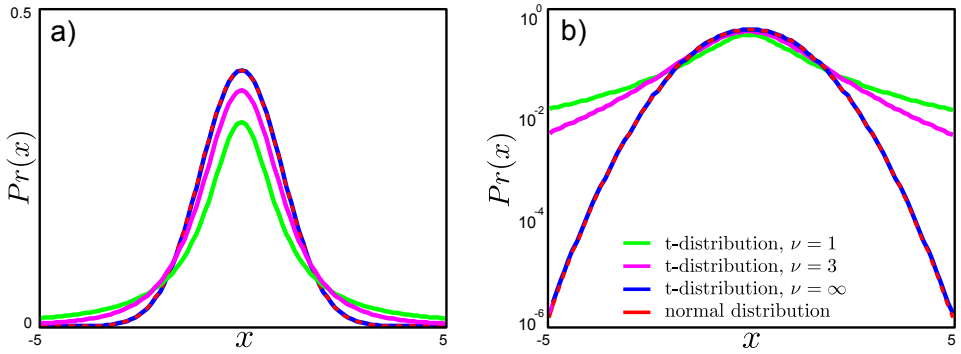


Figure 7.15 a) As well as the mean μ and scaling parameter σ^2, the t-distribution has a parameter ν which is termed the degrees of freedom. As ν decreases, the tails of the distribution become longer and the model becomes more robust. b) This is seen more clearly on a log scale.

$$\begin{aligned} Pr(x) &= \text{Stud}_{\mathbf{x}}\left[\mu, \sigma^2, \nu\right] \\ &= \frac{\Gamma\left[\frac{\nu+1}{2}\right]}{\sqrt{\nu\pi\sigma^2}\Gamma\left[\frac{\nu}{2}\right]}\left(1+\frac{(x-\mu)^2}{\nu\sigma^2}\right)^{-\frac{\nu+1}{2}}, \end{aligned} \tag{7.20}$$

where μ is the mean and σ^2 is the scale parameter. The degrees of freedom $\nu \in (0, \infty]$ controls the length of the tails: when ν is small there is considerable weight in the tails. For example, with $\mu = 0$ and $\sigma^2 = 1$ a data point at $x = -5$ is roughly $10^4 = 10000$ times more likely under the t-distribution with $\nu = 1$ than under the normal distribution. As ν tends to infinity, the distribution approximates a normal more and more closely and there is less weight in the tails. The variance of the distribution is given by $\sigma\nu/(\nu-2)$ for $\nu > 2$ and infinite if $0 < \nu \leq 2$.

The multivariate t-distribution has pdf

$$\begin{aligned} Pr(\mathbf{x}) &= \text{Stud}_{\mathbf{x}}\left[\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu\right] \\ &= \frac{\Gamma\left[\frac{\nu+D}{2}\right]}{(\nu\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}\Gamma\left[\frac{\nu}{2}\right]}\left(1+\frac{(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}{\nu}\right)^{-\frac{\nu+D}{2}}, \end{aligned} \tag{7.21}$$

where D is the dimensionality of the space, $\boldsymbol{\mu}$ is a $d\times 1$ mean vector, $\boldsymbol{\Sigma}$ is a $D\times D$ positive definite scale matrix, and $\nu \in [0, \infty]$ is the degrees of freedom. As for the multivariate normal distribution (figure 5.1), the scale matrix can take full, diagonal or spherical forms. The covariance of the distribution is given by $\boldsymbol{\Sigma}\nu/(\nu-2)$ for $\nu > 2$ and is infinite if $0 \leq \nu \leq 2$.

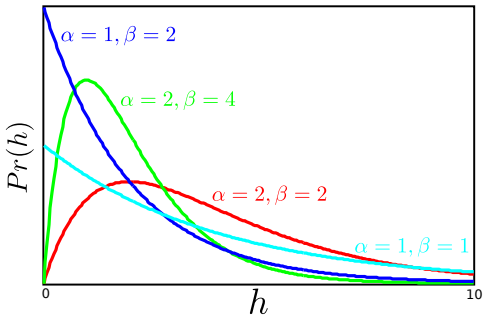


Figure 7.16 The gamma distribution is defined on positive real values and has two parameters α, β. The mean of the distribution is $\mathrm{E}[h] = \alpha/\beta$ and the variance is $\mathrm{E}[(h-\mathrm{E}[h])^2] = \alpha/\beta^2$. The t-distribution can be thought of as a weighted sum of normal distributions with the same mean, but covariances that depend inversely on the gamma distribution.

7.5.1 Student t-distribution as a marginalization

As for the mixtures of Gaussians, it is also possible to understand the t-distribution in terms of hidden variables. We define

Problem 7.5
Problem 7.6

$$\begin{aligned} Pr(\mathbf{x}|h) &= \mathrm{Norm}_x[\boldsymbol{\mu}, \boldsymbol{\Sigma}/h] \\ Pr(h) &= \mathrm{Gam}_h[\nu/2, \nu/2], \end{aligned} \tag{7.22}$$

where h is a scalar hidden variable and $\mathrm{Gam}[\alpha, \beta]$ is the gamma distribution with parameters α, β (figure 7.16). The gamma distribution is a continuous probability distribution defined on the positive real axis with probability density function

$$\mathrm{Gam}_h[\alpha, \beta] = \frac{\beta^\alpha}{\Gamma[\alpha]} \exp[-\beta h] h^{\alpha-1}, \tag{7.23}$$

where $\Gamma[\bullet]$ is the gamma function.

The t-distribution is the marginalization with respect to the hidden variable h of the joint distribution between the data $\mathbf{x}$ and h (figure 7.17),

$$\begin{aligned} Pr(\mathbf{x}) = \int Pr(\mathbf{x}, h) dh &= \int Pr(\mathbf{x}|h) Pr(h) dh \\ &= \int \mathrm{Norm}_x[\boldsymbol{\mu}, \boldsymbol{\Sigma}/h] \mathrm{Gam}_h[\nu/2, \nu/2] dh \\ &= \mathrm{Stud}_{\mathbf{x}}[\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu]. \end{aligned} \tag{7.24}$$

This formulation also provides a method to generate data from the t-distribution; we first generate h from the gamma distribution and then generate $\mathbf{x}$ from the associated normal distribution $Pr(\mathbf{x}|h)$. Hence the hidden variable has a simple interpretation: it tells us which one of the continuous family of underlying normal distributions was responsible for this data point.

7.5.2 Expectation maximization for fitting t-distributions

Since the pdf takes the form of a marginalization of the joint distribution with a hidden variable (equation 7.24), we can use the EM algorithm to learn the parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu\}$ from a set of training data $\{\mathbf{x}_i\}_{i=1}^I$.

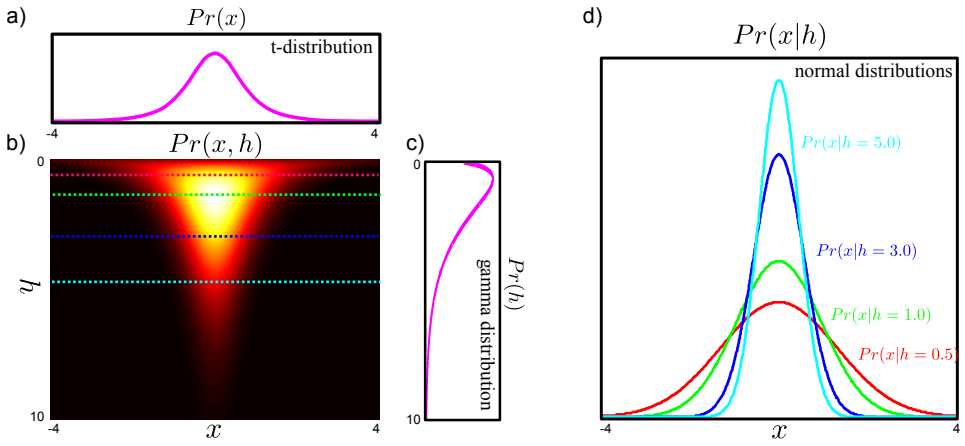


Figure 7.17 a) The t-distribution has a similar form to the normal distribution but longer tails. b) The t-distribution is the marginalization of the joint distribution $Pr(x,h)$ between the observed variable x and a hidden variable h. c) The prior distribution over the hidden variable h has a gamma distribution. d) The conditional distribution $Pr(x|h)$ is normal with a variance that depends on h. So the t-distribution can be considered as an infinite weighted sum of normal distributions with variances determined by the gamma prior (equation 7.24).

Algorithm 7.2

Problem 7.7

In the E-step (figure 7.18a-b) we maximize the bound with respect to the distributions $q_i(h_i)$ by finding the posterior $Pr(h_i|\mathbf{x}_i,\boldsymbol{\theta}^{[t]})$ over each hidden variable h_i given associated observation $\mathbf{x}_i$ and the current parameter settings. By Bayes' rule, we get

$$\begin{aligned} q_i(h_i) = Pr(h_i|\mathbf{x}_i,\boldsymbol{\theta}^{[t]}) &= \frac{Pr(\mathbf{x}_i|h_i,\boldsymbol{\theta}^{[t]})Pr(h_i)}{Pr(\mathbf{x}_i|\boldsymbol{\theta}^{[t]})} \\ &= \frac{\text{Norm}_{\mathbf{x}_i}[\boldsymbol{\mu},\boldsymbol{\Sigma}/h_i]\text{Gam}_{h_i}[\nu/2,\nu/2]}{Pr(\mathbf{x}_i)} \\ &= \text{Gam}_{h_i}\left[\frac{\nu+D}{2}, \frac{(\mathbf{x}_i-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}_i-\boldsymbol{\mu})}{2}+\frac{\nu}{2}\right], \end{aligned} \tag{7.25}$$

where we have used the fact that the gamma distribution is conjugate to the scaling factor for the normal variance. The E-step can be understood as follows: we are treating each data point $\mathbf{x}_i$ as if it were generated from one of the normals in the infinite mixture where the hidden variable h_i determines which normal. So, the E-step computes a distribution over h_i, which hence determines a distribution over which normal created the data.

We now compute the following expectations (section 2.7) with respect to the distribution in equation 7.25:

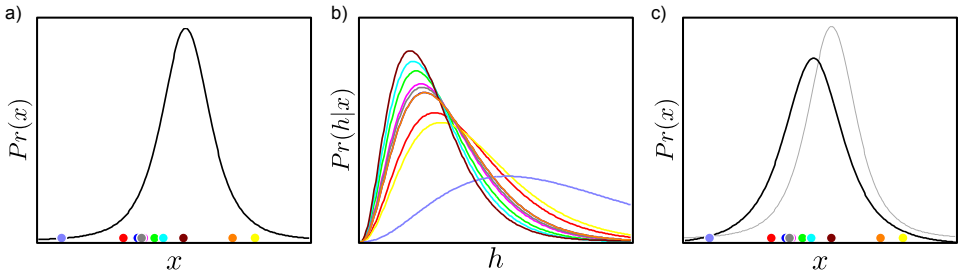


Figure 7.18 Expectation maximization for fitting t-distributions. a) Estimate of distribution before update. b) In the E-step we calculate the posterior distribution $Pr(h_i|x_i)$ over the hidden variable h_i for each data point x_i. The color of each curve corresponds to that of the original data point in (a). c) In the M-step we use these distributions over h to update the estimate of the parameters $\theta = \{\mu, \sigma^2, \nu\}$.

$$\begin{aligned} \mathrm{E}[h_i] &= \frac{(\nu + D)}{\nu + (\mathbf{x}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}_i - \boldsymbol{\mu})} && (7.26) \\ \mathrm{E}[\log[h_i]] &= \Psi\left[\frac{\nu + D}{2}\right] - \log\left[\frac{\nu + (\mathbf{x}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}_i - \boldsymbol{\mu})}{2}\right], && (7.27) \end{aligned}$$

where $\Psi[\bullet])$ is the *digamma* function. These will be needed in the E-step.

In the M-step (figure 7.18c) we maximize the bound with respect to the parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu\}$ so that

$$\begin{aligned} \hat{\boldsymbol{\theta}}^{[t+1]} &= \underset{\boldsymbol{\theta}}{\mathrm{argmax}} \left[\sum_{i=1}^{I} \int \hat{q}_i(h_i) \log\left[Pr(\mathbf{x}_i, h_i|\boldsymbol{\theta})\right] dh_i\right] \\ &= \underset{\boldsymbol{\theta}}{\mathrm{argmax}} \left[\sum_{i=1}^{I} \int \hat{q}_i(h_i) \left(\log\left[Pr(\mathbf{x}_i|h_i, \boldsymbol{\theta})\right] + \log\left[Pr(h_i)\right]\right) dh_i\right] \\ &= \underset{\boldsymbol{\theta}}{\mathrm{argmax}} \left[\sum_{i=1}^{I} \int Pr(h_i|\mathbf{x}_i, \boldsymbol{\theta}^{[t]}) \left(\log\left[Pr(\mathbf{x}_i|h_i, \boldsymbol{\theta})\right] + \log\left[Pr(h_i)\right]\right) dh_i\right] \\ &= \underset{\boldsymbol{\theta}}{\mathrm{argmax}} \left[\sum_{i=1}^{I} E\left[\log\left[Pr(\mathbf{x}_i|h_i, \boldsymbol{\theta})\right]\right] + E\left[\log\left[Pr(h_i)\right]\right]\right], \qquad (7.28) \end{aligned}$$

where the expectation is taken relative to the posterior distribution $Pr(h_i|\mathbf{x}_i, \boldsymbol{\theta}^{[t]})$. Substituting in the expressions for the normal likelihood $Pr(\mathbf{x}_i|h_i)$ and the gamma prior $Pr(h_i)$, we find that

$$\begin{aligned} E\left[\log\left[Pr(\mathbf{x}_i|h_i,\boldsymbol{\theta})\right]\right] &= \frac{D\mathrm{E}[\log h_i] - D\log 2\pi - \log|\boldsymbol{\Sigma}| - (\mathbf{x}_i-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}_i-\boldsymbol{\mu})\mathrm{E}[h_i]}{2} \\ E\left[\log\left[Pr(h_i)\right]\right] &= \frac{\nu}{2}\log\left[\frac{\nu}{2}\right] - \log\Gamma\left[\frac{\nu}{2}\right] + \left(\frac{\nu}{2}-1\right)\mathrm{E}[\log h_i] - \frac{\nu}{2}\mathrm{E}[h_i]. \end{aligned} \tag{7.29}$$

To optimize $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, we take derivatives of equation 7.28, set the resulting expressions to zero and rearrange to yield update equations

$$\begin{aligned} \boldsymbol{\mu}^{[t+1]} &= \frac{\sum_{i=1}^{I}\mathrm{E}[h_i]\mathbf{x}_i}{\sum_{i=1}^{I}\mathrm{E}[h_i]} \\ \boldsymbol{\Sigma}^{[t+1]} &= \frac{\sum_{i=1}^{I}\mathrm{E}[h_i](\mathbf{x}_i-\boldsymbol{\mu}^{[t+1]})(\mathbf{x}_i-\boldsymbol{\mu}^{[t+1]})^T}{\sum_{i=1}^{I}\mathrm{E}[h_i]}. \end{aligned} \tag{7.30}$$

These update equations have an intuitive form: for the mean, we are computing a weighted sum of the data. Outliers in the data set will tend to be explained best by the normal distributions in the infinite mixture which have larger covariances: for these distributions h is small (h scales the normal covariance inversely). Consequently, $\mathrm{E}[h]$ is small, and they are weighted less in the sum. The update for the $\boldsymbol{\Sigma}$ has a similar interpretation.

Unfortunately, there is no closed form solution for the degrees of freedom ν. We hence perform a one-dimensional line search to maximize equation 7.28 having substituted in the updated values of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, or use one of the optimization techniques described in chapter 9.

When we fit a t-distribution with a diagonal scale matrix $\boldsymbol{\Sigma}$ to the face data set, the mean $\boldsymbol{\mu}$ and scale matrix $\boldsymbol{\Sigma}$ (not shown) look visually similar to those for the normal model (figure 7.2). However, the model is not the same. The fitted degrees of freedom ν is 6.6. This low value indicates that the distribution has significantly longer tails than the normal model.

In conclusion, the multivariate t-distribution provides an improved description of data with outliers (figure 7.14). It has just one more parameter than the normal (the degrees of freedom, ν), and subsumes the normal as a special case (where ν becomes very large). However, this generality comes at a cost: there is no closed form solution for the maximum likelihood parameters and so we must resort to more complex approaches such as the EM algorithm to fit the distribution.

7.6 Factor analysis

We now address the final problem with the normal distribution. Visual data are often very high dimensional; in the face detection task the data comes in the form of 60×60 RGB images and is hence characterized as a $60\times60\times3 = 10800$ dimensional vector. To model this data with the full multivariate normal distribution, we

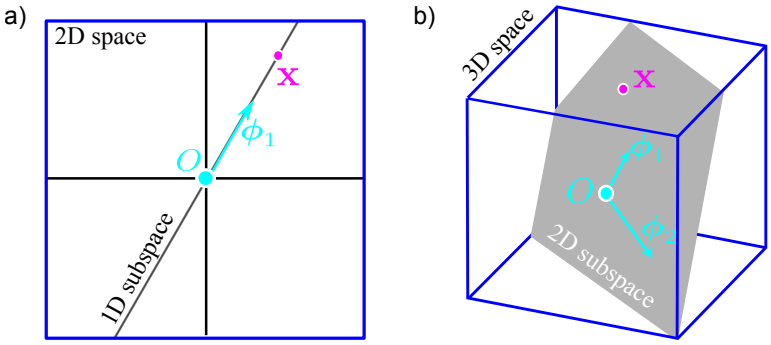


Figure 7.19 Linear subspaces a) A one dimensional subspace (a line through the origin, O) is embedded in a two dimensional space. Any point $\mathbf{x}$ in the subspace can be reached by weighting the single basis vector $\boldsymbol{\phi}_1$ appropriately. b) A two dimensional subspace (a plane through the origin, O) is embedded in a three dimensional space. Any point $\mathbf{x}$ in the subspace can be reached using a linear combination $\mathbf{x} = \alpha\boldsymbol{\phi}_1 + \beta\boldsymbol{\phi}_2$ of the two basis functions $\boldsymbol{\phi}_1, \boldsymbol{\phi}_2$ that describe the subspace. In general a K-dimensional subspace can be described using K basis functions.

require a covariance matrix of dimensions 10800×10800: we would need a very large number of training examples to get good estimates of all of these parameters in the absence of prior information. Furthermore, to store the covariance matrix we will need a large amount of memory, and there remains the problem of inverting this large matrix when we evaluate the normal likelihood (equation 5.1).

Of course, we could just use the diagonal form of the covariance matrix which contains only 10800 parameters. However, this is too great a simplification: we are assuming that each dimension of the data is independent and for face images this is clearly not true. For example, in the cheek region, the RGB values of neighboring pixels covary very closely. A good model should capture this information.

Factor analysis provides a compromise in which the covariance matrix is structured so that it contains fewer unknown parameters than the full matrix but more than the diagonal form. One way to think about the covariance of a factor analyzer is that it models part of the high-dimensional space with a full model and mops up remaining variation with a diagonal model.

More precisely, the factor analyzer describes a *linear subspace* with a full covariance model. A linear subspace is a subset of a high dimensional space that can be reached by taking linear combinations (weighted sums) of a fixed set of basis functions (figure 7.19). So, a line through the origin is a subspace in two dimensions as we can reach any point on it by weighting a single basis vector. A line through the origin is also a subspace in three dimensions, but so is a plane through the origin: we can reach any point on the plane by taking linear combinations of two basis vectors. In general, a D-dimensional space contains subspaces of dimensions $1, 2, \ldots, D-1$.

The probability density function of a factor analyzer is given by

$$Pr(\mathbf{x}) = \text{Norm}_{\mathbf{x}}[\boldsymbol{\mu}, \boldsymbol{\Phi}\boldsymbol{\Phi}^T + \boldsymbol{\Sigma}], \tag{7.31}$$

where the covariance matrix $\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \boldsymbol{\Sigma}$ contains a sum of two terms. The first term $\boldsymbol{\Phi}\boldsymbol{\Phi}^T$ describes a full covariance model over the subspace: the K columns of the portrait[1] rectangular matrix $\boldsymbol{\Phi} = [\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \ldots, \boldsymbol{\phi}_K]$ are termed *factors*. The factors are basis vectors that determine the subspace modeled. When we fit the model to data the factors will span the set of directions where the data covary the most. The second term $\boldsymbol{\Sigma}$ is a diagonal matrix that accounts for all remaining variation.

Notice that this model has $K \times D$ parameters to describe $\boldsymbol{\Phi}$ and another D parameters to describe the diagonal matrix $\boldsymbol{\Sigma}$. If the number of factors K is much less than the dimensionality of the data D then this model has fewer parameters than a normal with full covariance and hence can be learned from fewer training examples.

When $\boldsymbol{\Sigma}$ is a constant multiple of the identity matrix (i.e., models spherical covariance) the model is called *probabilistic principal component analysis*. This simpler model has slightly fewer parameters and can be fit in closed form (i.e., without the need for the EM algorithm), but otherwise it has no advantages over factor analysis (see section 17.5.1 for more details). We will hence restrict ourselves to a discussion of the more general factor analysis model.

7.6.1 Factor analysis as a marginalization

As for the mixtures of Gaussians and the t-distribution, it is possible to view the factor analysis model as a marginalization of a joint distribution between the observed data $\mathbf{x}$ and a K-dimensional hidden variable $\mathbf{h}$. We define

$$\begin{aligned} Pr(\mathbf{x}|\mathbf{h}) &= \text{Norm}_{\mathbf{x}}[\boldsymbol{\mu} + \boldsymbol{\Phi}\mathbf{h}, \boldsymbol{\Sigma}] \\ Pr(\mathbf{h}) &= \text{Norm}_{\mathbf{h}}[\mathbf{0}, \mathbf{I}], \end{aligned} \tag{7.32}$$

where $\mathbf{I}$ represents the identity matrix. It can be shown (but is not obvious) that

Problem 7.8

$$\begin{aligned} Pr(\mathbf{x}) = \int Pr(\mathbf{x}, \mathbf{h}) d\mathbf{h} &= \int Pr(\mathbf{x}|\mathbf{h}) Pr(\mathbf{h}) \, d\mathbf{h} \\ &= \int \text{Norm}_{\mathbf{x}}[\boldsymbol{\mu} + \boldsymbol{\Phi}\mathbf{h}, \boldsymbol{\Sigma}] \text{Norm}_{\mathbf{h}}[\mathbf{0}, \mathbf{I}] \, d\mathbf{h} \\ &= \text{Norm}_{\mathbf{x}}[\boldsymbol{\mu}, \boldsymbol{\Phi}\boldsymbol{\Phi}^T + \boldsymbol{\Sigma}], \end{aligned} \tag{7.33}$$

which was the original definition of the factor analyzer (equation 7.31).

Expressing factor analysis as a marginalization reveals a simple method to draw samples from the distribution. We first draw a hidden variable $\mathbf{h}$ from the normal prior. We then draw the sample $\mathbf{x}$ from a normal distribution with mean $\boldsymbol{\mu} + \boldsymbol{\Phi}\mathbf{h}$ and diagonal covariance $\boldsymbol{\Sigma}$ (see equation 7.32).

[1] That is, it is tall and thin, as opposed to landscape which would be short and wide

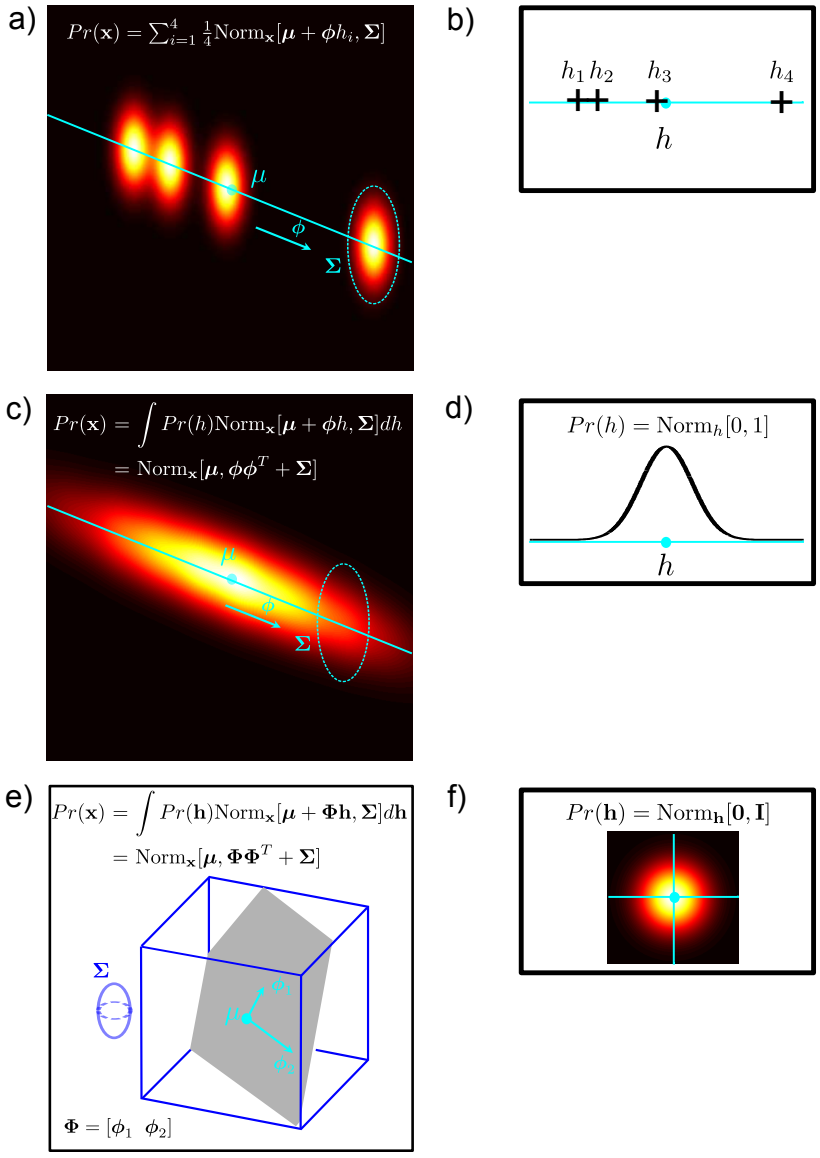


Figure 7.20 Relationship between factor analysis and mixtures of Gaussians (MoG). a) Consider a MoG model where each component has identical diagonal covariance $\boldsymbol{\Sigma}$. We could describe variation in a particular direction $\boldsymbol{\phi}$ by parameterizing the mean of each Gaussian as $\boldsymbol{\mu}_i = \boldsymbol{\mu} + \boldsymbol{\phi} h_i$. b) Different values of the scalar hidden variable h_i determine different positions along direction $\boldsymbol{\phi}$. c) Now we replace the MoG with an infinite sum (integral) over a continuous family of Gaussians, each of which is determined by a certain value of h. d) If we choose the prior over the hidden variable to be normal, then this integral has a closed form solution and is a factor analyzer. e) More generally we want to describe variance in a set of directions $\boldsymbol{\Phi} = [\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \ldots, \boldsymbol{\phi}_K]$ in a high dimensional space. f) To this end we use a K-dimensional hidden variable $\mathbf{h}$ and an associated normal prior $Pr(\mathbf{h})$.

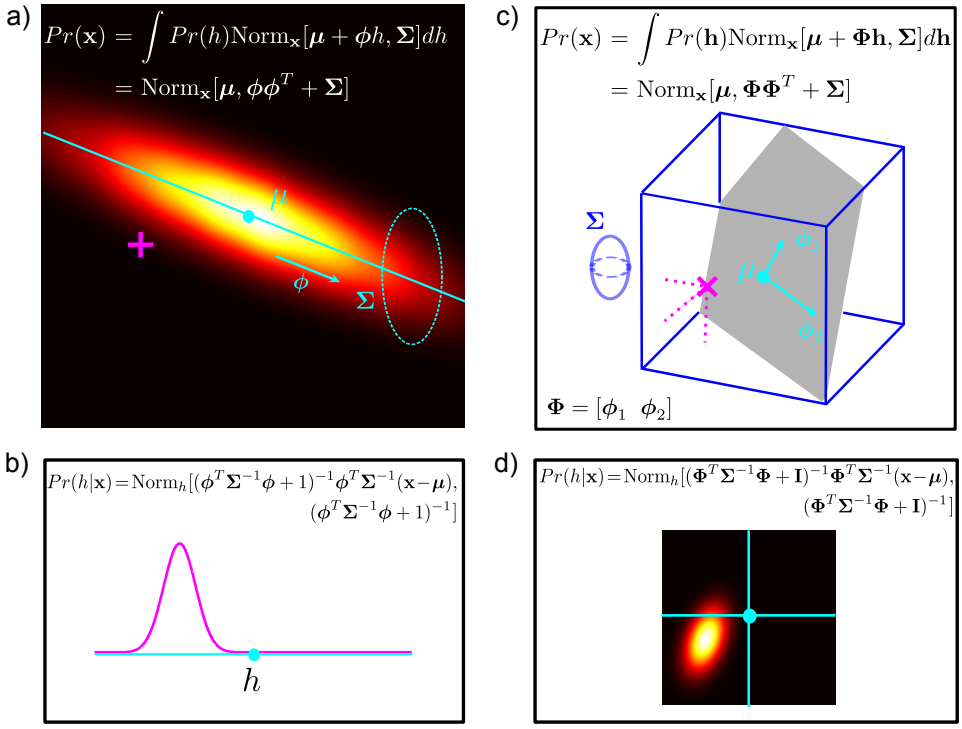


Figure 7.21 E-step for expectation maximization algorithm for factor analysis. (a) Two dimensional case with one factor. We are given a data point $\mathbf{x}$ (purple cross). (b) In the E-step we seek a distribution over possible values of the associated hidden variable h. It can be shown that this posterior distribution over h is itself normally distributed. (c) Three-dimensional case with two factors. Given a data point $\mathbf{x}$ (purple cross), we aim to find a distribution (d) over possible values of the associated hidden variable $\mathbf{h}$. Once more this posterior is normally distributed.

This leads us to a simple interpretation of the hidden variable $\mathbf{h}$: each element h_k weights the associated basis function $\boldsymbol{\phi}_k$ in the matrix $\boldsymbol{\Phi}$ and hence defines a point on the subspace (figure 7.19). The final density (equation 7.31) is hence an infinite weighted sum of normal distributions with the same diagonal covariance $\boldsymbol{\Sigma}$ and means $\boldsymbol{\mu} + \boldsymbol{\Phi}\mathbf{h}$ that are distributed over the subspace. The relationship between mixture models and factor analysis is explored further in figure 7.20.

7.6.2 Expectation maximization for learning factor analyzers

Algorithm 7.3

Since the factor analyzer can be expressed as a marginalization of a joint distribution between the observed data $\mathbf{x}$ and a hidden variable $\mathbf{h}$ (equation 7.33), it is possible to use the EM algorithm to learn the parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}, \boldsymbol{\Phi}, \boldsymbol{\Sigma}\}$. Once

more, we follow the recipe described in section 7.3.

In the E-step (figure 7.21) we optimize the bound with respect to the distributions $q_i(\mathbf{h}_i)$. To do this we compute the posterior probability distribution $Pr(\mathbf{h}_i|\mathbf{x}_i)$ over each hidden variable $\mathbf{h}_i$ given the associated observed data $\mathbf{x}_i$ and the current values of the parameters $\boldsymbol{\theta}^{[t]}$. To this end we apply Bayes' rule:

Problem 7.9

$$\begin{aligned}\hat{q}(\mathbf{h}_i) &= Pr(\mathbf{h}_i|\mathbf{x}_i, \boldsymbol{\theta}^{[t]}) \\ &= \frac{Pr(\mathbf{x}_i|\mathbf{h}_i, \boldsymbol{\theta}^{[t]})Pr(\mathbf{h}_i)}{Pr(\mathbf{x}_i|\boldsymbol{\theta}^{[t]})} \\ &= \frac{\text{Norm}_{\mathbf{x}_i}[\boldsymbol{\mu} + \boldsymbol{\Phi}\mathbf{h}_i, \boldsymbol{\Sigma}]\text{Norm}_{\mathbf{h}_i}[\mathbf{0}, \mathbf{I}]}{Pr(\mathbf{x}_i|\boldsymbol{\theta}^{[t]})} \\ &= \text{Norm}_{\mathbf{h}_i}[(\boldsymbol{\Phi}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi} + \mathbf{I})^{-1}\boldsymbol{\Phi}^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}), (\boldsymbol{\Phi}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi} + \mathbf{I})^{-1}]\end{aligned} \tag{7.34}$$

where we have made use of the change of variables relation (section 5.7) and then the fact that the product of two normals is proportional to a third normal (section 5.6). The resulting constant of proportionality exactly cancels out with the term $Pr(\mathbf{x})$ ensuring that the result is a valid probability distribution.

The E-step computes a probability distribution over the possible causes $\mathbf{h}$ for the observed data. This implicitly defines a probability distribution over the positions $\boldsymbol{\Phi}\mathbf{h}$ on the subspace that might have generated this example.

We extract the following expectations from the posterior distribution (equation 7.34) as they will be needed in the M-step:

$$\begin{aligned}\text{E}[\mathbf{h}_i] &= (\boldsymbol{\Phi}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi} + \mathbf{I})^{-1}\boldsymbol{\Phi}^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu}) \\ \text{E}[\mathbf{h}_i\mathbf{h}_i^T] &= E\left[(\mathbf{h}_i - \text{E}[\mathbf{h}_i])(\mathbf{h}_i - \text{E}[\mathbf{h}_i])^T\right] + \text{E}[\mathbf{h}_i]\text{E}[\mathbf{h}_i]^T \\ &= (\boldsymbol{\Phi}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi} + \mathbf{I})^{-1} + \text{E}[\mathbf{h}_i]\text{E}[\mathbf{h}_i]^T.\end{aligned} \tag{7.35}$$

In the M-step, we optimize the bound with respect to the parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}, \boldsymbol{\Phi}, \boldsymbol{\Sigma}\}$ so that

$$\begin{aligned}\hat{\boldsymbol{\theta}}^{[t+1]} &= \underset{\boldsymbol{\theta}}{\text{argmax}}\left[\sum_{i=1}^{I}\int \hat{q}_i(\mathbf{h}_i)\log\left[Pr(\mathbf{x}, \mathbf{h}_i, \boldsymbol{\theta})\right]\, d\mathbf{h}_i\right] \\ &= \underset{\boldsymbol{\theta}}{\text{argmax}}\left[\sum_{i=1}^{I}\int \hat{q}_i(\mathbf{h}_i)\left[\log\left[Pr(\mathbf{x}|\mathbf{h}_i, \boldsymbol{\theta})\right] + \log\left[Pr(\mathbf{h}_i)\right]\right]\, d\mathbf{h}_i\right] \\ &= \underset{\boldsymbol{\theta}}{\text{argmax}}\left[\sum_{i=1}^{I}\int \hat{q}_i(\mathbf{h}_i)\log\left[Pr(\mathbf{x}|\mathbf{h}_i, \boldsymbol{\theta})\right]\, d\mathbf{h}_i\right] \\ &= \underset{\boldsymbol{\theta}}{\text{argmax}}\left[\sum_{i=1}^{I} E\left[\log Pr(\mathbf{x}|\mathbf{h}_i, \boldsymbol{\theta})\right]\right],\end{aligned} \tag{7.36}$$

where we have removed the term $\log[Pr(\mathbf{h}_i)]$ as it is not dependent on the variables $\boldsymbol{\theta}$. The expectations $\text{E}[\bullet]$ are taken with respect to the relevant posterior distributions $\hat{q}_i(\mathbf{h}_i) = Pr(\mathbf{h}_i|\mathbf{x}_i, \boldsymbol{\theta}^{[t]})$. The expression for $\log[Pr(\mathbf{x}_i|\mathbf{h}_i)]$ is given by

$$\log[Pr(\mathbf{x}_i|\mathbf{h}_i)] = -\frac{D\log(2\pi) + \log|\boldsymbol{\Sigma}| + (\mathbf{x}_i - \boldsymbol{\mu} - \boldsymbol{\Phi}\mathbf{h}_i)^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu} - \boldsymbol{\Phi}\mathbf{h}_i)}{2}, \quad (7.37)$$

where D is the dimensionality of the data.

Problem 7.10

We optimize equation 7.36 by taking derivatives with respect to the parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}, \boldsymbol{\Phi}, \boldsymbol{\Sigma}\}$, equating the resulting expressions to zero and rearranging to yield

$$\begin{aligned}
\hat{\boldsymbol{\mu}} &= \frac{\sum_{i=1}^{I}\mathbf{x}_i}{I} \\
\hat{\boldsymbol{\Phi}} &= \left(\sum_{i=1}^{I}(\mathbf{x}_i - \hat{\boldsymbol{\mu}})\mathrm{E}[\mathbf{h}_i]^T\right)\left(\sum_{i=1}^{I}\mathrm{E}[\mathbf{h}_i\mathbf{h}_i^T]\right)^{-1} \\
\hat{\boldsymbol{\Sigma}} &= \frac{1}{I}\sum_{i=1}^{I}\mathrm{diag}\left[(\mathbf{x}_i - \hat{\boldsymbol{\mu}})(\mathbf{x}_i - \hat{\boldsymbol{\mu}})^T - \hat{\boldsymbol{\Phi}}\mathrm{E}[\mathbf{h}_i](\mathbf{x}^T - \boldsymbol{\mu})\right], \qquad (7.38)
\end{aligned}$$

where the function diag[•] is the operation of setting all elements of the matrix argument to zero except those on the diagonal.

Figure 7.22 shows the parameters of a factor analysis model fitted to the face data using ten iterations of the EM algorithm. The different factors encode different *modes of variation* of the data set which often have real-world interpretations such as changes in pose or lighting.

In conclusion, the factor analyzer is an efficient model for capturing the covariance in high-dimensional data. It devotes one set of parameters $\boldsymbol{\Phi}$ to describing the directions in which the data are most correlated and a second set $\boldsymbol{\Sigma}$ describes the remaining variation.

7.7 Combining models

The mixture of Gaussians, t-distribution, and factor analysis models are constructed similarly: each is a weighted sum or integral of a set of constituent normal distributions. Mixture of Gaussian models consist of a weighted sum of K normal distributions with different means and variances. The t-distribution consists of an infinite weighted sum of normal distributions with the same mean, but different covariances. Factor analysis models consist of an infinite weighted sum of normal distributions with different means, but the same diagonal covariance.

In light of these similarities, it is perhaps unsurprising then that the models can be easily combined. If we combine mixture models and factor analyzers, we get a *mixture of factor analyzers (MoFA)* model. This is a weighted sum of factor analyzers, each of which has a different mean and allocates high probability density to a different subspace. Similarly, combining mixture models and t-distributions results in a *mixture of t-distributions* or *robust mixture model*. Combining t-distributions and factor analyzers, we can construct a *robust subspace model*, which models data

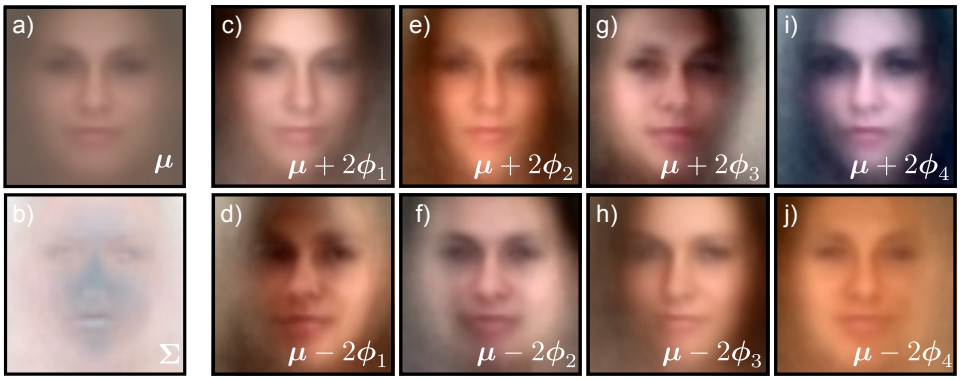


Figure 7.22 Factor analyzer with ten factors (four shown) for face classes. a) Mean $\boldsymbol{\mu}$ for face model. b) Diagonal covariance component $\boldsymbol{\Sigma}$ for face model. To visualize the effect of the first factor $\boldsymbol{\phi}_1$ we add (c) or subtract (d) a multiple of it from the mean: we are moving along one axis of the 10D subspace that seems to encode mainly the mean intensity. Other factors (e-j) encode changes in the hue and the pose of the face.

that lie primarily in a subspace but is tolerant to outliers. Finally, combining all three models, we get a *mixture of robust subspace models*. This has the combined benefits of all three approaches (it is multi-modal and robust and makes efficient use of parameters). The associated density function is:

$$Pr(\mathbf{x}) = \sum_{k=1}^{K} \lambda_k \text{Stud}_{\mathbf{x}}\left[\boldsymbol{\mu}_k, \boldsymbol{\Phi}_k\boldsymbol{\Phi}_k^T + \boldsymbol{\Sigma}_k, \nu_k\right], \tag{7.39}$$

where $\boldsymbol{\mu}_k$,$\boldsymbol{\Phi}_k$ and $\boldsymbol{\Sigma}_k$ represent the mean, factors, and diagonal covariance matrix belonging to the k^{th} component, λ_k represents the weighting of the k^{th} component and ν_k represents the degrees of freedom of the k^{th} component. To learn this model, we would use a series of interleaved expectation maximization algorithms.

7.8 Expectation maximization in detail

Throughout this chapter, we have employed the expectation maximization (EM) algorithm, using the recipe from section 7.3. We now examine the EM algorithm in detail to understand why this recipe works.

The EM algorithm is used to find maximum likelihood or MAP estimates of model parameters $\boldsymbol{\theta}$ where the likelihood $Pr(\mathbf{x}|\boldsymbol{\theta})$ of the data $\mathbf{x}$ can be written as

$$Pr(\mathbf{x}|\boldsymbol{\theta}) = \sum_k Pr(\mathbf{x}, h=k|\boldsymbol{\theta}) = \sum_k Pr(\mathbf{x}|h=k,\boldsymbol{\theta})Pr(h=k) \tag{7.40}$$

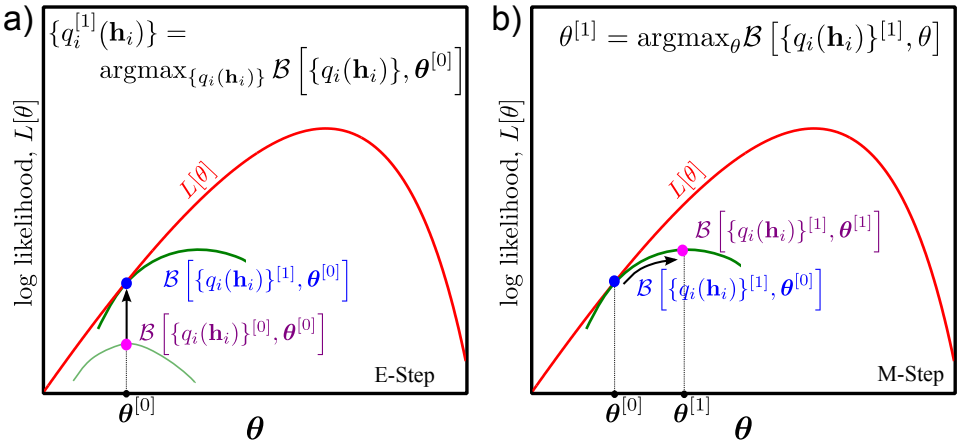


Figure 7.23 E-step and M-step. a) In the E-step, we manipulate the distributions $\{q_i(\mathbf{h}_i)\}$ to find the best new lower bound given parameters $\boldsymbol{\theta}$. This optimal lower bound will touch the log likelihood at the current parameter values $\boldsymbol{\theta}$ (we cannot do better than this!). b) In the M-step, we hold $\{q_i(\mathbf{h}_i)\}$ constant and optimize $\boldsymbol{\theta}$ with respect to the new bound.

and

$$Pr(\mathbf{x}|\boldsymbol{\theta}) = \int Pr(\mathbf{x}, \mathbf{h}|\boldsymbol{\theta})\, d\mathbf{h} = \int Pr(\mathbf{x}|\mathbf{h}, \boldsymbol{\theta}) Pr(\mathbf{h})\, d\mathbf{h} \tag{7.41}$$

for discrete and continuous hidden variables, respectively. In other words, the likelihood $Pr(\mathbf{x}|\boldsymbol{\theta})$ is a marginalization of a joint distribution over the data and the hidden variables. We will work with the continuous case.

The EM algorithm relies on the idea of a lower bounding function (or *lower bound*), $\mathcal{B}[\boldsymbol{\theta}]$ on the log likelihood. This is a function of the parameters $\boldsymbol{\theta}$ that is always guaranteed to be equal to or lower than the log likelihood. The lower bound is carefully chosen so that it is easy to maximize with respect to the parameters.

This lower bound is also parameterized by a set of probability distributions $\{q_i(\mathbf{h}_i)\}_{i=1}^I$ over the hidden variables, so we write it as $\mathcal{B}[\{q_i(\mathbf{h}_i)\}, \boldsymbol{\theta}]$. Different probability distributions $q_i(\mathbf{h}_i)$ predict different lower bounds $\mathcal{B}[\{q_i(\mathbf{h}_i)\}, \boldsymbol{\theta}]$ and hence different functions of $\boldsymbol{\theta}$ that lie everywhere below the true log likelihood (figure 7.5b).

In the EM algorithm we alternate between expectation steps (E-steps) and maximization steps (M-steps) where

- in the E-step (figure 7.23a) we fix $\boldsymbol{\theta}$ and find the best lower bound $\mathcal{B}[\{q_i(\mathbf{h}_i)\}, \boldsymbol{\theta}]$ with respect to the distributions $q_i(\mathbf{h}_i)$. In other words, at iteration t

$$q_i^{[t]}[\mathbf{h}_i] = \underset{q_i[\mathbf{h}_i]}{\text{argmax}} \left[\mathcal{B}[\{q_i(\mathbf{h}_i)\}, \theta^{[t-1]}]\right]. \tag{7.42}$$

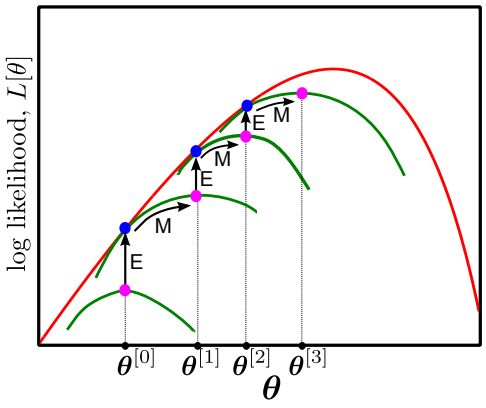


Figure 7.24 Expectation maximization algorithm. We iterate the expectation and maximization steps by alternately changing the distributions $q_i(\mathbf{h}_i)$ and the parameter $\boldsymbol{\theta}$ so that the bound increases. In the E-step, the bound is maximized with respect to $q_i(\mathbf{h}_i)$ for fixed parameters $\boldsymbol{\theta}$: the new function with respect to $\boldsymbol{\theta}$ touches the true log likelihood at $\boldsymbol{\theta}$. In the M-step, we find the maximum of this function. In this way we are guaranteed to reach a local maximum in the likelihood function.

The best lower bound will be a function that is as high as possible at the current parameter estimates $\boldsymbol{\theta}$. Since it must be everywhere equal to or lower than the log likelihood, the highest possible value is the log likelihood itself. So the bound touches the log-likelihood curve for the current parameters $\boldsymbol{\theta}$.

- in the M-step (figure 7.23b), we fix $q_i(\mathbf{h}_i)$ and find the values of $\boldsymbol{\theta}$ that maximize this bounding function $\mathcal{B}[\{q_i(\mathbf{h}_i)\}, \boldsymbol{\theta}]$. In other words, we compute

$$\boldsymbol{\theta}^{[t]} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left[\mathcal{B}[\{q_i^{[t]}(\mathbf{h}_i)\}, \boldsymbol{\theta}] \right]. \tag{7.43}$$

By iterating these steps, the (local) maximum of the actual log likelihood is approached (figure 7.24). To complete our picture of the EM algorithm, we must

- define $\mathcal{B}[\{q_i(\mathbf{h}_i)\}, \boldsymbol{\theta}^{[t-1]}]$ and show that it always lies below the log likelihood,
- show which probability distribution $q_i(\mathbf{h}_i)$ optimizes the bound in the E-step,
- show how to optimize the bound with respect to $\boldsymbol{\theta}$ in the M-step.

These three issues are tackled in sections 7.8.1, 7.8.2 and 7.8.3, respectively.

7.8.1 Lower bound for EM algorithm

We define the lower bound $\mathcal{B}[\{q_i(\mathbf{h}_i)\}, \boldsymbol{\theta}]$ to be

$$\begin{aligned}
\mathcal{B}[\{q_i(\mathbf{h}_i)\}, \boldsymbol{\theta}] &= \sum_{i=1}^{I} \int q_i(\mathbf{h}_i) \log \left[\frac{Pr(\mathbf{x}_i, \mathbf{h}_i|\boldsymbol{\theta})}{q_i(\mathbf{h}_i)} \right] d\mathbf{h}_i \\
&\leq \sum_{i=1}^{I} \log \left[\int q_i(\mathbf{h}_i) \frac{Pr(\mathbf{x}_i, \mathbf{h}_i|\boldsymbol{\theta})}{q_i(\mathbf{h}_i)} \, d\mathbf{h}_i \right] \\
&= \sum_{i=1}^{I} \log \left[\int Pr(\mathbf{x}_i, \mathbf{h}_i|\boldsymbol{\theta}) \, d\mathbf{h}_i \right],
\end{aligned} \tag{7.44}$$

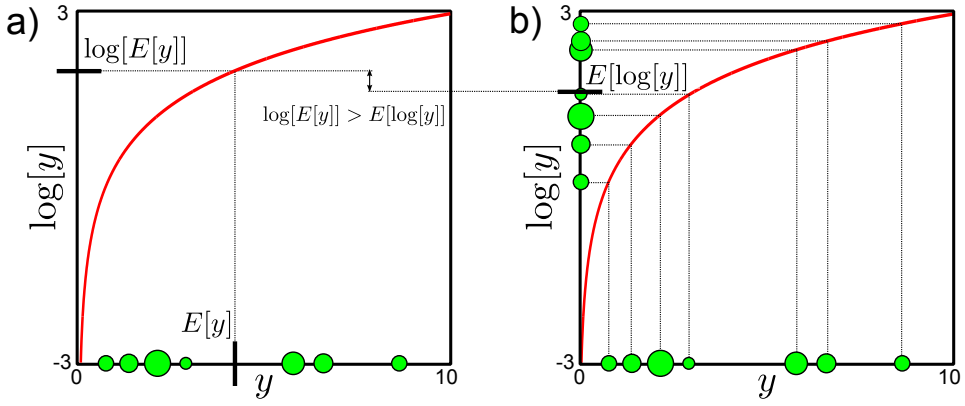


Figure 7.25 Jensen's inequality for the logarithmic function (discrete case). a) Taking a weighted average of examples E[y] and passing them through the log function. b) Passing the samples through the log function and taking a weighted average E[log[y]]. The latter case always produces a smaller value than the former (E[log[y]] $\leq$ log(E[y])): higher valued examples are relatively compressed by the concave log function.

where we have used *Jensen's inequality* between the first and second lines. This states that because the logarithm is a concave function, we can write

$$\int Pr(y)\log[y]dy \leq \log\left[\int yPr(y)dy\right] \tag{7.45}$$

or E[log[y]] $\leq$ log(E[y]). Figure 7.25 illustrates Jensen's inequality for a discrete variable.

7.8.2 The E-step

In the E-step, we update the bound $\mathcal{B}[\{q_i(\mathbf{h}_i)\}, \boldsymbol{\theta}]$ with respect to the distributions $q_i(\mathbf{h}_i)$. To see how to do this, we manipulate the expression for the bound as follows:

$$\begin{aligned}
\mathcal{B}[\{q_i(\mathbf{h}_i)\},\boldsymbol{\theta}] &= \sum_{i=1}^{I}\int q_i(\mathbf{h}_i)\log\left[\frac{Pr(\mathbf{x}_i,\mathbf{h}_i|\boldsymbol{\theta})}{q_i(\mathbf{h}_i)}\right]\,d\mathbf{h}_i \\
&= \sum_{i=1}^{I}\int q_i(\mathbf{h}_i)\log\left[\frac{Pr(\mathbf{h}_i|\mathbf{x}_i,\boldsymbol{\theta})Pr(\mathbf{x}_i|\boldsymbol{\theta})}{q_i(\mathbf{h}_i)}\right]\,d\mathbf{h}_i \\
&= \sum_{i=1}^{I}\int q_i(\mathbf{h}_i)\log\left[Pr(\mathbf{x}_i|\boldsymbol{\theta})\right]\,d\mathbf{h}_i - \sum_{i=1}^{I}\int q_i(\mathbf{h}_i)\log\left[\frac{q_i(\mathbf{h}_i)}{Pr(\mathbf{h}_i|\mathbf{x}_i,\boldsymbol{\theta})}\right]\,d\mathbf{h}_i \\
&= \sum_{i=1}^{I}\log\left[Pr(\mathbf{x}_i|\boldsymbol{\theta})\right] - \sum_{i=1}^{I}\int q_i(\mathbf{h}_i)\log\left[\frac{q_i(\mathbf{h}_i)}{Pr(\mathbf{h}_i|\mathbf{x}_i,\boldsymbol{\theta})}\right]\,d\mathbf{h}_i, \qquad (7.46)
\end{aligned}$$

where the hidden variables from the first term are integrated out between the last two lines. The first term in this expression is constant with respect to the distributions $q_i(\mathbf{h}_i)$ and so to optimize the bound we must find the distributions $\hat{q}_i(\mathbf{h}_i)$ that satisfy

$$\begin{aligned}
\hat{q}_i(\mathbf{h}_i) &= \underset{q_i(\mathbf{h}_i)}{\text{argmax}}\left[-\int q_i(\mathbf{h}_i)\log\left[\frac{q_i(\mathbf{h}_i)}{Pr(\mathbf{h}_i|\mathbf{x}_i,\boldsymbol{\theta})}\right]\,d\mathbf{h}_i\right] \\
&= \underset{q_i(\mathbf{h}_i)}{\text{argmax}}\left[\int q_i(\mathbf{h}_i)\log\left[\frac{Pr(\mathbf{h}_i|\mathbf{x}_i,\boldsymbol{\theta})}{q_i(\mathbf{h}_i)}\right]\,d\mathbf{h}_i\right] \\
&= \underset{q_i(\mathbf{h}_i)}{\text{argmin}}\left[-\int q_i(\mathbf{h}_i)\log\left[\frac{Pr(\mathbf{h}_i|\mathbf{x}_i,\boldsymbol{\theta})}{q_i(\mathbf{h}_i)}\right]\,d\mathbf{h}_i\right]. \qquad (7.47)
\end{aligned}$$

This expression is known as the *Kullback-Leibler* divergence between $q_i(\mathbf{h}_i)$ and $Pr(\mathbf{h}_i|\mathbf{x}_i,\boldsymbol{\theta})$. It is a measure of distance between probability distributions. We can use the inequality $\log[y] \leq y-1$ (plot the functions to convince yourself of this!) to show that this cost function (including the minus sign) is always positive,

$$\begin{aligned}
\int q_i(\mathbf{h}_i)\log\left[\frac{Pr(\mathbf{h}_i|\mathbf{x}_i,\boldsymbol{\theta})}{q_i(\mathbf{h}_i)}\right]\,d\mathbf{h}_i &\leq \int q_i(\mathbf{h}_i)\left(\frac{Pr(\mathbf{h}_i|\mathbf{x}_i,\boldsymbol{\theta})}{q_i(\mathbf{h}_i)}-1\right)\,d\mathbf{h}_i \\
&= \int Pr(\mathbf{h}_i|\mathbf{x}_i,\boldsymbol{\theta}) - q_i(\mathbf{h}_i)\,d\mathbf{h}_i \\
&= 1-1=0, \qquad (7.48)
\end{aligned}$$

implying that when we reintroduce the minus sign the cost function *must* be positive. So the criteria in equation 7.46 will be maximized when the Kullback-Leibler divergence is zero. This value is reached when $q_i(\mathbf{h}_i)=Pr(\mathbf{h}_i|\mathbf{x}_i,\boldsymbol{\theta})$ so that

$$\begin{aligned}
\int q_i(\mathbf{h}_i)\log\left[\frac{Pr(\mathbf{h}_i|\mathbf{x}_i,\boldsymbol{\theta})}{q_i(\mathbf{h}_i)}\right]\,d\mathbf{h}_i &= \int Pr(\mathbf{h}_i|\mathbf{x}_i,\boldsymbol{\theta})\log\left[\frac{Pr(\mathbf{h}_i|\mathbf{x}_i,\boldsymbol{\theta})}{Pr(\mathbf{h}_i|\mathbf{x}_i,\boldsymbol{\theta})}\right]\,d\mathbf{h}_i \\
&= \int Pr(\mathbf{h}_i|\mathbf{x}_i,\boldsymbol{\theta})\log\left[1\right]\,d\mathbf{h}_i = 0. \qquad (7.49)
\end{aligned}$$

In other words, to maximize the bound with respect to $q_i(\mathbf{h}_i)$ we set this to be the posterior distribution $Pr(\mathbf{h}_i|\mathbf{x}_i, \boldsymbol{\theta})$ over the hidden variables $\mathbf{h}_i$ given the current set of parameters. In practice, this is computed using Bayes' rule,

$$Pr(\mathbf{h}_i|\mathbf{x}_i, \boldsymbol{\theta}) = \frac{Pr(\mathbf{x}_i|\mathbf{h}_i, \boldsymbol{\theta})Pr(\mathbf{h}_i)}{Pr(\mathbf{x}_i)}. \tag{7.50}$$

So the E-step consists of computing the posterior distribution for each hidden variable using Bayes' rule.

7.8.3 The M-step

In the M-step we maximize the bound with respect to the parameters $\boldsymbol{\theta}$ so that

$$\begin{aligned}
\boldsymbol{\theta}^{[t]} &= \underset{\boldsymbol{\theta}}{\text{argmax}} \left[\mathcal{B}[\{q_i^{[t]}(\mathbf{h}_i)\}, \boldsymbol{\theta}]\right] \\
&= \underset{\boldsymbol{\theta}}{\text{argmax}} \left[\sum_{i=1}^{I} \int q_i^{[t]}(\mathbf{h}_i) \log\left[\frac{Pr(\mathbf{x}_i, \mathbf{h}_i|\boldsymbol{\theta})}{q_i^{[t]}(\mathbf{h}_i)}\right] d\mathbf{h}_i\right] \\
&= \underset{\boldsymbol{\theta}}{\text{argmax}} \left[\sum_{i=1}^{I} \int q_i^{[t]}(\mathbf{h}_i) \log\left[Pr(\mathbf{x}_i, \mathbf{h}_i|\boldsymbol{\theta})\right] - q_i^{[t]}(\mathbf{h}_i) \log\left[q_i^{[t]}(\mathbf{h}_i)\right] d\mathbf{h}_i\right] \\
&= \underset{\boldsymbol{\theta}}{\text{argmax}} \left[\sum_{i=1}^{I} \int q_i^{[t]}(\mathbf{h}_i) \log\left[Pr(\mathbf{x}_i, \mathbf{h}_i|\boldsymbol{\theta})\right] d\mathbf{h}_i\right],
\end{aligned} \tag{7.51}$$

where we have omitted the second term as it does not depend on the parameters. If you look back at the algorithms in this chapter, you will see that we have maximized exactly this criterion.

7.9 Applications

The models in this chapter have many uses in computer vision. We now present a cross-section of applications. As an example of two-class classification using mixtures of Gaussians densities, we reconsider the face detection application that has been a running theme throughout this chapter. To illustrate multi-class classification, we describe an object recognition model based on t-distributions. We also describe a segmentation application which is an example of unsupervised learning: we do not have labeled training data to build our model.

To illustrate the use of the factor analyzer for classification, we present a face recognition example. To illustrate its use for regression, we consider the problem of changing a face image from one pose to another. Finally, we highlight the fact that hidden variables can take on real-world interpretations by considering a model that explains the weak spatial alignment of digits.

7.9.1 Face detection

In face detection we attempt to infer a discrete label $w \in \{0, 1\}$ indicating whether a face is present or not based on observed data $\mathbf{x}$. We will describe the likelihood for each world state with a mixtures of Gaussians model, where the covariances of the Gaussian components are constrained to be diagonal so that

$$Pr(\mathbf{x}|w = m) = \sum_{k=1}^{K} \lambda_{mk} \text{Norm}_{\mathbf{x}}[\boldsymbol{\mu}_{km}, \boldsymbol{\Sigma}_{km}]. \tag{7.52}$$

where m indexes the world state and k indexes the component of the mixture distribution.

We will assume that we have no prior knowledge about whether the face is present or not so that $Pr(w = 0) = Pr(w = 1) = 0.5$. We fit the two likelihood terms using a set of labeled training pairs $\{\mathbf{x}_i, w_i\}$. In practice this means learning one mixtures of Gaussians model for the non-faces based on the data where $w_i = 0$ and a separate model for the faces based on the data where $w = 1$.

For test data $\mathbf{x}^*$, we compute the posterior probability over w using Bayes' rule

$$Pr(w^* = 1|\mathbf{x}^*) = \frac{Pr(\mathbf{x}^*|w^* = 1)Pr(w^* = 1)}{\sum_{k=0}^{1} Pr(\mathbf{x}^*|w^* = k)Pr(w^* = k)}. \tag{7.53}$$

Table 7.1 shows percent correct classification for 100 test examples The results are based on models learned from 1000 training examples of each class.

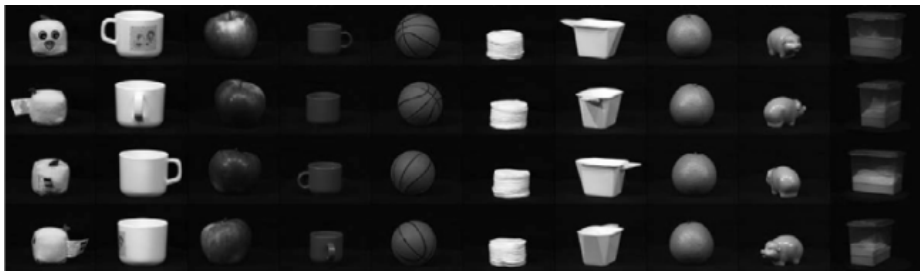
	Color	Grayscale	Equalized
Single Gaussian	76%	79%	80%
Mixture of 10 Gaussians	81%	85%	89%

Table 7.1: Percent correct classification rates for two different models and three different types of preprocessing. In each case, the data $\mathbf{x}^*$ was assumed to be a face if the posterior probability $Pr(w = 1|\mathbf{x}^*)$ was greater than 0.5.

The first column shows results where the data vector consists of the RGB values with a 24×24 region (the running example in this chapter used 60×60 pixel regions, but this is unnecessarily large). The results are compared to classification based on a single normal distribution. The subsequent columns of the table show results for systems trained and tested with grayscale 24×24 pixel regions and grayscale 24×24 regions that have been histogram equalized (section 13.1.2).

There are two insights to be gleaned from these classification results. First, the choice of model does make a difference; the mixtures of Gaussians density always results in better classification performance than the single Gaussian model. Second, the choice of *preprocessing* is also critical to the final performance. This book concerns models for vision, but it is important to understand that this is not the only thing that determines the final performance of real-world systems. A brief summary of preprocessing methods is presented in chapter 13.

a)



b)

74	66	66	90	61	97	47	62	43	71
74	25	44	91	43	80	63	24	08	55

Figure 7.26 Object recognition. a) The training database consists of a series of different views of ten different objects. The goal is to learn a class-conditional density function for each object and classify new examples using Bayes' rule. b) Percent correct results for class conditional densities based on the t-distribution (top row) and the normal distributions (bottom row). The robust model performs better, especially on objects with specularities. Images from Amsterdam library (Guesebroek *et al.* 2005).

The reader should not depart with the impression that this is a sensible approach to face detection. Even the best performance of 89% is far below what would be required in a real face detector: consider that in a single image we might classify patches at 10000 different positions and scales, so an 11% error rate will be unacceptable. Moreover, evaluating each patch under both class conditional density functions is too computationally expensive to be practical. In practice, face detection would normally be achieved using discriminative methods (chapter 9).

7.9.2 Object recognition

In object recognition the goal is to assign a discrete world vector $w_i \in \{1, 2, \ldots, M\}$ indicating which of M categories is present based on observed data $\mathbf{x}$. To this end, Aeschliman *et al.* (2010) split each image into 100 10×10 pixel regions arranged in a regular grid. The grayscale pixel data from the j^{th} region were concatenated to form a 100×1 vector $\mathbf{x}_{ij}$. They treated the regions independently, and described each with a t-distribution so that the class conditional density functions were

$$Pr(\mathbf{x}_i|w = m) = \prod_{j=1}^{J} \text{Stud}_{\mathbf{x}_{ij}}[\boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm}, \nu_{jm}]. \tag{7.54}$$

Figure 7.26 shows results based on training with 10 classes from the Amsterdam library of images (Guesebroek *et al.* 2005). Each class consists of 72 images taken at 5 degree intervals around the object. The data was divided randomly into 36 test images and 36 training images for each class. The prior probabilities of the classes were set to uniform, and the posterior distribution $Pr(w_i|\mathbf{x}_i)$ was calculated using

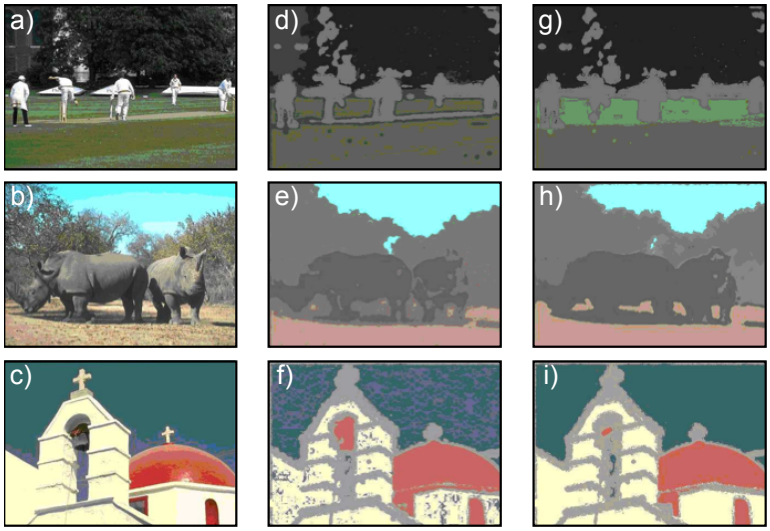


Figure 7.27 Segmentation. a-c) Original images. d-f) Segmentation results based on a mixture of five normal distributions. The pixels associated with the k^{th} component are colored with the mean RGB values of the pixels that are assigned to this value g-i) Segmentation results based on a mixture of K t-distributions. The segmentation here is less noisy than for the MoG model. Results from Sfikas *et al.* (2007) ©IEEE 2007.

Bayes' rule. A test object was classified according to the class with the highest posterior probability.

The results show the superiority of the t-distribution: for almost every class the percent correct performance is better, and this is especially true for objects such as the china pig where the specularities act as outliers. By adding just one more parameter per patch, the performance increases from a mean of 51% to 68%.

7.9.3 Segmentation

The goal of segmentation is to assign a discrete label $\{w_n\}_{n=1}^N$ which takes one of K values $w_n \in \{1, 2, \ldots, K\}$ to each of the N pixels in the image so that regions that belong to the same object are assigned the same label. The segmentation model depends on observed data vectors $\{\mathbf{x}_n\}_{n=1}^N$ at each of the N pixels that would typically include the RGB pixel values, the (x, y) position of the pixel and other information characterizing local texture.

We will frame this problem as *unsupervised learning*. In other words, we do not have the luxury of having training images where the state of the world is known. We must both learn the parameters $\boldsymbol{\theta}$ and estimate the world states $\{w_i\}_{i=1}^I$ from the image data $\{\mathbf{x}_n\}_{n=1}^N$.

We will assume that the k^{th} object is associated with a normal distribution with parameters $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ and prevalence λ_k so that

$$\begin{aligned} Pr(w_n) &= \text{Cat}_{w_n}[\boldsymbol{\lambda}] \\ Pr(\mathbf{x}_i|w_i = k) &= \text{Norm}_{\mathbf{x}_i}[\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k]. \end{aligned} \tag{7.55}$$

Marginalizing out the world state w, we have

$$Pr(\mathbf{x}_{1\ldots N}) = \prod_{n=1}^{N} \sum_{k=1}^{K} \lambda_k \text{Norm}_{\mathbf{x}_n}[\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k]. \tag{7.56}$$

To fit this model, we find the parameters $\boldsymbol{\theta} = \{\lambda_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^{K}$ using the EM algorithm. To assign a class to each pixel, we then find the value of the world state that have the highest posterior probability given the observed data

$$\hat{w}_i = \underset{w_i}{\text{argmax}} \left[Pr(w_i|\mathbf{x}_i) \right], \tag{7.57}$$

where the posterior is computed as in the E-step.

Figure 7.27 shows results from this model and a similar mixture model based on t-distributions from Sfikas *et al.* (2007). The mixture models manage to partition the image quite well into different regions. Unsurprisingly, the t-distribution results are rather less noisy than those based on the normal distribution.

7.9.4 Frontal face recognition

The goal of face identification (figure 7.28) is to assign a label $w \in \{1 \ldots M\}$ indicating which of M possible identities the face belongs to based on a data vector $\mathbf{x}$. The model is learned from labeled training data $\{\mathbf{x}_i, w_i\}_{i=1}^{I}$ where the identity is known. In a simple system, the data vector might consist of the concatenated grayscale values from the face image, which should be reasonably large (say 50×50 pixels) to ensure that the identity is well represented.

Since the data are high dimensional, a reasonable approach is to model each class conditional density function with a factor analyzer

$$Pr(\mathbf{x}_i|w_i = k) = \text{Norm}_{\mathbf{x}_i}[\boldsymbol{\mu}_k, \boldsymbol{\Phi}_k \boldsymbol{\Phi}_k^T + \boldsymbol{\Sigma}_k] \tag{7.58}$$

where the parameters for the k^{th} identity $\boldsymbol{\theta}_k = \boldsymbol{\mu}_k, \boldsymbol{\Phi}_k, \boldsymbol{\Sigma}_k$ can be learned from the subset of data that belongs to that identity using the EM algorithm. We also assign priors $P(w = k)$ according to the prevalence of each identity in the database.

To perform recognition, we compute the posterior distribution $Pr(w^*|\mathbf{x}^*)$ for the new data example $\mathbf{x}^*$ using Bayes' rule. We assign the identity that maximizes this posterior distribution.

This approach works well if there are sufficient examples of each gallery individual to learn a factor analyzer, and if the poses of all of the faces are similar. In the next example, we develop a method to change the pose of faces, so that we can cope with the case where the poses differ.

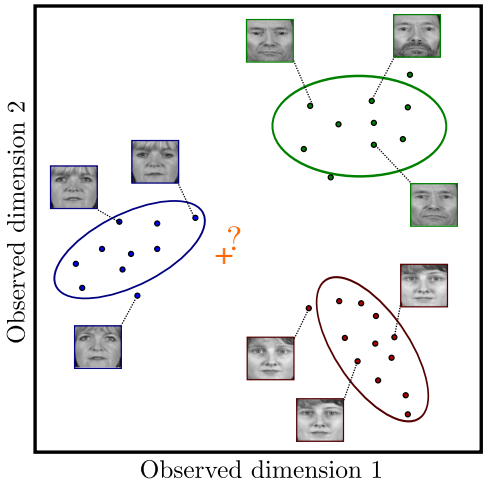


Figure 7.28 Face recognition. Our goal is to take the RGB values of a facial image $\mathbf{x}$ and assign a label $w \in \{1 \ldots K\}$ corresponding to the identity. Since the data are high-dimensional , we model the class conditional density function $Pr(\mathbf{x}|w=k)$ for each individual in the database as a factor analyzer. To classify a new face, we apply Bayes' rule with suitable priors $Pr(w)$ to compute the posterior distribution $Pr(w|\mathbf{x})$. We choose the label $\hat{w} = \text{argmax}_w[Pr(w = k|\mathbf{x})]$ that maximizes the posterior. This approach assumes that there are sufficient training examples to learn a factor analyzer for each class.

7.9.5 Changing face pose (regression)

To change the pose of the face, we predict the RGB values $\mathbf{w}$ of the face in the new pose given a face $\mathbf{x}$ in the old pose. This is different from the previous examples in that it is a regression problem: the output $\mathbf{w}$ is a continuous multidimensional variable rather than a class label.

We form a compound variable $\mathbf{z} = [\mathbf{x}^T\ \mathbf{w}^T]^T$ by concatenating the RGB values from the two poses together. We now model the joint density as $Pr(\mathbf{z}) = Pr(\mathbf{x}, \mathbf{w})$ with a factor analyzer

$$Pr(\mathbf{x}, \mathbf{w}) = Pr(\mathbf{z}) = \text{Norm}_{\mathbf{z}}[\boldsymbol{\mu}, \boldsymbol{\Phi}\boldsymbol{\Phi}^T + \boldsymbol{\Sigma}], \tag{7.59}$$

which we learn for paired training examples $\{\mathbf{x}_i, \mathbf{w}_i\}_{i=1}^I$ where the identity is known to be the same for each pair.

To find the non-frontal face $\mathbf{w}^*$ corresponding to a new frontal face $\mathbf{x}^*$ we use the approach of section **??**: the posterior over $\mathbf{w}^*$ is just the conditional distribution $Pr(\mathbf{w}|\mathbf{x})$. Since the joint distribution is normal, we can compute the posterior distribution in closed form using equation 5.5. Using the notation

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_{\mathbf{x}} \\ \boldsymbol{\mu}_{\mathbf{w}} \end{bmatrix} \text{ and } \boldsymbol{\Phi}\boldsymbol{\Phi}^T + \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Phi}_{\mathbf{x}}\boldsymbol{\Phi}_{\mathbf{x}}^T + \boldsymbol{\Sigma}_{\mathbf{x}} & \boldsymbol{\Phi}_{\mathbf{x}}\boldsymbol{\Phi}_{\mathbf{w}}^T \\ \boldsymbol{\Phi}_{\mathbf{w}}\boldsymbol{\Phi}_{\mathbf{x}}^T & \boldsymbol{\Phi}_{\mathbf{w}}\boldsymbol{\Phi}_{\mathbf{w}}^T + \boldsymbol{\Sigma}_{\mathbf{w}} \end{bmatrix}, \tag{7.60}$$

the posterior is given by

$$\begin{aligned} Pr(\mathbf{w}^*|\mathbf{x}^*) \;=\; \text{Norm}_{\mathbf{w}^*}[&\boldsymbol{\mu}_{\mathbf{w}} + \boldsymbol{\Phi}_{\mathbf{w}}\boldsymbol{\Phi}_{\mathbf{x}}^T(\boldsymbol{\Phi}_{\mathbf{x}}\boldsymbol{\Phi}_{\mathbf{x}}^T + \boldsymbol{\Sigma}_{\mathbf{x}})^{-1}(\mathbf{x}^* - \boldsymbol{\mu}_{\mathbf{x}}), \\ &\boldsymbol{\Phi}_{\mathbf{w}}\boldsymbol{\Phi}_{\mathbf{w}}^T + \boldsymbol{\Sigma}_{\mathbf{w}} - \boldsymbol{\Phi}_{\mathbf{w}}\boldsymbol{\Phi}_{\mathbf{x}}^T(\boldsymbol{\Phi}_{\mathbf{x}}\boldsymbol{\Phi}_{\mathbf{x}}^T + \boldsymbol{\Sigma}_{\mathbf{x}})^{-1}\boldsymbol{\Phi}_{\mathbf{x}}\boldsymbol{\Phi}_{\mathbf{w}}^T], \end{aligned} \tag{7.61}$$

and the most probable $\mathbf{w}^*$ is at the mean of this distribution.

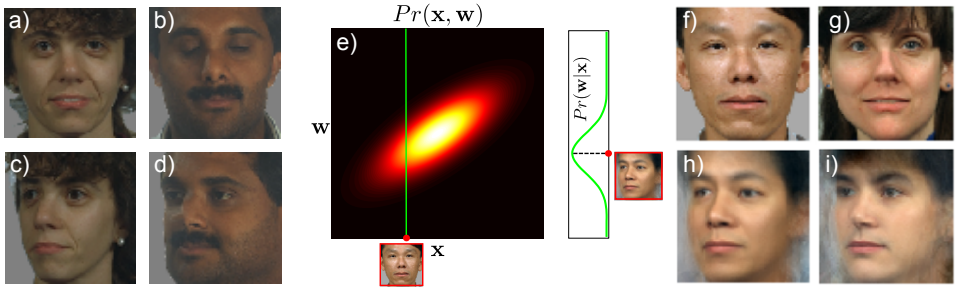


Figure 7.29 Regression example: we aim to predict quarter left face image $\mathbf{w}$ from the frontal image $\mathbf{x}$. To this end, we take paired examples of frontal faces (a-b) and quarter left faces (c-d) and learn a joint probability model $Pr(\mathbf{x}, \mathbf{w})$ by concatenating the variables to form $\mathbf{z} = [\mathbf{x}^T, \mathbf{w}^T]^T$ and fitting a factor analyzer to $\mathbf{z}$. e) Since the factor analyzer has a normal density (equation 7.31), we can predict the conditional distribution $Pr(\mathbf{w}|\mathbf{x})$ of a quarter-left face given a frontal face which will also be normal (see section 5.5). f-g) Two frontal faces. h-i) MAP predictions for non-frontal faces (the mean of the normal distribution $Pr(\mathbf{w}|\mathbf{x})$).

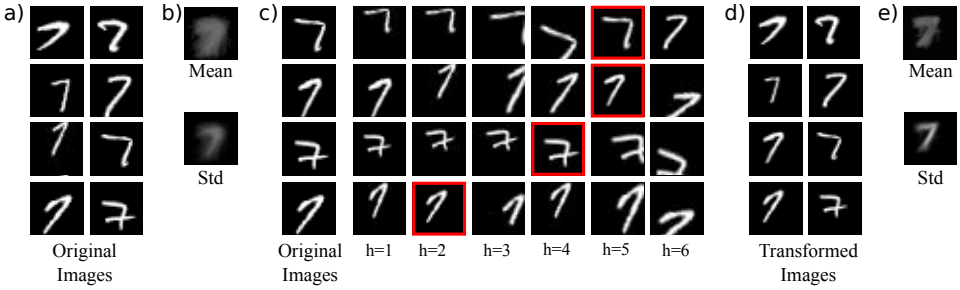


Figure 7.30 Modeling transformations with hidden variables. a) The original set of digit images are only weakly aligned. b) The mean and standard deviation images are consequently blurred out. The probability density model does not fit well. c) Each possible value of a discrete hidden variable represents a different transformation (here inverse transformations are shown). The red square highlights the most likely choice of hidden variable after ten iterations. d) The inversely transformed digits (based on most likely hidden variable). d) The new mean and standard deviation images are more focused: the probability density function fits better.

7.9.6 Transformations as hidden variables

Finally, we consider a model that is closely related to the mixture of Gaussians, where the hidden variables have a clear real-world interpretation. Consider the problem of building a density function for a set of poorly aligned images of digits

(figure 7.30). A simple normal distribution with diagonal covariance produces only a very poor representation of the data because most of the variation is due to the poor alignment.

We construct a generative model that first draws an *aligned* image $\mathbf{x}'$ from a normal distribution, and then translates it using one of a discrete set $\{\text{trans}_k[\bullet]\}_{k=1}^{K}$ of K possible transformations to explain the poorly aligned image $\mathbf{x}$. In mathematical terms, we have

$$\begin{aligned} Pr(\mathbf{x}') &= \text{Norm}_{\mathbf{x}'}[\boldsymbol{\mu}, \boldsymbol{\Sigma}] \\ Pr(h) &= \text{Cat}_h[\lambda] \\ \mathbf{x} &= \text{trans}_h[\mathbf{x}'] \end{aligned} \tag{7.62}$$

where $h \in \{1, \ldots, K\}$ is a hidden variable that denotes which of the possible transformations warped this example.

This model can be learned using an EM-like procedure. In the E-step, we compute a probability distribution $Pr(h_i|\mathbf{x}_i)$ over the hidden variables by applying all of the inverse transformations to each example and evaluating how likely the result is under the current parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. In the M-step, we update these parameters by taking weighted sums of the inverse transformed images.

Summary

In this chapter we have introduced the idea of the *hidden variable* to induce structure in density models. The main approach to learning models of this kind is the expectation maximization algorithm. This is an iterative approach that is only guaranteed to find a local maximum. We have seen that although these models are more sophisticated than the normal distribution, they are still not really good representations of the density of high-dimensional visual data.

Notes

Expectation maximization: The EM algorithm was originally described by Dempster *et al.* (1977) although the presentation in this chapter owes more to the perspective espoused in Neal & Hinton (1999). A comprehensive summary of the EM algorithm and its extensions can be found in McLachlan & Krishnan (2008).

Mixtures of Gaussians: The mixtures of Gaussians model is closely related to the *K-means* algorithm (discussed in section 13.4.4), which is a pure clustering algorithm but does not have a probabilistic interpretation. Mixture models are used extensively in computer vision. Common applications include skin detection (e.g., Jones & Rehg 2002) and background subtraction (e.g., Stauffer & Grimson 1999).

t-distributions: General information about the t distribution can be found in Kotz & Nadarajah (2004). The EM algorithm for fitting the t-distribution is given in Liu & Rubin (1995), and other fitting methods are discussed in Nadarajah & Kotz (2008) and Aeschliman *et al.* (2010). Applications of t-distributions in vision include object recognition (Aeschliman *et al.* 2010) and tracking (Loxam & Drummond 2008; Aeschliman *et al.* 2010), and they are also used as the building blocks of sparse image priors (Roth & Black 2009).

Subspace models: The EM algorithm to learn the factor analyzer is due to Rubin & Thayer (1982). Factor analysis is closely related to several other models including probabilistic principal component analysis (Tipping & Bishop 1999), which is discussed in section 17.5.1 and principal component analysis which is discussed in section 13.4.2. Subspace models have been extended to the nonlinear case by Lawrence (2004). This is discussed in detail in section 17.8. In chapter 18 we present a series of models based on factor analysis, which explicitly encode the identity and style of the object.

Combining models: Ghahramani & Hinton (1996c) introduced the mixtures of factor analyzers model. Peel & McLachlan (2000) present an algorithm for learning robust mixture models (mixtures of t-distributions). Khan & Dellaert (2004) and Zhao & Jiang (2006) both present subspace models based on the t-distribution. De Ridder & Franc (2003) combined mixture models, subspace models and t-distributions to create a distribution that is mutli-modal, robust and oriented along a subspace.

Face detection, face recognition, object recognition: Models based on subspace distributions were used in early methods for face and object recognition (e.g., Moghaddam & Pentland 1997; Murase & Nayar 1995). Modern face detection methods mainly rely on discriminative methods (chapter 9), and the current state of the art in object recognition relies on bag of words approaches (chapter 20). Face recognition applications do not usually have the luxury of having many examples of each individual and so cannot build a separate density for each. However, modern approaches are still largely based on subspace methods (see chapter 18).

Segmentation: Belongie *et al.* (1998) use a mixtures of Gaussians scheme similar to that described to segment the image as part of a content-based image retrieval system. A modern approach to segmentation based on the mixture of Gaussians model can be found in Ma *et al.* (2007). Sfikas *et al.* (2007) compared the segmentation performance of mixtures of Gaussians and mixtures of t-distributions.

Other uses for hidden variables: Frey & Jojic (1999a) and Frey & Jojic (1999b) used hidden variables to model unseen transformations applied to data from mixture models and subspace models, respectively. Jojic & Frey (2001) used discrete hidden variables to represent the index of the layer in a multi-layered model of a video. Jojic *et al.*

(2003) presented a structured mixture model, where the mean and variance parameters are represented as images and the hidden variable indexes the starting position of sub-patch from this image.

Problems

Problem 7.1 Consider a computer vision system for machine inspection of oranges in which the goal is to tell if the orange is ripe. For each image we separate the orange from the background and calculate the average color of the pixels, which we describe as a 3×1 vector $\mathbf{x}$. We are given training pairs $\{\mathbf{x}_i, w_i\}$ of these vectors, each with an associated binary variable $w \in \{0, 1\}$ that indicates that this training example was unripe ($w = 0$) or ripe ($w = 1$). Describe how to build a generative classifier that could classify new examples $\mathbf{x}^*$ as being ripe or unripe.

Problem 7.2 It turns out that a small subset of the training labels w_i in the previous example were wrong. How could you modify your classifier to cope with this situation?

Problem 7.3 Derive the M-step equations for the mixtures of Gaussians model (equation 7.19).

Problem 7.4 Consider modeling some univariate continuous visual data $x \in [0, 1]$ using a *mixture of beta distributions*. Write down an equation for this model. Describe in words what will occur in (i) the E-step and (ii) the M-step.

Problem 7.5 Prove that the student t-distribution over x is the marginalization with respect to h of the joint distribution $Pr(x, h)$ between x and a hidden variable h where

$$\text{Stud}_x[\mu, \sigma^2, \nu] = \int \text{Norm}_x[\mu, \sigma^2/h]\text{Gam}_h[\nu/2, \nu/2]dh.$$

Problem 7.6 Show that the peak of the Gamma distribution $\text{Gam}_z[\alpha, \beta]$ is at

$$\hat{z} = \frac{\alpha - 1}{\beta}.$$

Problem 7.7 Show that the Gamma distribution is conjugate to the inverse scaling factor of the variance in a normal distribution so that

$$\text{Norm}_{\mathbf{x}_i}[\boldsymbol{\mu}, \boldsymbol{\Sigma}/h_i]\text{Gam}_{h_i}[\nu/2, \nu/2] = \kappa\text{Gam}_{h_i}[\alpha, \beta],$$

and find the constant of proportionality κ and the new parameters α and β.

Problem 7.8 The model for factor analysis can be written as

$$\mathbf{x}_i = \boldsymbol{\mu} + \boldsymbol{\Phi}\mathbf{h}_i + \boldsymbol{\epsilon}_i,$$

where $\mathbf{h}_i$ is distributed normally with mean zero and identity covariance and $\boldsymbol{\epsilon}_i$ is distributed normally with mean zero and covariance $\boldsymbol{\Sigma}$. Determine expressions for

1. $E[\mathbf{x}_i]$,

2. $E[(\mathbf{x}_i - E[\mathbf{x}_i])(\mathbf{x}_i - E[\mathbf{x}_i])^T]$.

Problem 7.9 Derive the E-step for factor analysis (equation 7.34).

Problem 7.10 Derive the M-step for factor analysis (equation 7.38).

Chapter 8

Regression models

This chapter concerns regression problems: the goal is to estimate a univariate world state w based on observed measurements $\mathbf{x}$. The discussion is limited to discriminative methods in which the distribution $Pr(w|\mathbf{x})$ of the world state is directly modeled. This contrasts with chapter 7 where the focus was on generative models in which the likelihood $Pr(\mathbf{x}|w)$ of the observations is modeled.

To motivate the regression problem, consider *body pose estimation*: here the goal is to estimate the joint angles of a human body, based on an observed image of the person in an unknown pose (figure 8.1). Such an analysis could form the first step toward activity recognition.

We assume that the image has already been preprocessed and a low dimensional vector $\mathbf{x}$ that represents the shape of the contour has been extracted. Our goal is to use this data vector to predict a second vector containing the joint angles for each of the major body joints. In practice, we will estimate each joint angle separately; we can hence concentrate our discussion on how to estimate a univariate quantity w from continuous observed data $\mathbf{x}$. We begin by assuming that the relation between the world and the data is linear and that the uncertainty around this prediction is normally distributed with constant variance. This is the linear regression model.

8.1 Linear regression

The goal of linear regression is to predict the posterior distribution $Pr(w|\mathbf{x})$ over the world state w based on observed data $\mathbf{x}$. Since this is a discriminative model, we proceed by choosing a probability distribution over the world w and making the parameters dependent on the data $\mathbf{x}$. The world state w is univariate and continuous and so a suitable distribution is the univariate normal. In linear regression (figure 8.2), we make the mean μ of this normal distribution a linear function $\phi_0 + \boldsymbol{\phi}^T\mathbf{x}_i$ of the data and treat the variance σ^2 as a constant so that

$$Pr(w_i|\mathbf{x}_i, \boldsymbol{\theta}) = \text{Norm}_{w_i}\left[\phi_0 + \boldsymbol{\phi}^T\mathbf{x}_i, \sigma^2\right], \tag{8.1}$$

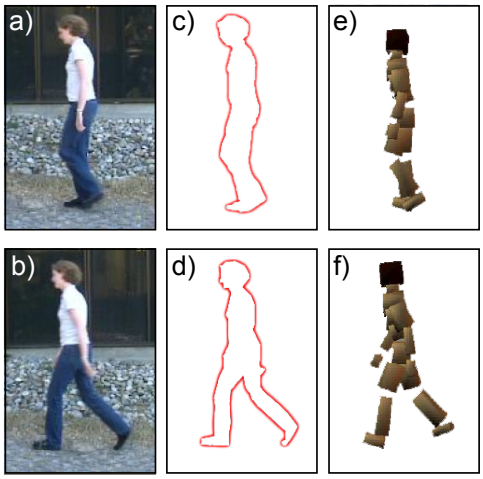


Figure 8.1 Body pose estimation. a–b) Human beings in unknown poses. c–d) The silhouette is found by segmenting the image and the contour extracted by tracing around the edge of the silhouette. A 100D $\mathbf{x}$ is extracted that describes the contour shape based on the shape context descriptor (see section 13.3.5). e–f) The goal is to estimate the vector $\mathbf{w}$ containing the major joint angles of the body. This is a regression problem as each element of the world state $\mathbf{w}$ is continuous. Adapted from Agarwal & Triggs (2006).

where $\boldsymbol{\theta} = \{\phi_0, \boldsymbol{\phi}, \sigma^2\}$ are the model parameters. The term ϕ_0 can be interpreted as the y-intercept of a hyperplane and the entries of $\boldsymbol{\phi} = [\phi_1, \phi_2, \ldots, \phi_D]^T$ are its gradients with respect to each of the D data dimensions.

It is cumbersome to treat the y-intercept separately from the gradients, so we apply a trick that allows us to simplify the subsequent notation. We attach a 1 to the start of every data vector $\mathbf{x}_i \leftarrow [1 \quad \mathbf{x}_i^T]^T$ and attach the y-intercept ϕ_0 to the start of the gradient vector $\boldsymbol{\phi} \leftarrow [\phi_0 \quad \boldsymbol{\phi}^T]^T$ so that we can now equivalently write

$$Pr(w_i|\mathbf{x}_i, \boldsymbol{\theta}) = \text{Norm}_{w_i}\left[\boldsymbol{\phi}^T\mathbf{x}_i, \sigma^2\right]. \tag{8.2}$$

In fact, since each training data example is considered independent, we can write the probability $Pr(\mathbf{w}|\mathbf{X})$ of the entire training set as a single normal distribution with a diagonal covariance so that

$$Pr(\mathbf{w}|\mathbf{X}) = \text{Norm}_{\mathbf{w}}[\mathbf{X}^T\boldsymbol{\phi}, \sigma^2\mathbf{I}], \tag{8.3}$$

where $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_I]$ and $\mathbf{w} = [w_1, w_2, \ldots, w_I]^T$.

Inference for this model is very simple: for a new datum $\mathbf{x}^*$ we simply evaluate equation 8.2 to find the posterior distribution $Pr(w^*|\mathbf{x}^*)$ over the world state w^*. Hence we turn our main focus to learning.

8.1.1 Learning

Algorithm 7.1

The learning algorithm estimates the model parameters $\boldsymbol{\theta} = \{\boldsymbol{\phi}, \sigma^2\}$ from paired training examples $\{\mathbf{x}_i, w_i\}_{i=1}^I$. In the maximum likelihood approach we seek

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}}{\text{argmax}}\left[Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\theta})\right] \\ &= \underset{\boldsymbol{\theta}}{\text{argmax}}\left[\log[Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\theta})]\right],\end{aligned} \tag{8.4}$$

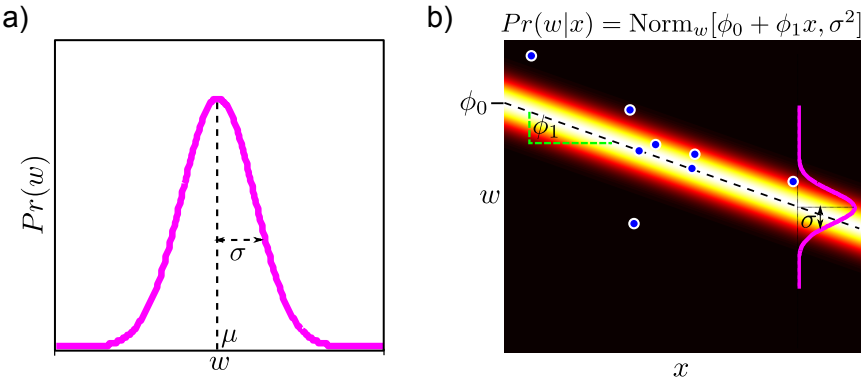


Figure 8.2 Linear regression model with univariate data x. a) We choose a univariate normal distribution over the world state w. b) The parameters of this distribution are now made to depend on the data x: the mean μ is a linear function $\phi_0 + \phi_1 x$ of the data and the variance σ^2 is constant. The parameters ϕ_0 and ϕ_1 represent the intercept and slope of the linear function, respectively.

where as usual we have taken the logarithm of the criterion. The logarithm is a monotonic transformation, and so it does not change the position of the maximum, but the resulting cost function is easier to optimize. Substituting in we find that

$$\hat{\boldsymbol{\phi}}, \hat{\sigma}^2 = \underset{\boldsymbol{\phi},\sigma^2}{\text{argmax}} \left[-\frac{I\log[2\pi]}{2} - \frac{I\log[\sigma^2]}{2} - \frac{(\mathbf{w} - \mathbf{X}^T\boldsymbol{\phi})^T(\mathbf{w} - \mathbf{X}^T\boldsymbol{\phi})}{2\sigma^2} \right]. \tag{8.5}$$

We now take the derivatives with respect to $\boldsymbol{\phi}$ and σ^2, equate the resulting expressions to zero and solve to find

Problem 8.3

$$\begin{aligned} \hat{\boldsymbol{\phi}} &= (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{w} \\ \hat{\sigma}^2 &= \frac{(\mathbf{w} - \mathbf{X}^T\boldsymbol{\phi})^T(\mathbf{w} - \mathbf{X}^T\boldsymbol{\phi})}{I}. \end{aligned} \tag{8.6}$$

Figure 8.2b shows an example fit with univariate data x. In this case, the model describes the data reasonably well.

8.1.2 Problems with the linear regression model

There are three main limitations of the linear regression model.

- The predictions of the model are overconfident; for example, small changes in the estimated slope ϕ_1 make increasingly large changes in the predictions as we move further from the y-intercept ϕ_0. However, this is not reflected in the posterior distribution.

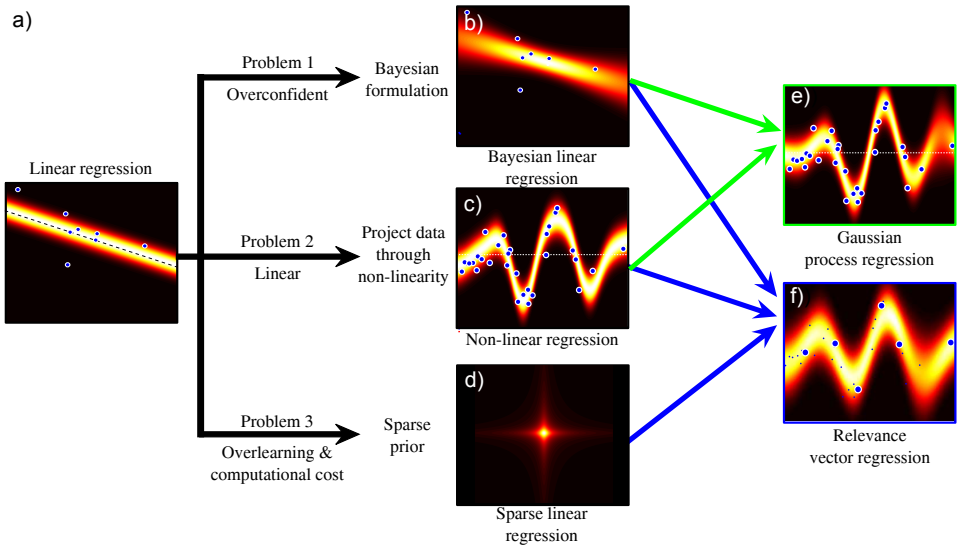


Figure 8.3 Family of regression models. There are several limitations to linear regression which we deal with in subsequent sections. The linear regression model with maximum likelihood learning is overconfident, and hence we develop a Bayesian version. It is unrealistic to always assume a linear relationship between the data and the world and to this end, we introduce a nonlinear version. The linear regression model has many parameters when the data dimension is high, and hence we consider a sparse version of the model. The ideas of Bayesian estimation, nonlinear functions and sparsity are variously combined to form the Gaussian process regression and relevance vector regression model.

- We are limited to linear functions and usually there is no particular reason that the visual data and world state should be linearly related.

- When the observed data $\mathbf{x}$ is high=dimensional, it may be that many elements of this variable aren't useful for predicting the state of the world, and so the resulting model is unnecessarily complex.

We tackle each of these problems in turn. In the following section we address the overconfidence of the model by developing a Bayesian approach to the same problem. In section 8.3, we generalize this model to fit nonlinear functions. In section 8.6, we introduce a sparse version of the regression model where most of the weighting coefficients $\boldsymbol{\phi}$ are encouraged to be zero. The relationships between the models in this chapter are indicated in figure 8.3.

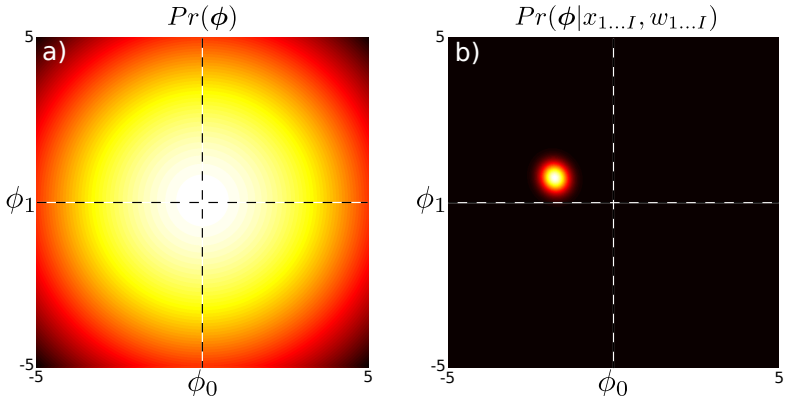


Figure 8.4 Bayesian linear regression. a) Prior $Pr(\boldsymbol{\phi})$ over intercept ϕ_0 and slope ϕ_1 parameters. This represents our knowledge about the parameters before we observe the data. b) Posterior distribution $Pr(\boldsymbol{\phi}|\mathbf{X}, \mathbf{w})$ over intercept and slope parameters. This represents our knowledge about the parameters after observing the data from figure 8.2b: we are considerably more certain but there remain a range of possible parameter values.

8.2 Bayesian linear regression

In the Bayesian approach, we compute a probability distribution over possible values of the parameters $\boldsymbol{\phi}$ (we will assume for now that σ^2 is known, see section 8.2.2). When we evaluate the probability of new data, we take a weighted average of the predictions induced by the different possible values.

Algorithm 7.2

Since the gradient vector $\boldsymbol{\phi}$ is multivariate and continuous, we model the prior $Pr(\boldsymbol{\phi})$ as normal with zero mean and spherical covariance,

$$Pr(\boldsymbol{\phi}) = \text{Norm}_{\boldsymbol{\phi}}[\mathbf{0}, \sigma_p^2\mathbf{I}], \tag{8.7}$$

where σ_p^2 scales the prior covariance and $\mathbf{I}$ is the identity matrix. Typically σ_p^2 is set to a large value to reflect the fact that our prior knowledge is weak.

Given paired training examples $\{\mathbf{x}_i, w_i\}_{i=1}^I$, the posterior distribution over the parameters can be computed using Bayes' rule

$$Pr(\boldsymbol{\phi}|\mathbf{X}, \mathbf{w}) = \frac{Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\phi})Pr(\boldsymbol{\phi})}{Pr(\mathbf{w}|\mathbf{X})}, \tag{8.8}$$

where, as before, the likelihood is given by

$$Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\theta}) \quad = \quad \text{Norm}_{\mathbf{w}}\left[\mathbf{X}^T\boldsymbol{\phi}, \sigma^2\mathbf{I}\right]. \tag{8.9}$$

The posterior distribution can be computed in closed form (using the relations in sections 5.7 and 5.6) and is given by the expression:

Problem 8.4

$$Pr(\boldsymbol{\phi}|\mathbf{X},\mathbf{w}) = \text{Norm}_{\boldsymbol{\phi}}\left[\frac{1}{\sigma^2}\mathbf{A}^{-1}\mathbf{X}\mathbf{w}, \mathbf{A}^{-1}\right], \tag{8.10}$$

where

$$\mathbf{A} = \frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T + \frac{1}{\sigma_p^2}\mathbf{I}. \tag{8.11}$$

Note that the posterior distribution $Pr(\boldsymbol{\phi}|\mathbf{X},\mathbf{w})$ is always narrower than the prior distribution $Pr(\boldsymbol{\phi})$ (figure 8.4); the data provides information that refines our knowledge of the parameter values.

Problem 8.5

We now turn to the problem of computing the predictive distribution over the world state w^* for a new observed data vector $\mathbf{x}^*$. We take an infinite weighted sum (i.e., an integral) over the predictions $Pr(w^*|\mathbf{x}^*,\boldsymbol{\phi})$ implied by each possible $\boldsymbol{\phi}$ where the weights are given by the posterior distribution $Pr(\boldsymbol{\phi}|\mathbf{X},\mathbf{w})$.

$$\begin{aligned} Pr(w^*|\mathbf{x}^*,\mathbf{X},\mathbf{w}) &= \int Pr(w^*|\mathbf{x}^*,\boldsymbol{\phi})Pr(\boldsymbol{\phi}|\mathbf{X},\mathbf{w})d\boldsymbol{\phi} \\ &= \int \text{Norm}_{w^*}[\boldsymbol{\phi}^T\mathbf{x}^*, \sigma^2]\text{Norm}_{\boldsymbol{\phi}}\left[\frac{1}{\sigma^2}\mathbf{A}^{-1}\mathbf{X}\mathbf{w}, \mathbf{A}^{-1}\right]d\boldsymbol{\phi} \\ &= \text{Norm}_{w^*}\left[\frac{1}{\sigma^2}\mathbf{x}^{*T}\mathbf{A}^{-1}\mathbf{X}\mathbf{w}, \mathbf{x}^{*T}\mathbf{A}^{-1}\mathbf{x}^* + \sigma^2\right]. \end{aligned} \tag{8.12}$$

To compute this, we reformulated the integrand using the relations from sections 5.7 and 5.6 as the product of a normal distribution in $\boldsymbol{\phi}$ and a constant with respect to $\boldsymbol{\phi}$. The integral of the normal distribution must be one, and so the final result is just the constant. This constant is itself a normal distribution in w^*.

This Bayesian formulation of linear regression (figure 8.5) is less confident about its predictions, and the confidence decreases as the test data $\mathbf{x}^*$ departs from the mean $\bar{\mathbf{x}}$ of the observed data. This is because uncertainty in the gradient causes increasing uncertainty in the predictions as we move further away from the bulk of the data. This agrees with our intuitions: predictions ought to become less confident as we extrapolate further from the data.

8.2.1 Practical concerns

To implement this model we must compute the $D \times D$ matrix inverse $\mathbf{A}^{-1}$ (equation 8.12). If the dimension D of the original data is large, then it will be difficult to compute this inverse directly.

Fortunately, the structure of $\mathbf{A}$ is such that it can be inverted far more efficiently. We exploit the Woodbury identity (see appendix C.8.4), to rewrite $\mathbf{A}^{-1}$ as

$$\mathbf{A}^{-1} = \left(\frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T + \frac{1}{\sigma_p^2}\mathbf{I}_D\right)^{-1} = \sigma_p^2\mathbf{I}_D - \sigma_p^2\mathbf{X}\left(\mathbf{X}^T\mathbf{X} + \frac{\sigma^2}{\sigma_p^2}\mathbf{I}_I\right)^{-1}\mathbf{X}^T, \tag{8.13}$$

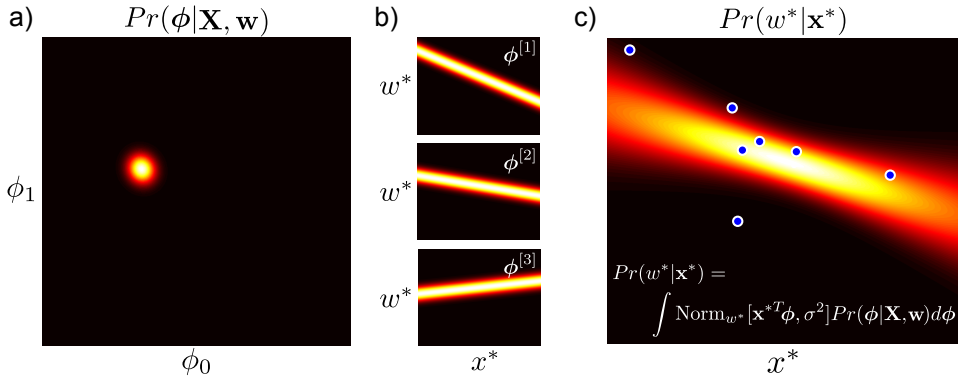


Figure 8.5 Bayesian linear regression. a) In learning we compute the posterior distribution $Pr(\boldsymbol{\phi}|\mathbf{X},\mathbf{w})$ over the intercept and slope parameters: there is a family of parameter settings that are compatible with the data. b) Three samples from the posterior, each of which corresponds to a different regression line. c) To form the predictive distribution we take an infinite weighted sum (integral) of the predictions from all of the possible parameter settings, where the weight is given by the posterior probability. The individual predictions vary more as we move from the centroid $\bar{\mathbf{x}}$ and this is reflected in the fact that the certainty is lower on either side of the plot.

where we have explicitly noted the dimensionality of each of the identity matrices $\mathbf{I}$ as a subscript. The expression still includes an inversion, but now it is of size $I \times I$ where I is the number of examples. If the number of examples I is fewer than the number of data dimensions D, then this formulation is more practical. This formulation also demonstrates clearly that the posterior covariance is less than the prior; the posterior covariance is the prior covariance σ_p^2 with a data-dependent term subtracted from it.

Substituting the new expression for $\mathbf{A}^{-1}$ into equation 8.12, we derive a new expression for the predictive distribution,

$$Pr(w^*|\mathbf{x}^*,\mathbf{X},\mathbf{w}) = \tag{8.14}$$
$$\text{Norm}_{w^*}\left[\frac{\sigma_p^2}{\sigma^2}\mathbf{x}^{*T}\mathbf{X}\mathbf{w} - \frac{\sigma_p^2}{\sigma^2}\mathbf{x}^{*T}\mathbf{X}\left(\mathbf{X}^T\mathbf{X}+\frac{\sigma^2}{\sigma_p^2}\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{X}\mathbf{w},\right.$$
$$\left.\sigma_p^2\mathbf{x}^{*T}\mathbf{x}^* - \sigma_p^2\mathbf{x}^{*T}\mathbf{X}\left(\mathbf{X}^T\mathbf{X}+\frac{\sigma^2}{\sigma_p^2}\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{x}^* + \sigma^2\right].$$

It is notable that only inner products of the data vectors (e.g., in the terms $\mathbf{X}^T\mathbf{x}^*$, or $\mathbf{X}^T\mathbf{X}$) are required to compute this expression. We will exploit this fact when we generalize these ideas to nonlinear regression (section 8.3).

8.2.2 Fitting the variance

Problem 8.7

The previous analysis has concentrated exclusively on the slope parameters $\boldsymbol{\phi}$. In principle, we could have taken a Bayesian approach to estimating the variance parameter σ^2 as well. However, for simplicity we will compute a point estimate of σ^2 using the maximum likelihood approach. To this end, we optimize the *marginal likelihood*, which is the likelihood after marginalizing out $\boldsymbol{\phi}$ and is given by

$$\begin{aligned} Pr(\mathbf{w}|\mathbf{X},\sigma^2) &= \int Pr(\mathbf{w}|\mathbf{X},\boldsymbol{\phi},\sigma^2)Pr(\boldsymbol{\phi})\, d\boldsymbol{\phi} \\ &= \int \text{Norm}_{\mathbf{w}}[\mathbf{X}^T\boldsymbol{\phi},\sigma^2\mathbf{I}]\text{Norm}_{\boldsymbol{\phi}}[\mathbf{0},\sigma_p^2\mathbf{I}]\, d\boldsymbol{\phi} \\ &= \text{Norm}_{\mathbf{w}}[\mathbf{0},\sigma_p^2\mathbf{X}^T\mathbf{X}+\sigma^2\mathbf{I}] \end{aligned} \tag{8.15}$$

where the integral was solved using the same technique as for equation 8.12.

To estimate the variance, we maximize the log of this expression with respect to σ^2. Since the unknown is a scalar it is straightforward to optimize this function by simply evaluating the function over a range of values and choosing the maximum. Alternatively, we could use a general purpose nonlinear optimization technique (see appendix B).

8.3 Non-linear regression

Problem 8.9

It is unrealistic to assume that there is always a linear relationship between the world state w and the input data $\mathbf{x}$. In developing an approach to nonlinear regression, we would like to retain the mathematical convenience of the linear model while extending the class of functions that can be described.

Consequently, the approach that we describe is extremely simple: we first pass each data example through a nonlinear transformation

$$\mathbf{z}_i = \mathbf{f}[\mathbf{x}_i], \tag{8.16}$$

to create a new data vector $\mathbf{z}_i$ which is usually higher dimensional than the original data. Then we proceed as before: we describe the mean of the posterior distribution $Pr(w_i|\mathbf{x}_i)$ as a linear function $\boldsymbol{\phi}^T\mathbf{z}_i$ of the transformed measurements so that

$$Pr(w_i|\mathbf{x}_i,\boldsymbol{\theta}) = \text{Norm}_{w_i}[\boldsymbol{\phi}^T\mathbf{z}_i,\sigma^2]. \tag{8.17}$$

For example, consider the case of 1D polynomial regression:

$$Pr(w_i|x_i) = \text{Norm}_{w_i}[\phi_0+\phi_1 x_i+\phi_2 x_i^2+\phi_3 x_i^3,\sigma^2]. \tag{8.18}$$

This model can be considered as computing the nonlinear transformation

$$\mathbf{z}_i = \begin{bmatrix} 1 \\ x_i \\ x_i^2 \\ x_i^3 \end{bmatrix}, \tag{8.19}$$

and so it has the general form of equation 8.17.

8.3.1 Maximum likelihood

To find the maximum likelihood solution for the gradient vector $\boldsymbol{\phi}$ we first combine all of the transformed training data relations (equation 8.17) into a single expression:

$$Pr(\mathbf{w}|\mathbf{X}) = \text{Norm}_{\mathbf{w}}[\mathbf{Z}^T\boldsymbol{\phi}, \sigma^2\mathbf{I}]. \tag{8.20}$$

The optimal weights can now be computed as

$$\begin{aligned} \hat{\boldsymbol{\phi}} &= (\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}\mathbf{w} \\ \hat{\sigma}^2 &= \frac{(\mathbf{w}-\mathbf{Z}^T\boldsymbol{\phi})^T(\mathbf{w}-\mathbf{Z}^T\boldsymbol{\phi})}{I}, \end{aligned} \tag{8.21}$$

where the matrix $\mathbf{Z}$ contains the transformed vectors $\{\mathbf{z}_i\}_{i=1}^I$ in its columns. These equations were derived by replacing the original data term $\mathbf{X}$ by the transformed data $\mathbf{Z}$ in the equivalent linear expressions (equation 8.6). For a new observed data example $\mathbf{x}^*$ we compute the vector $\mathbf{z}^*$ and then evaluate equation 8.17.

Figures 8.6 and 8.7 provide two more examples of this approach. In figure 8.6, the new vector $\mathbf{z}$ is computed by evaluating the data $\mathbf{x}$ under a set of radial basis functions:

$$\mathbf{z}_i = \begin{bmatrix} 1 \\ \exp\left[-(x_i-\alpha_1)^2/\lambda\right] \\ \exp\left[-(x_i-\alpha_2)^2/\lambda\right] \\ \exp\left[-(x_i-\alpha_3)^2/\lambda\right] \\ \exp\left[-(x_i-\alpha_4)^2/\lambda\right] \\ \exp\left[-(x_i-\alpha_5)^2/\lambda\right] \\ \exp\left[-(x_i-\alpha_6)^2/\lambda\right] \end{bmatrix}. \tag{8.22}$$

The term *radial basis functions* can be used to denote any spherically symmetric function, and here we have used the Gaussian. The parameters $\{\alpha_k\}_{k=1}^K$ are the centers of the functions, and λ is a scaling factor that determines their width. The functions themselves are shown in figure 8.6b. Because they are spatially localized, each one accounts for a part of the original data space. We can approximate a function by taking weighted sums $\boldsymbol{\phi}^T\mathbf{z}$ of these functions. For example, when they are weighted as in figure 8.6c, they create the function in figure 8.6a.

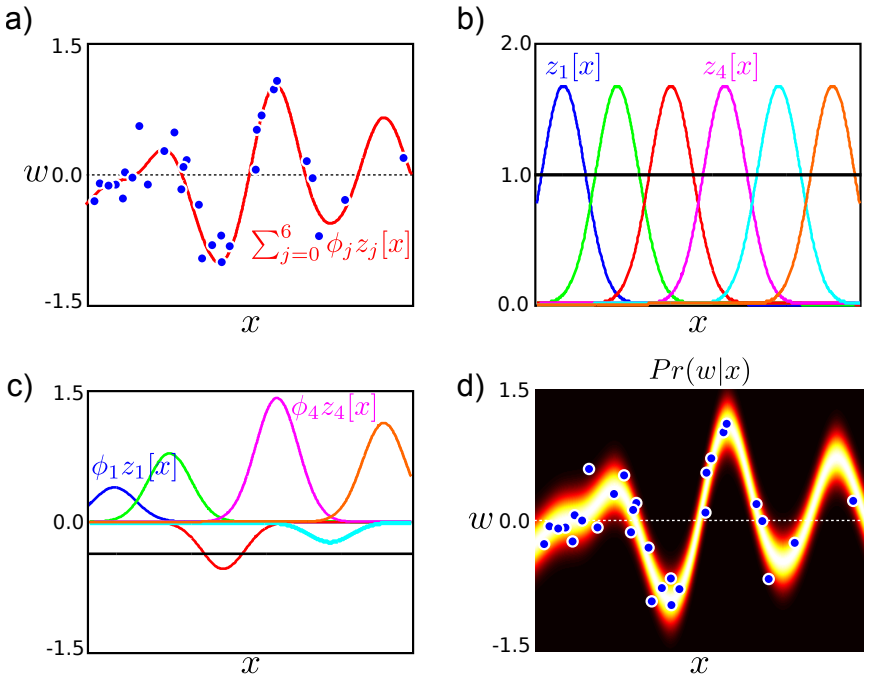


Figure 8.6 Non-linear regression using radial basis functions. a) The relationship between the data x and world w is clearly not linear. b) We compute a new seven dimensional vector $\mathbf{z}$ by evaluating the original observation x against each of six radial basis functions (Gaussians) and a constant function (black line). c) The mean of the predictive distribution (red line in (a)) can be formed by taking a linear sum $\boldsymbol{\phi}^T\mathbf{z}$ of these seven functions where the weights are as shown. The weights are estimated by maximum likelihood estimation of the linear regression model using the nonlinearly transformed data $\mathbf{z}$ instead of the original data $\mathbf{x}$. d) The final distribution $Pr(w|x)$ has a mean that is a sum of these functions and constant variance σ^2.

In figure 8.7 we compute a different nonlinear transformation and regress against the same data. This time, the transformation is based on arc tangent functions so that

$$\mathbf{z}_i = \begin{bmatrix} \arctan[\lambda x_i - \alpha_1] \\ \arctan[\lambda x_i - \alpha_2] \\ \arctan[\lambda x_i - \alpha_3] \\ \arctan[\lambda x_i - \alpha_4] \\ \arctan[\lambda x_i - \alpha_5] \\ \arctan[\lambda x_i - \alpha_6] \\ \arctan[\lambda x_i - \alpha_7] \end{bmatrix}. \tag{8.23}$$

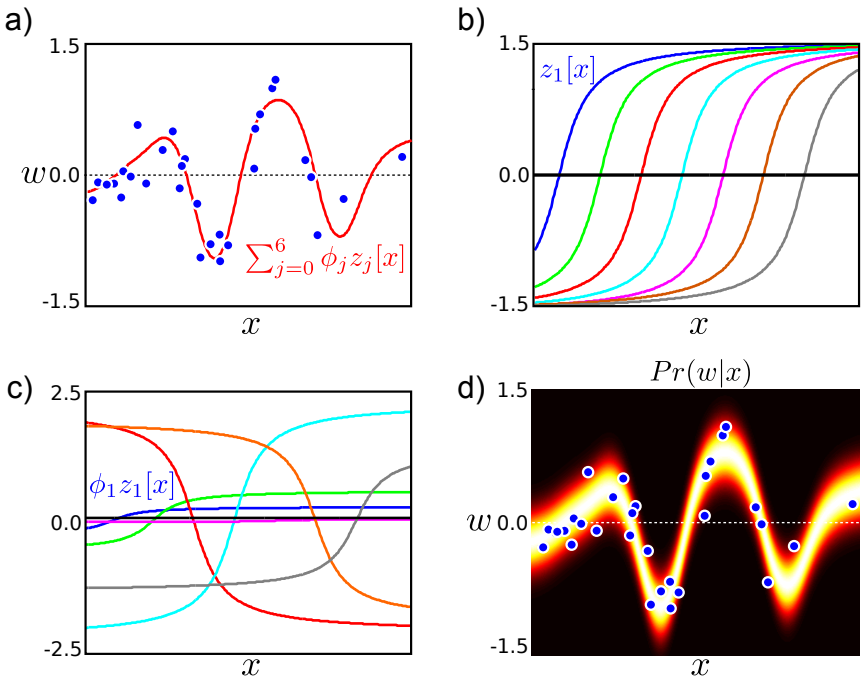


Figure 8.7 Non-linear regression using arc tangent functions. a) The relationship between the data x and world w is not linear. b) We compute a new seven dimensional vector $\mathbf{z}$ by evaluating the original observation x against each of seven arc tangent functions. c) The mean of the predictive distribution (red line in (a)) can be formed by taking a linear sum of these seven functions weighted as shown. The optimal weights were established using the maximum likelihood approach. d) The final distribution $Pr(w|x)$ has a mean that is a sum of these weighted functions and constant variance.

Here, the parameter λ controls the speed with which the function changes, and the parameters $\{\alpha_m\}_{m=1}^{7}$ determine the horizontal offsets of the arc tangent functions.

In this case, it is harder to understand the role of each weighted arc tangent function in the final regression, but nonetheless they collectively approximate the function well.

8.3.2 Bayesian nonlinear regression

In the Bayesian solution, the weights $\boldsymbol{\phi}$ of the nonlinear basis functions are treated as uncertain: in learning we compute the posterior distribution over these weights. For a new observation $\mathbf{x}^*$, we compute the transformed vector $\mathbf{z}^*$ and compute an infinite weighted sum over the predictions due to the possible parameter values (figure 8.8). The new expression for the predictive distribution is

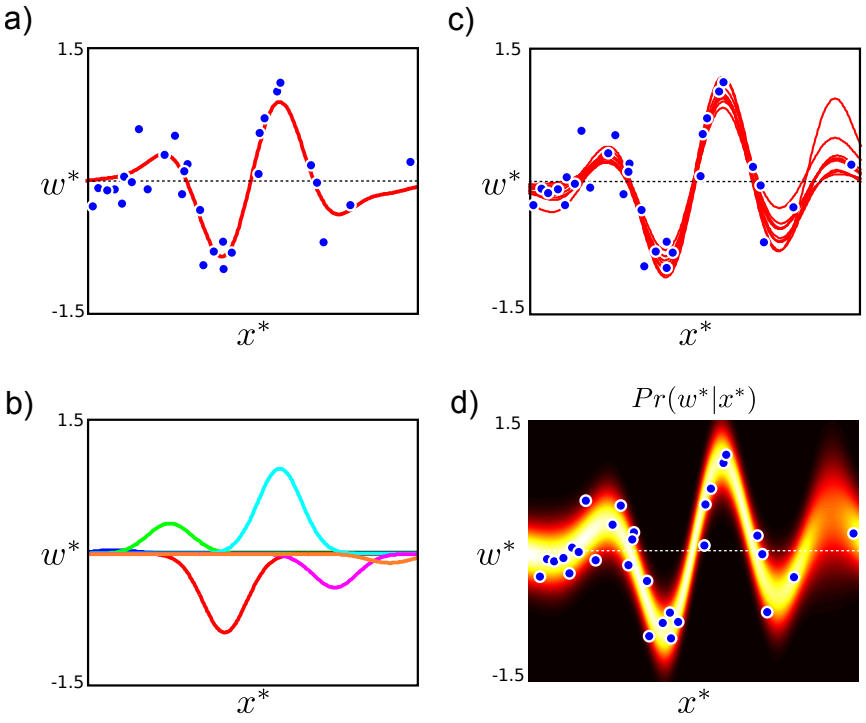


Figure 8.8 Bayesian nonlinear regression using radial basis functions. a) The relationship between the data and measurements is nonlinear. b) As in figure 8.6, the mean of the predictive distribution is constructed as a weighted linear sum of radial basis functions. However in the Bayesian approach we compute the posterior distribution over the weights $\boldsymbol{\phi}$ of these basis functions. c) Different draws from this distribution of weight parameters result in different predictions. d) The final predictive distribution is formed from an infinite weighted average of these predictions where the weight is given by the posterior probability. The variance of the predictive distribution depends on both the mutual agreement of these predictions and the noise σ^2. The uncertainty is greatest in the region on the right where there is little data and so the individual predictions vary widely.

$$Pr(w^*|\mathbf{z}^*, \mathbf{X}, \mathbf{w}) = \tag{8.24}$$
$$\text{Norm}_w\left[\frac{\sigma_p^2}{\sigma^2}\mathbf{z}^{*T}\mathbf{Z}\mathbf{w} - \frac{\sigma_p^2}{\sigma^2}\mathbf{z}^{*T}\mathbf{Z}\left(\mathbf{Z}^T\mathbf{Z} + \frac{\sigma^2}{\sigma_p^2}\mathbf{I}\right)^{-1}\mathbf{Z}^T\mathbf{Z}\mathbf{w},\right.$$
$$\left.\sigma_p^2\mathbf{z}^{*T}\mathbf{z}^* - \sigma_p^2\mathbf{z}^{*T}\mathbf{Z}\left(\mathbf{Z}^T\mathbf{Z} + \frac{\sigma^2}{\sigma_p^2}\mathbf{I}\right)^{-1}\mathbf{Z}^T\mathbf{z}^* + \sigma^2\right],$$

where we have simply substituted the transformed vectors $\mathbf{z}$ for the original data

$\mathbf{x}$ in equation 8.14. The prediction variance depends on both the uncertainty in $\boldsymbol{\phi}$ and the additive variance σ^2. The Bayesian solution is less confident than the maximum likelihood solution (compare figures 8.8d and 8.7d), especially in regions where the data are sparse.

To compute the additive variance σ^2 we again optimize the marginal likelihood. The expression for this can be found by substituting $\mathbf{Z}$ for $\mathbf{X}$ in equation 8.15.

8.4 Kernels and the kernel trick

The Bayesian approach to nonlinear regression described in the previous section is rarely used directly in practice: the final expression for the predictive distribution (equation 8.24) relies on computing inner products $\mathbf{z}_i^T\mathbf{z}_j$. However, when the transformed space is high-dimensional it may be costly to compute the vectors $\mathbf{z}_i = \mathbf{f}[\mathbf{x}_i]$ and $\mathbf{z}_j = \mathbf{f}[\mathbf{x}_j]$ explicitly and then compute the inner product $\mathbf{z}_i^T\mathbf{z}_j$.

An alternative approach is to use *kernel substitution* in which we directly define a single *kernel function* $\mathrm{k}[\mathbf{x}_i, \mathbf{x}_j]$ as a replacement for the operation $\mathbf{f}[\mathbf{x}_i]^T\mathbf{f}[\mathbf{x}_j]$. For many transformations $\mathbf{f}[\bullet]$ it is more efficient to evaluate the kernel function directly than to transform the variables separately and then compute the dot product.

Taking this idea one step further, it is possible to choose a kernel function $\mathrm{k}[\mathbf{x}_i, \mathbf{x}_j]$ with no knowledge of what transformation $\mathbf{f}[\bullet]$ that it corresponds to. When we use kernel functions, we no longer explicitly compute the transformed vector $\mathbf{z}$. One advantage of this is we can define kernel functions that correspond to projecting the data into very high dimensional or even infinite spaces. This is sometimes called the *kernel trick*.

Clearly, the kernel function must be carefully chosen so that it does in fact correspond to computing some function $\mathbf{z} = \mathbf{f}[\mathbf{x}]$ for each data vector and taking the inner product of the resulting values: for example, since $\mathbf{z}_i^T\mathbf{z}_j = \mathbf{z}_j^T\mathbf{z}_i$ the kernel function must treat its arguments symmetrically so that $\mathrm{k}[\mathbf{x}_i, \mathbf{x}_j] = \mathrm{k}[\mathbf{x}_j, \mathbf{x}_i]$.

More precisely, *Mercer's theorem* states that a kernel function is valid when the kernel's arguments are in a measurable space, and the kernel is positive semi-definite so that

$$\sum_{ij} \mathrm{k}[\mathbf{x}_i, \mathbf{x}_j] a_i a_j \geq 0 \tag{8.25}$$

for any finite subset $\{\mathbf{x}_n\}_{n=1}^N$ of vectors in the space and any real numbers $\{a_n\}_{n=1}^N$. Examples of valid kernels include:

- linear

$$\mathrm{k}[\mathbf{x}_i, \mathbf{x}_j] = \mathbf{x}_i^T\mathbf{x}_j, \tag{8.26}$$

- degree p polynomial

$$\mathrm{k}[\mathbf{x}_i, \mathbf{x}_j] = (\mathbf{x}_i^T\mathbf{x}_j + 1)^p, \tag{8.27}$$

- radial basis function (RBF) or Gaussian

$$\mathrm{k}[\mathbf{x}_i, \mathbf{x}_j] = \exp\left[-0.5\left(\frac{(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j)}{\lambda^2}\right)\right]. \tag{8.28}$$

The last of these is particularly interesting. It can be shown that this kernel function corresponds to computing *infinite* length vectors $\mathbf{z}$ and taking their dot product. The entries of $\mathbf{z}$ correspond to evaluating a radial basis function (figure 8.6b) at every possible point in the space of $\mathbf{x}$.

It is also possible to create new kernels by combining two or more existing kernels. For example, sums and products of valid kernels are guaranteed to be positive semi-definite and so are also valid kernels.

8.5 Gaussian process regression

Algorithm 8.3

We now replace the inner products $\mathbf{z}_i^T\mathbf{z}_j$ in the nonlinear regression algorithm (equation 8.24) with kernel functions. The resulting model is termed *Gaussian process regression*. The predictive distribution for a new datum $\mathbf{x}^*$ is

$$\begin{aligned} Pr(w^*|\mathbf{x}^*, \mathbf{X}, \mathbf{w}) = & \\ \mathrm{Norm}_{w^*}\Bigg[&\frac{\sigma_p^2}{\sigma^2}\mathbf{K}[\mathbf{x}^*, \mathbf{X}]\mathbf{w} - \frac{\sigma_p^2}{\sigma^2}\mathbf{K}[\mathbf{x}^*, \mathbf{X}]\left(\mathbf{K}[\mathbf{X}, \mathbf{X}] + \frac{\sigma^2}{\sigma_p^2}\mathbf{I}\right)^{-1}\mathbf{K}[\mathbf{X}, \mathbf{X}]\mathbf{w}, \\ &\sigma_p^2\mathbf{K}[\mathbf{x}^*, \mathbf{x}^*] - \sigma_p^2\mathbf{K}[\mathbf{x}^*, \mathbf{X}]\left(\mathbf{K}[\mathbf{X}, \mathbf{X}] + \frac{\sigma^2}{\sigma_p^2}\mathbf{I}\right)^{-1}\mathbf{K}[\mathbf{X}, \mathbf{x}^*] + \sigma^2\Bigg]. \end{aligned} \tag{8.29}$$

where the notation $\mathbf{K}[\mathbf{X}, \mathbf{X}]$ represents a matrix of dot products where element (i, j) is given by $\mathrm{k}[\mathbf{x}_i, \mathbf{x}_j]$.

Note that kernel functions may also contain parameters. For example, the RBF kernel (equation 8.28) takes the parameter λ, which determines the width of the underlying RBF functions and hence the smoothness of the function (figure 8.9). Kernel parameters such as λ can be learned by maximizing the marginal likelihood:

$$\begin{aligned} \hat{\lambda} &= \underset{\lambda}{\mathrm{argmax}}\left[Pr(\mathbf{w}|\mathbf{X}, \sigma^2)\right] \\ &= \underset{\lambda}{\mathrm{argmax}}\left[\int Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\phi}, \sigma^2)Pr(\boldsymbol{\phi})d\boldsymbol{\phi}\right] \\ &= \underset{\lambda}{\mathrm{argmax}}\left[\mathrm{Norm}_{\mathbf{w}}[\mathbf{0}, \sigma_p^2\mathbf{K}[\mathbf{X}, \mathbf{X}] + \sigma^2\mathbf{I}]\right]. \end{aligned} \tag{8.30}$$

This typically requires a nonlinear optimization procedure.

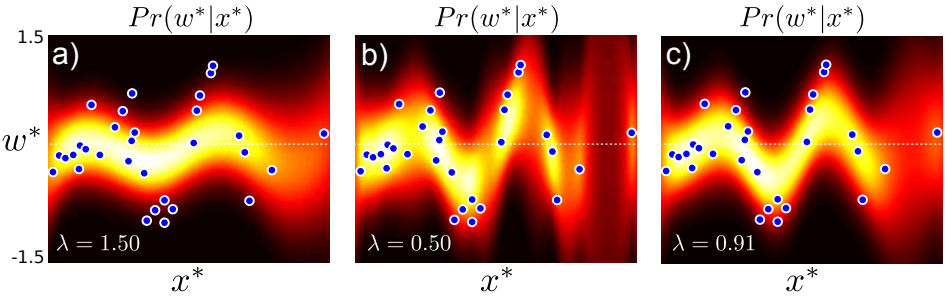


Figure 8.9 Gaussian process regression using an RBF kernel. a) When the length scale parameter λ is large, the function is too smooth. b) For small values of the length parameter the model does not successfully interpolate between the examples. c) The regression using the maximum likelihood length scale parameter is neither too smooth nor disjointed.

8.6 Sparse linear regression

We now turn our attention to the third potential disadvantage of linear regression. It is often the case that only a small subset of the dimensions of $\mathbf{x}$ are useful for predicting w. However, without modification, the linear regression algorithm will assign non-zero values to the gradient $\boldsymbol{\phi}$ in these directions. The goal of *sparse* linear regression is to adapt the algorithm to find a gradient vector $\boldsymbol{\phi}$ where most of the entries are zero. The resulting classifier will be faster, since we no longer even have to make all of the measurements. Furthermore, simpler models are preferable to complex ones; they capture the main trends in the data without over-fitting to peculiarities of the training set and generalize better to new test examples.

Algorithm 8.4

To encourage sparse solutions, we impose a penalty for every non-zero weighted data dimension. We replace the normal prior over the gradient parameters $\boldsymbol{\phi} = [\phi_1, \phi_2, \ldots, \phi_D]^T$ with a product of one-dimensional t-distributions so that

$$\begin{aligned} Pr(\boldsymbol{\phi}) &= \prod_{d=1}^{D} \text{Stud}_{\phi_d}[0, 1, \nu] \\ &= \prod_{d=1}^{D} \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)} \left(1 + \frac{\phi_d^2}{\nu}\right)^{-(\nu+1)/2}. \end{aligned} \tag{8.31}$$

The product of univariate t-distributions has ridges of high probability along the coordinate axes, which encourages sparseness (see figure 8.10). We expect the final solution to be a trade-off between fitting the training data accurately and the sparseness of $\boldsymbol{\phi}$ (and hence the number of training data dimensions that contribute to the solution).

Adopting the Bayesian approach, our aim is to compute the posterior distribution $Pr(\boldsymbol{\phi}|\mathbf{X}, \mathbf{w}, \sigma^2)$ over the possible values of the gradient variable $\boldsymbol{\phi}$ using this new prior so that

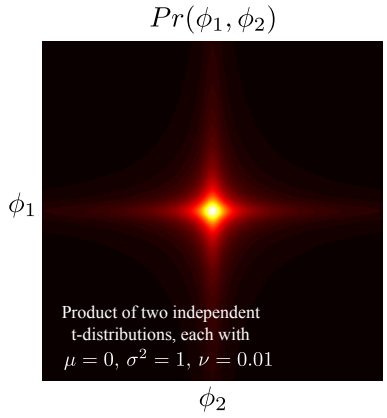


Figure 8.10 A product of two 1D t-distributions where each has small degrees of freedom ν. This 2D distribution favors sparseness (where one or both variables are close to zero). In higher dimensions, the product of t-distributions encourages solutions where *most* variables are set to zero. Note that the product of 1D distributions is *not* the same as a multivariate t-distribution with a spherical covariance matrix, which looks like a multivariate normal distribution but with longer tails.

$$Pr(\boldsymbol{\phi}|\mathbf{X}, \mathbf{w}, \sigma^2) = \frac{Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\phi}, \sigma^2)Pr(\boldsymbol{\phi})}{Pr(\mathbf{w}|\mathbf{X}, \sigma^2)}. \tag{8.32}$$

Unfortunately, there is no simple closed form expression for the posterior on the left hand side. The prior is no longer normal and the conjugacy relationship is lost.

To make progress, we re-express each t-distribution as an infinite weighted sum of normal distributions where a hidden variable h_d determines the variance (section 7.5), so that

$$\begin{aligned} Pr(\boldsymbol{\phi}) &= \prod_{d=1}^{D} \int \text{Norm}_{\phi_d}[0, 1/h_d]\text{Gam}_{h_d}[\nu/2, \nu/2]\, dh_d \\ &= \int \text{Norm}_{\boldsymbol{\phi}}[0, \mathbf{H}^{-1}] \prod_{d=1}^{D} \text{Gam}_{h_d}[\nu/2, \nu/2]\, d\mathbf{H}, \end{aligned} \tag{8.33}$$

where the matrix $\mathbf{H}$ contains the hidden variables $\{h_d\}_{d=1}^{D}$ on its diagonal and zeros elsewhere. We now write out the expression for the marginal likelihood (likelihood after integrating over the gradient parameters $\boldsymbol{\phi}$) as

$$\begin{aligned} Pr(\mathbf{w}|\mathbf{X}, \sigma^2) &\propto \int Pr(\mathbf{w}, \boldsymbol{\phi}|\mathbf{X}, \sigma^2)\, d\boldsymbol{\phi} \\ &= \int Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\phi}, \sigma^2)Pr(\boldsymbol{\phi})\, d\boldsymbol{\phi} \\ &= \int \text{Norm}_{\mathbf{w}}[\mathbf{X}^T\boldsymbol{\phi}, \sigma^2\mathbf{I}] \int \text{Norm}_{\boldsymbol{\phi}}[0, \mathbf{H}^{-1}] \prod_{d=1}^{D} \text{Gam}_{h_d}[\nu/2, \nu/2]\, d\mathbf{H}d\boldsymbol{\phi} \\ &= \int\int \text{Norm}_{\mathbf{w}}[\mathbf{X}^T\boldsymbol{\phi}, \sigma^2\mathbf{I}]\text{Norm}_{\boldsymbol{\phi}}[0, \mathbf{H}^{-1}] \prod_{d=1}^{D} \text{Gam}_{h_d}[\nu/2, \nu/2]\, d\mathbf{H}d\boldsymbol{\phi} \\ &= \int \text{Norm}_{\mathbf{w}}[\mathbf{0}, \mathbf{X}^T\mathbf{H}^{-1}\mathbf{X} + \sigma^2\mathbf{I}] \prod_{d=1}^{D} \text{Gam}_{h_d}[\nu/2, \nu/2]\, d\mathbf{H}. \end{aligned} \tag{8.34}$$

where we have computed the integral over $\boldsymbol{\phi}$ using the same method as in equation 8.12.

Unfortunately, we still cannot compute the remaining integral in closed form, so we instead take the approach of maximizing over hidden variables to give an approximate expression for the marginal likelihood

$$Pr(\mathbf{w}|\mathbf{X},\sigma^2) \approx \max_{\mathbf{H}}\left[\text{Norm}_{\mathbf{w}}[\mathbf{0}, \mathbf{X}^T\mathbf{H}^{-1}\mathbf{X}+\sigma^2\mathbf{I}]\prod_{d=1}^{D}\text{Gam}_{h_d}[\nu/2,\nu/2]\right]. \tag{8.35}$$

As long as the true distribution over the hidden variables is concentrated tightly around the mode, this will be a reasonable approximation. When h_d takes a large value, the prior has a small variance $(1/h_d)$, and the associated coefficient ϕ_d will be forced to be close to zero: in effect, this means that the d^{th} dimension of $\mathbf{x}$ does not contribute to the solution and can be dropped from the equations.

The general approach to fitting the model is now clear. There are two unknown quantities – the variance σ^2 and the hidden variables $\mathbf{h}$ and we alternately update each to maximize the log marginal likelihood.[1]

- To update the hidden variables, we take the derivative of the log of this expression with respect to $\mathbf{H}$, equate the result to zero, and re-arrange to get the iteration

$$h_d^{new} = \frac{1-h_d\Sigma_{dd}+\nu}{\mu_d^2+\nu}, \tag{8.36}$$

  where μ_d is the d^{th} element of the mean $\boldsymbol{\mu}$ of the posterior distribution over the weights $\boldsymbol{\phi}$ and Σ_{dd} is the d^{th} element of the diagonal of the covariance $\boldsymbol{\Sigma}$ of the posterior distribution over the weights (equation 8.10) so that

$$\begin{aligned}\boldsymbol{\mu} &= \frac{1}{\sigma^2}\mathbf{A}^{-1}\mathbf{X}\mathbf{w}\\ \boldsymbol{\Sigma} &= \mathbf{A}^{-1},\end{aligned} \tag{8.37}$$

  and $\mathbf{A}$ is defined as

$$\mathbf{A} = \frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T+\mathbf{H}. \tag{8.38}$$

- To update the variance, we take the derivative of the log of this expression with respect to σ^2, equate the result to zero, and simplify to get

$$(\sigma^2)^{new} = \frac{1}{D-\sum_d(1-h_d\Sigma_{dd})}(\mathbf{w}-\mathbf{X}\boldsymbol{\mu})^T(\mathbf{w}-\mathbf{X}\boldsymbol{\mu}). \tag{8.39}$$

[1] More details about how these (non-obvious) update equations were generated can be found in section 3.5 of Bishop (2006) and Tipping (2001).

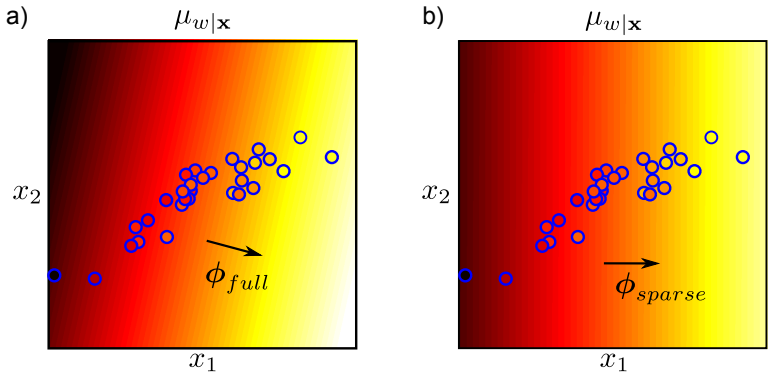


Figure 8.11 Sparse linear regression. a) Bayesian linear regression from two-dimensional data. The background color represents the mean $\mu_{w|\mathbf{x}}$ of the Gaussian prediction $Pr(w|\mathbf{x})$ for w. The variance of $Pr(w|\mathbf{x})$ is not shown. The color of the data points indicates the training value w, so for a perfect regression fit this should match exactly the surrounding color. Here the elements of $\boldsymbol{\phi}$ take arbitrary values and so the gradient of the function points in an arbitrary direction. b) Sparse linear regression. Here, the elements of $\boldsymbol{\phi}$ are encouraged to be zero where they are not necessary to explain the data. The algorithm has found a good fit where the second element of $\boldsymbol{\phi}$ is zero and so there is no dependence on the vertical axis.

In between each of these updates, the posterior mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\Sigma}$ should be recalculated.

In practice, we choose a very small value for the degrees of freedom ($\nu < 10^{-3}$) to encourage sparseness. We may also restrict the maximum possible values of the hidden variables h_i to ensure numerical stability.

At the end of the training, all dimensions of $\boldsymbol{\phi}$ where the hidden variable h_d is large (say > 1000) are discarded. Figure 8.11 shows an example fit to some two-dimensional data. The sparse solution depends only on one of the two possible directions and so is twice as efficient.

In principle, a nonlinear version of this algorithm can be generated by transforming the input data $\mathbf{x}$ to create the vector $\mathbf{z} = \mathbf{f}[\mathbf{x}]$. However, if the transformed data $\mathbf{z}$ is very high-dimensional, we will need correspondingly more hidden variables h_d to cope with these dimensions. Obviously, this idea will not transfer to kernel functions where the dimensionality of the transformed data could be infinite.

To resolve this problem, we will develop the *relevance vector machine*. This model also imposes sparsity, but it does so in a way that makes the final prediction depend only on a sparse subset of the training data, rather than a sparse subset of the observed dimensions. Before we can investigate this model, we must develop a version of linear regression where there is one parameter per data example rather than one per observed dimension. This model is known as *dual linear regression*.

8.7 Dual linear regression

In the standard linear regression model the parameter vector $\boldsymbol{\phi}$ contains D entries corresponding to each of the D dimensions of the (possibly transformed) input data. In the dual formulation, we re-parameterize the model in terms of a vector $\boldsymbol{\psi}$ which has I entries where I is the number of training examples. This is more efficient in situations where we are training a model where the input data are high dimensional, but the number of examples is small ($I < D$), and leads to other interesting models, such as relevance vector regression.

Problem 8.10

8.7.1 Dual model

In the dual model, we retain the original linear dependence of the prediction w on the input data $\mathbf{x}$ so that

$$Pr(w_i|\mathbf{x}_i) = \text{Norm}_{\mathbf{x}_i}[\boldsymbol{\phi}^T\mathbf{x}_i, \sigma^2]. \tag{8.40}$$

However, we now represent the slope parameters $\boldsymbol{\phi}$ as a weighted sum of the observed data points so that

$$\boldsymbol{\phi} = \mathbf{X}\boldsymbol{\psi}, \tag{8.41}$$

where $\boldsymbol{\psi}$ is a $I \times 1$ vector representing the weights (figure 8.12). We term this the *dual parameterization*. Notice that if there are fewer data examples than data dimensions, then there will be fewer unknowns here than in the standard linear regression model and hence learning and inference will be more efficient. Note that the term 'dual' is heavily overloaded in computer science, and the reader should be careful not to confuse this use with its other meanings.

If the data dimensionality D is less than the number of examples I then we can find parameters $\boldsymbol{\psi}$ to represent any gradient vector $\boldsymbol{\phi}$. However, if $D > I$ (often true in vision where measurements can be high dimensional), then the vector $\mathbf{X}\boldsymbol{\psi}$ can only span a subspace of the possible gradient vectors. However, this is not a problem: if there was no variation in the data $\mathbf{X}$ in a given direction in space, then the gradient along that axis should be zero anyway since we have no information about how the world state w varies in this direction.

Making the substitution from equation 8.41, the regression model becomes

$$Pr(w_i|\mathbf{x}_i, \boldsymbol{\theta}) = \text{Norm}_{\mathbf{x}_i}[\boldsymbol{\psi}^T\mathbf{X}^T\mathbf{x}_i, \sigma^2], \tag{8.42}$$

or writing all of the data likelihoods in one term

$$Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\theta}) = \text{Norm}_{\mathbf{w}}\left[\mathbf{X}^T\mathbf{X}\boldsymbol{\psi}, \sigma^2\mathbf{I}\right], \tag{8.43}$$

where the parameters of the model are $\boldsymbol{\theta} = \{\boldsymbol{\psi}, \sigma^2\}$. We now consider how to learn this model using both the maximum likelihood and Bayesian approaches.

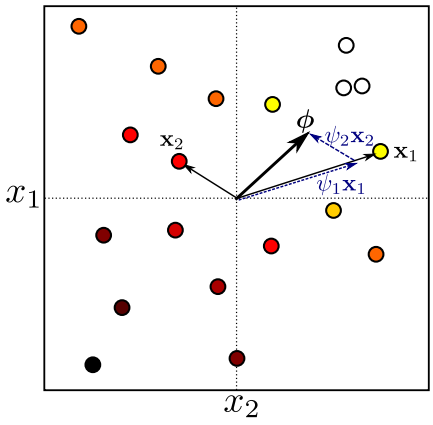


Figure 8.12 Dual variables. Two dimensional training data $\{\mathbf{x}_i\}_{i=1}^I$ and associated world state $\{w_i\}_{i=1}^I$ (indicated by marker color). The linear regression parameter $\boldsymbol{\phi}$ determines the direction in this 2D space in which w changes most quickly. We can alternately represent the gradient direction as a weighted sum of data examples. Here we show the case $\boldsymbol{\phi} = \psi_1\mathbf{x}_1 + \psi_2\mathbf{x}_2$. In practical problems the data dimensionality D is greater than the number of examples I so we take a weighted sum $\boldsymbol{\phi} = \mathbf{X}\boldsymbol{\psi}$ of all of the data points. This is the dual parameterization.

Maximum likelihood solution

Algorithm 8.5

We apply the maximum likelihood method to estimate the parameters $\boldsymbol{\psi}$ in the dual formulation. To this end we maximize the logarithm of the likelihood (equation 8.43) with respect to $\boldsymbol{\psi}$ and σ^2 so that

$$\hat{\boldsymbol{\psi}}, \hat{\sigma}^2 = \underset{\boldsymbol{\psi},\sigma^2}{\operatorname{argmax}} \left[-\frac{I\log[2\pi]}{2} - \frac{I\log[\sigma]}{2} - \frac{(\mathbf{w}-\mathbf{X}^T\mathbf{X}\boldsymbol{\psi})^T(\mathbf{w}-\mathbf{X}^T\mathbf{X}\boldsymbol{\psi})}{2\sigma^2} \right]. \quad (8.44)$$

Problem 8.11

To maximize this expression, we take derivatives with respect to $\boldsymbol{\psi}$ and σ^2, equate the resulting expressions to zero, and solve to find

$$\begin{aligned} \hat{\boldsymbol{\psi}} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{w} \\ \hat{\sigma}^2 &= \frac{(\mathbf{w}-\mathbf{X}^T\mathbf{X}\boldsymbol{\psi})^T(\mathbf{w}-\mathbf{X}^T\mathbf{X}\boldsymbol{\psi})}{I}. \end{aligned} \quad (8.45)$$

This solution is actually the same as for the original linear regression model (equations 8.6). For example, if we substitute in the definition $\boldsymbol{\phi} = \mathbf{X}\boldsymbol{\psi}$,

$$\begin{aligned} \hat{\boldsymbol{\phi}} = \mathbf{X}\hat{\boldsymbol{\psi}} &= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{w} \\ &= (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{X}^T\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{w} \\ &= (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{w}, \end{aligned} \quad (8.46)$$

which was the original maximum likelihood solution for $\boldsymbol{\phi}$.

Bayesian solution

We now explore the Bayesian approach to the dual regression model. As before, we treat the dual parameters $\boldsymbol{\psi}$ as uncertain, assuming that the noise σ^2 is known. Once again, we will estimate this separately using maximum likelihood.

The goal of the Bayesian approach is to compute the posterior distribution $Pr(\boldsymbol{\psi}|\mathbf{X}, \mathbf{w})$ over possible values of the parameters $\boldsymbol{\psi}$ given the training data pairs $\{\mathbf{x}_i, w_i\}_{i=1}^I$. We start by defining a prior $Pr(\boldsymbol{\psi})$ over the parameters. Since we have no particular prior knowledge, we choose a normal distribution with a large spherical covariance,

$$Pr(\boldsymbol{\psi}) = \text{Norm}_{\boldsymbol{\psi}}[\mathbf{0}, \sigma_p^2\mathbf{I}]. \tag{8.47}$$

We use Bayes' rule to compute the posterior distribution over the parameters

$$Pr(\boldsymbol{\psi}|\mathbf{X}, \mathbf{w}, \sigma^2) = \frac{Pr(\mathbf{X}|\mathbf{w}, \boldsymbol{\psi}, \sigma^2)Pr(\boldsymbol{\psi})}{Pr(\mathbf{X}|\mathbf{w}, \sigma^2)}, \tag{8.48}$$

which can be shown to yield the closed form expression

$$Pr(\boldsymbol{\psi}|\mathbf{X}, \mathbf{w}, \sigma^2) = \text{Norm}_{\boldsymbol{\psi}}\left[\frac{1}{\sigma^2}\mathbf{A}^{-1}\mathbf{X}^T\mathbf{X}\mathbf{w}, \mathbf{A}^{-1}\right], \tag{8.49}$$

where

$$\mathbf{A} = \frac{1}{\sigma^2}\mathbf{X}^T\mathbf{X}\mathbf{X}^T\mathbf{X} + \frac{1}{\sigma_p^2}\mathbf{I}. \tag{8.50}$$

To compute the predictive distribution $Pr(\mathbf{w}^*|\mathbf{x}^*)$, we take an infinite weighted sum over the predictions of the model associated with each possible value of the parameters $\boldsymbol{\psi}$,

$$\begin{aligned} Pr(w^*|\mathbf{x}^*, \mathbf{X}, \mathbf{w}) &= \int Pr(w^*|\mathbf{x}^*, \boldsymbol{\psi})Pr(\boldsymbol{\psi}|\mathbf{X}, \mathbf{w})\, d\boldsymbol{\psi} \qquad (8.51)\\ &= \text{Norm}_{w^*}\left[\frac{1}{\sigma^2}\mathbf{x}^{*T}\mathbf{X}\mathbf{A}^{-1}\mathbf{X}^T\mathbf{X}\mathbf{w}^*, \mathbf{x}^{*T}\mathbf{X}\mathbf{A}^{-1}\mathbf{X}^T\mathbf{x}^* + \sigma^2\right]. \end{aligned}$$

To generalize the model to the nonlinear case, we replace the training data $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_I]$ with the transformed data $\mathbf{Z} = [\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_I]$ and the test data $\mathbf{x}^*$ with the transformed test data $\mathbf{z}^*$. Since the resulting expression depends only on inner products of the form $\mathbf{Z}^T\mathbf{Z}$ and $\mathbf{Z}^T\mathbf{z}^*$, it is directly amenable to kernelization. Algorithm 8.6

As for the original regression model, the variance parameter σ^2 can be estimated by maximizing the log of the marginal likelihood which is given by

$$Pr(\mathbf{w}|\mathbf{X}, \sigma^2) = \text{Norm}_{\mathbf{w}}[\mathbf{0}, \sigma_p^2\mathbf{X}^T\mathbf{X}\mathbf{X}^T\mathbf{X} + \sigma^2\mathbf{I}]. \tag{8.52}$$

8.8 Relevance vector regression

Having developed the dual approach to linear regression, we are now in a position to develop a model that depends only sparsely on the training data. To this end, Algorithm 8.7

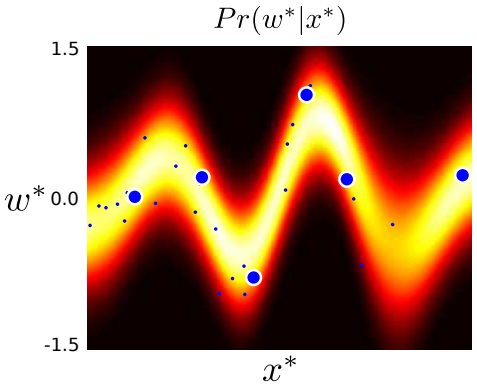


Figure 8.13 Relevance vector regression. A prior applying sparseness is applied to the dual parameters. This means that the final classifier only depends on a subset of the data points (indicated by the six larger points). The resulting regression function is considerably faster to evaluate and tends to be simpler: this means it is less likely to overfit to random statistical fluctuations in the training data and generalizes better to new data.

we impose a penalty for every non-zero weighted training example. We achieve this by replacing the normal prior over the dual parameters $\boldsymbol{\psi}$ with a product of one dimensional t-distributions so that

$$Pr(\boldsymbol{\psi}) = \prod_{i=1}^{I} \text{Stud}_{\boldsymbol{\psi}_i}[0, 1, \nu]. \tag{8.53}$$

This model is known as *relevance vector regression*.

This situation is exactly analogous to the sparse linear regression model (section 8.6) except that now we are working with dual variables. As for the sparse model it is not possible to marginalize over the variables $\boldsymbol{\psi}$ with the t-distributed prior. Our approach will again be to approximate the t-distributions by maximizing with respect to their hidden variables rather than marginalizing over them (equation 8.35). By analogy with section 8.6, the marginal likelihood becomes:

$$Pr(\mathbf{w}|\mathbf{X}, \sigma^2) \approx \max_{\mathbf{H}} \left[\text{Norm}_{\mathbf{w}}[\mathbf{0}, \mathbf{X}^T\mathbf{X}\mathbf{H}^{-1}\mathbf{X}^T\mathbf{X} + \sigma^2\mathbf{I}] \prod_{d=1}^{D} \text{Gam}_{h_d}[\nu/2, \nu/2] \right]. \tag{8.54}$$

where the matrix $\mathbf{H}$ contains the hidden variables $\{h_i\}_{i=1}^{I}$ associated with the t-distribution on its diagonal and zeros elsewhere. Notice that this expression is similar to equation 8.52 except that instead of every data point having the same prior variance σ_p^2, they now have individual variances that are determined by the hidden variables that form the elements of the diagonal matrix $\mathbf{H}$.

In relevance vector regression, we alternately (i) optimize the marginal likelihood with respect to the hidden variables and (ii) optimize the marginal likelihood with respect to the variance parameter σ^2 using

$$h_i^{new} = \frac{1 - h_i\Sigma_{ii} + \nu}{\mu_i^2 + \nu}, \tag{8.55}$$

and

$$(\sigma^2)^{new} = \frac{1}{I - \sum_i (1 - h_i \Sigma_{ii})} \left(\mathbf{w} - \mathbf{X}^T\mathbf{X}\boldsymbol{\mu}\right)^T \left(\mathbf{w} - \mathbf{X}^T\mathbf{X}\boldsymbol{\mu}\right), \tag{8.56}$$

In between each step we update the mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\Sigma}$ of the posterior distribution

$$\begin{aligned} \boldsymbol{\mu} &= \frac{1}{\sigma^2}\mathbf{A}^{-1}\mathbf{X}^T\mathbf{X}\mathbf{w} \\ \boldsymbol{\Sigma} &= \mathbf{A}^{-1}, \end{aligned} \tag{8.57}$$

where $\mathbf{A}$ is defined as

$$\mathbf{A} = \frac{1}{\sigma^2}\mathbf{X}^T\mathbf{X}\mathbf{X}^T\mathbf{X} + \mathbf{H}. \tag{8.58}$$

At the end of the training all data examples where the hidden variable h_i is large (say > 1000) are discarded as here the coefficients $\boldsymbol{\psi}_i$ will be very small and contribute almost nothing to the solution.

Since this algorithm depends only on inner products, a nonlinear version of this algorithm can be generated by replacing the inner products with a kernel function $\mathrm{k}[\mathbf{x}_i, \mathbf{x}_j]$. If the kernel itself contains parameters, these may be also be manipulated to improve the log marginal variance during the fitting procedure. Figure 8.13 shows an example fit using the RBF kernel. The final solution now only depends on six data points but nonetheless still captures the important aspects of the data.

8.9 Regression to multivariate data

Throughout this chapter we have discussed predicting a scalar value w_i from multivariate data $\mathbf{x}_i$. In real-world situations such as the pose regression problem, the world states $\mathbf{w}_i$ are multivariate. It is trivial to extend the models in this chapter: we simply construct a separate regressor for each dimension. The exception to this rule is the relevance vector machine: here we might want to ensure that the sparse structure is common for each of these models, so the efficiency gains are retained. To this end, we modify the model so that a single set of hidden variables is shared across the model for each world state dimension.

8.10 Applications

Regression methods are used less frequently in vision than classification, but nonetheless there are many useful applications. The majority of these involve estimating the position or pose of objects, since the unknowns in such problems are naturally treated as continuous.

Figure 8.14 Body pose estimation results. a) Silhouettes of walking avatar. b) Estimated body pose based on silhouette using a relevance vector machine. The RVM used radial basis functions and constructed its final solution from just 156 of 2636 (6%) of the training examples. It produced a mean test error of 6.0^o averaged over the three joint angles for the 18 main body parts and the overall compass direction of the model. Adapted from Agarwal & Triggs (2006).

8.10.1 Human body pose estimation

Agarwal & Triggs (2006) developed a system based on the relevance vector machine to predict body pose $\mathbf{w}$ from silhouette data $\mathbf{x}$. To encode the silhouette, they computed a 60-dimensional shape context feature (section 13.3.5) at each of 400-500 points on the boundary of the object. To reduce the data dimensionality, they computed the similarity of each shape context feature to each of 100 different prototypes. Finally, they formed a 100-dimensional histogram containing the aggregated 100-dimensional similarities for all of the boundary points. This histogram was used as the data vector $\mathbf{x}$. The body pose was encoded by the 3 joint angles of each of the 18 major body joints and the overall azimuth (compass heading) of the body. The resulting 55-dimensional vector was used as the world state $\mathbf{w}$.

A relevance vector machine was trained using 2636 data vectors $\mathbf{x}_i$ extracted from silhouettes that were rendered using the commercial program POSER from known motion capture data $\mathbf{w}_i$. Using a radial basis function kernel, the relevance vector machine based its solution on just 6% of these training examples. The body pose angles of test data could be predicted to within an average of 6^o error (figure 8.14). They also demonstrated that the system worked reasonably well on silhouettes from real images (figure 8.1).

Silhouette information is by its nature ambiguous: it is very hard to tell which leg is in front of the other based on a single silhouette. Agarwal & Triggs (2006) partially circumvented this system by tracking the body pose $\mathbf{w}_i$ through a video sequence. Essentially, the ambiguity at a given frame is resolved by encouraging the estimated pose in adjacent frames in the sequence to be similar: information from frames where the pose vector is well defined is propagated through the sequence to resolve ambiguities in other parts (see chapter 19).

However, the ambiguity of silhouette data is an argument for *not* using this type of classifier: the regression models in this chapter are designed to give a unimodal normal output. To effectively classify single frames of data, we should use a regression method that produces a multi-modal prediction that can effectively describe the ambiguity.

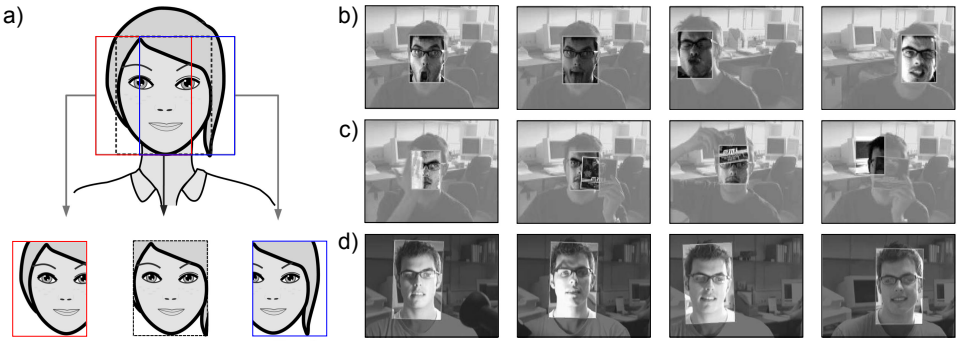


Figure 8.15 Tracking using displacement experts. The goal of the system is to predict a displacement vector indicating the motion of the object based on the pixel data at its last known position. a) The system is trained by perturbing the bounding box around the object to simulate the motion of the object. b) The system successfully tracks a face, even in the presence c) of partial occlusions. d) If the system is trained using gradient vectors rather than raw pixel values, it is also quite robust to changes in illumination. Adapted from Williams *et al.* (2005) ©2005 IEEE.

8.10.2 Displacement experts

Regression models are also used to form *displacement experts* in tracking applications. The goal is to take a region of the image $\mathbf{x}$ and return a set of numbers $\mathbf{w}$ that indicate the change in position of an object relative to the window. The world state $\mathbf{w}$ might simply contain the horizontal and vertical translation vectors, or might contain parameters of a more complex 2D transformation (chapter 15). For simplicity, we will describe the former situation.

Training data are extracted as follows. A bounding box around of the object of interest (car, face, etc.) is identified in a number of frames. For each of these frames, the bounding box is perturbed by a pre-determined set of translation vectors, to simulate the object moving in the opposite direction (figure 8.15a). In this way, we associate a translation vector $\mathbf{w}_i$ with each perturbation. The data from the perturbed bounding box are extracted, resized to a standard shape, and histogram equalized (section 13.1.2) to induce a degree of invariance to illumination changes. The resulting values are then concatenated to form the data vector $\mathbf{x}_i$.

Williams *et al.* (2005) describe a system of this kind in which the elements of $\mathbf{w}$ were learned by a set of independent relevance vector machines. They initialize the position of the object using a standard object detector (see chapter 9). In the subsequent frame, they compute a prediction for the displacement vector $\mathbf{w}$ using the relevance vector machines on the data $\mathbf{x}$ from the original position. This prediction is combined in a Kalman filter-like system (chapter 19) that imposes prior knowledge about the continuity of the motion to create a robust method for tracking known objects in scenes. Figure 8.15b-d show a series of tracking results from this system.

Discussion

The goal of this chapter was to introduce discriminative approaches to regression. These have niche applications in vision related to predicting the pose and position of objects. However, the main reason for studying these models is that the concepts involved (sparsity, dual variables, kernelization) are all important for discriminative classification methods. These are very widely used but are rather more complex and are discussed in the following chapter.

Notes

Regression methods: Rasmussen & Williams (2006) is a comprehensive resource on Gaussian processes. The relevance vector machine was first introduced by Tipping (2001). Several innovations within the vision community have extended these models. Williams *et al.* (2006) presented a semi-supervised method for Gaussian process regression in which the world state $\mathbf{w}$ is only known for a subset of examples. Ranganathan & Yang (2008) presented an efficient algorithm for online learning of Gaussian processes when the kernel matrix is sparse. Thayananthan *et al.* (2006) developed a multivariate version of the relevance vector machine.

Applications: Applications of regression in vision include head pose estimation (Williams *et al.* 2006; Ranganathan & Yang 2008; Rae & Ritter 1998), body tracking (Williams *et al.* 2006; Agarwal & Triggs 2006; Thayananthan *et al.* 2006), eye tracking (Williams *et al.* 2006), and tracking of other objects (Williams *et al.* 2005; Ranganathan & Yang 2008).

Multimodal posterior: One of the drawbacks of using the methods in this chapter is that they always produce a unimodal normally distributed posterior. For some problems (e.g., body pose estimation), the posterior probability over the world state may be genuinely multimodal – there is more than one interpretation of the data. One approach to this is to build many regressors that relate small parts of the world state to the data (Thayananthan *et al.* 2006). Alternatively, it is possible to use generative regression methods in which either the joint density is modeled directly (Navaratnam *et al.* 2007) or the likelihood and prior are modeled separately (Urtasun *et al.* 2006). In these methods, the posterior distribution over the world is multi-modal. However, the cost of this is that it is intractable to compute exactly, and so we must rely on optimization techniques to find its modes.

Problems

Problem 8.1 Consider a regression problem where the world state w is known to be positive. To cope with this we could construct a regression model in which the world state is modeled as a gamma distribution. We could constrain both parameters α, β of the gamma distribution to be the same so that $\alpha = \beta$ and make them a function of the data $\mathbf{x}$. Describe a maximum likelihood approach to fitting this model.

Problem 8.2 Consider a robust regression model based on the t-distribution rather than the normal distribution. Define this model precisely in mathematical terms and sketch out a maximum likelihood approach to fitting the parameters.

Problem 8.3 Prove that the maximum likelihood solution for the gradient in the linear regression model is

$$\hat{\boldsymbol{\phi}} = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{w}.$$

Problem 8.4 For the Bayesian linear regression model (section 8.2), show that the posterior distribution over the parameters $\boldsymbol{\phi}$ is given by

$$Pr(\boldsymbol{\phi}|\mathbf{X}, \mathbf{w}) = \text{Norm}_{\boldsymbol{\phi}}\left[\frac{1}{\sigma^2}\mathbf{A}^{-1}\mathbf{X}\mathbf{w}, \mathbf{A}^{-1}\right],$$

where

$$\mathbf{A} = \frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T + \frac{1}{\sigma_p^2}\mathbf{I}.$$

Problem 8.5 For the Bayesian linear regression model (section 8.2) show that the predictive distribution for a new data example $\mathbf{x}^*$ is given by

$$Pr(w^*|\mathbf{x}^*, \mathbf{X}, \mathbf{w}) = \text{Norm}_{w^*}\left[\frac{1}{\sigma^2}\mathbf{x}^{*T}\mathbf{A}^{-1}\mathbf{X}\mathbf{w}, \mathbf{x}^{*T}\mathbf{A}^{-1}\mathbf{x}^* + \sigma^2\right].$$

Problem 8.6 Use the matrix inversion lemma (appendix C.8.4) to show that

$$\mathbf{A}^{-1} = \left(\frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T + \frac{1}{\sigma_p^2}\mathbf{I}_D\right)^{-1} = \sigma_p^2\mathbf{I}_D - \sigma_p^2\mathbf{X}\left(\mathbf{X}^T\mathbf{X} + \frac{\sigma^2}{\sigma_p^2}\mathbf{I}_I\right)^{-1}\mathbf{X}^T.$$

Problem 8.7 Compute the derivative of the marginal likelihood

$$Pr(\mathbf{w}|\mathbf{X}, \sigma^2) \quad = \quad \text{Norm}_{\mathbf{w}}[\mathbf{0}, \sigma_p^2\mathbf{X}^T\mathbf{X} + \sigma^2\mathbf{I}],$$

with respect to the variance parameter σ^2.

Problem 8.8
Compute a closed form expression for the approximated t-distribution used to impose sparseness.

$$q(h) = \max_h\left[\text{Norm}_{\boldsymbol{\phi}}[0, h^{-1}]\text{Gam}_h[\nu/2, \nu/2]\right].$$

Plot this function for $\nu = 2$. Plot the 2D function $[h_1, h_2] = q(h_1)q(h_2)$ for $\nu = 2$.

Problem 8.9 Describe maximum likelihood learning and inference algorithms for a non-linear regression model based on polynomials where

$$Pr(w|x) = \text{Norm}_w[\phi_0 + \phi_1 x + \phi_2 x^2 + \phi_3 x^3, \sigma^2].$$

Problem 8.10 I wish to learn a linear regression model in which I predict the world w from I examples of $D \times 1$ data $\mathbf{x}$ using the maximum likelihood method. If $I > D$ is it more efficient to use the dual parameterization or the original linear regression model?

Problem 8.11 Show that the maximum likelihood estimate for the parameters $\boldsymbol{\psi}$ in the dual linear regression model (section 8.7) is given by

$$\hat{\boldsymbol{\psi}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{w}.$$

Chapter 9

Classification models

This chapter concerns discriminative models for classification. The goal is to directly model the posterior probability distribution $Pr(w|\mathbf{x})$ over a discrete world state $w \in \{1, \ldots K\}$ given the continuous observed data vector $\mathbf{x}$. Models for classification are very closely related to those for regression and the reader should be familiar with the contents of chapter 8 before proceeding.

To motivate the models in this chapter, we will consider *gender classification*: here we observe a 60×60 RGB image containing a face (figure 9.1) and concatenate the RGB values to form the 10800×1 vector $\mathbf{x}$. Our goal is to take the vector $\mathbf{x}$ and return the probability distribution $Pr(w|\mathbf{x})$ over a label $w \in \{0, 1\}$ indicating whether the face is male ($w = 0$) or female ($w = 1$).

Gender classification is a *binary classification* task as there are only two possible values of the world state. Throughout most of this chapter, we will restrict our discussion to binary classification. We discuss how to extend these models to cope with an arbitrary number of classes in section 9.9.

9.1 Logistic regression

We will start by considering *logistic regression*, which despite its name is a model that can be applied to classification. Logistic regression (figure 9.2) is a discriminative model; we select a probability distribution over the world state $w \in \{0, 1\}$ and make its parameters contingent on the observed data $\mathbf{x}$. Since the world state is binary, we will describe it with a Bernoulli distribution and we will make the single Bernoulli parameter λ (indicating the probability that the world state takes the value $w = 1$) a function of the measurements $\mathbf{x}$.

In contrast to the regression model, we cannot simply make the parameter λ a linear function $\phi_0 + \boldsymbol{\phi}^T\mathbf{x}$ of the measurements; a linear function can return any value, but the parameter λ must lie between 0 and 1. Consequently, we first compute the linear function and then pass this through the *logistic sigmoid function* $\text{sig}[\bullet]$ that maps the range $[-\infty, \infty]$ to $[0, 1]$. The final model is hence

Figure 9.1 Gender classification. Consider a 60×60 pixel image of a face. We concatenate the RGB values to make a 10800×1 data vector $\mathbf{x}$. The goal of gender classification is to use the data $\mathbf{x}$ to infer a label $w \in \{0, 1\}$ indicating whether the window contains a) a male or b) a female face. This is challenging because the differences are subtle and there is image variation due to changes in pose, lighting, and expression. Note that real systems would preprocess the image before classification by registering the faces more closely and compensating in some way for lighting variation (see chapter 13).

$$Pr(w|\phi_0, \boldsymbol{\phi}, \mathbf{x}) = \text{Bern}_w\left[\text{sig}[a]\right], \tag{9.1}$$

where a is termed the *activation* and is given by the linear function

$$a = \phi_0 + \boldsymbol{\phi}^T\mathbf{x}. \tag{9.2}$$

The logistic sigmoid function sig[•] is given by

$$\text{sig}[a] = \frac{1}{1+\exp[-a]}. \tag{9.3}$$

Problem 9.1

As the activation a tends to ∞ this function tends to one. As a tends to $-\infty$ it tends to zero. When a is zero, the logistic sigmoid function returns a value of one half. For 1D data x, the overall effect of this transformation is to describe a sigmoid curve relating x to λ (figures 9.2c and 9.3a). The horizontal position of the sigmoid is determined by the place that the linear function a crosses zero (i.e., the x-intercept) and the steepness of the sigmoid depends on the gradient ϕ_1.

In more than one dimension, the relationship between $\mathbf{x}$ and λ is more complex (figure 9.3b). The predicted parameter λ has a sigmoid profile in the direction of the gradient vector $\boldsymbol{\phi}$ but is constant in all perpendicular directions. This induces a linear *decision boundary*. This is the set of positions in data space $\{\mathbf{x} : Pr(w = 1|\mathbf{x}) = 0.5\}$ where the posterior probability is 0.5; the decision boundary separates the region where the world state w is more likely to be 0 from the region where it is more likely to be 1. For logistic regression, the decision boundary takes the form of a hyperplane with the normal vector in the direction of $\boldsymbol{\phi}$.

As for regression, we can simplify the notation by attaching the y-intercept ϕ_0 to the start of the parameter vector $\boldsymbol{\phi}$ so that $\boldsymbol{\phi} \leftarrow [\phi_0 \quad \boldsymbol{\phi}^T]^T$ and attaching 1 to the start of the data vector $\mathbf{x}$ so that $\mathbf{x} \leftarrow [1 \quad \mathbf{x}^T]^T$. After these changes, the activation is now $a = \boldsymbol{\phi}^T\mathbf{x}$, and the final model becomes

$$Pr(w|\boldsymbol{\phi}, \mathbf{x}) = \text{Bern}_w\left[\frac{1}{1+\exp[-\boldsymbol{\phi}^T\mathbf{x}]}\right]. \tag{9.4}$$

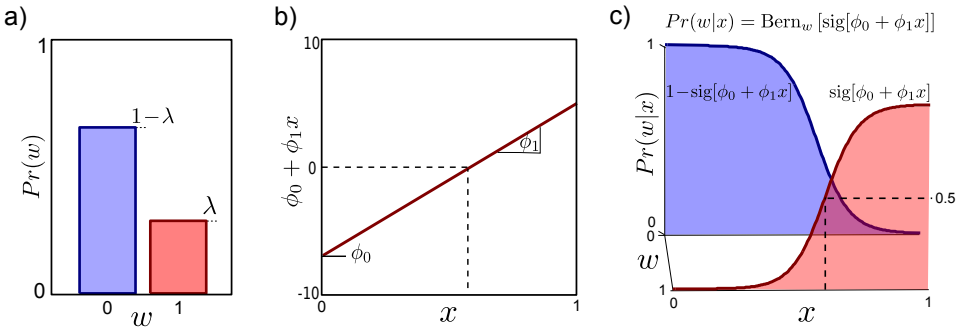


Figure 9.2 Logistic regression. a) We represent the world state w as a Bernoulli distribution and make the Bernoulli parameter λ a function of the observations x. b) We compute the activation a as a linear sum $a = \phi_0 + \phi_1 x$ of the observations. c) The Bernoulli parameter λ is formed by passing the activation through a logistic sigmoid function $\text{sig}[\bullet]$ to constrain the value to lie between 0 and 1, giving the characteristic sigmoid shape. In learning, we fit parameters $\boldsymbol{\theta} = \{\phi_0, \phi_1\}$ using training pairs $\{x_i, w_i\}$. In inference, we take a new datum x^* and evaluate the posterior $Pr(w^*|x^*)$ over the state.

Notice that this is very similar to the linear regression model (section 8.1) except for the introduction of the nonlinear logistic sigmoid function $\text{sig}[\bullet]$ (explaining the unfortunate name 'logistic regression'). However, this small change has serious implications: maximum likelihood learning of the parameters $\boldsymbol{\phi}$ is considerably harder than for linear regression, and to adopt the Bayesian approach we will be forced to make approximations.

9.1.1 Learning: maximum likelihood

In maximum likelihood learning, we consider fitting the parameters $\boldsymbol{\phi}$ using I paired examples of training data $\{\mathbf{x}_i, w_i\}_{i=1}^I$ (figure 9.3). Assuming independence of the training pairs we have

Algorithm 9.1

$$\begin{aligned} Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\phi}) &= \prod_{i=1}^{I} \lambda^{w_i}(1-\lambda)^{1-w_i} \\ &= \prod_{i=1}^{I} \left(\frac{1}{1+\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]}\right)^{w_i} \left(\frac{\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]}{1+\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]}\right)^{1-w_i}, \end{aligned} \tag{9.5}$$

where $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_I]$ is a matrix containing the measurements and $\mathbf{w} = [w_1, w_2, \ldots, w_I]^T$ is a vector containing all of the binary world states. The maximum likelihood method finds parameters $\boldsymbol{\phi}$, which maximize this expression.

As usual, however, it is simpler to maximize the logarithm L of this expression. Since the logarithm is a monotonic transformation, it does not change the position

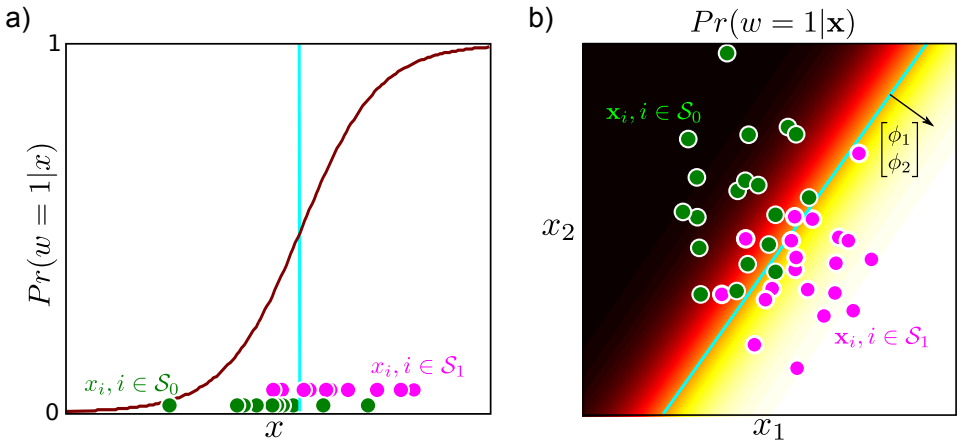


Figure 9.3 Logistic regression model fitted to two different datasets. a) One dimensional data. Green points denote set of examples $\mathcal{S}_0$ where $w = 0$. Pink points denote $\mathcal{S}_1$ where $w = 1$. Note that in this (and all future figures in this chapter) we have only plotted the probability $Pr(w = 1|x)$ (compare to figure 9.2c). The probability $Pr(w = 0|x)$ can be computed as $1 - Pr(w = 1|\mathbf{x})$. b) Two-dimensional data. Here, the model has a sigmoid profile in the direction of the gradient $\boldsymbol{\phi}$ and $Pr(w = 1|\mathbf{x})$ is constant in the orthogonal directions. The decision boundary (cyan line) is linear.

of the maximum with respect to $\boldsymbol{\phi}$. Applying the logarithm replaces the product with a sum so that

$$L = \sum_{i=1}^{I} w_i \log\left[\frac{1}{1+\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]}\right] + \sum_{i=1}^{I}(1-w_i)\log\left[\frac{\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]}{1+\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]}\right]. \quad (9.6)$$

Problem 9.2

The derivative of the log likelihood L with respect to the parameters $\boldsymbol{\phi}$ is

$$\frac{\partial L}{\partial \boldsymbol{\phi}} = -\sum_{i=1}^{I}\left(\frac{1}{1+\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]} - w_i\right)\mathbf{x}_i = -\sum_{i=1}^{I}(\text{sig}[a_i] - w_i)\,\mathbf{x}_i. \quad (9.7)$$

Unfortunately, when we equate this expression to zero, there is no way to rearrange to get a closed form solution for the parameters $\boldsymbol{\phi}$. Instead, we must rely on a *nonlinear optimization* technique to find the maximum of this objective function. Optimization techniques are discussed in detail in appendix B. In brief, we start with an initial estimate of the solution $\boldsymbol{\phi}$ and iteratively improve it until no more progress can be made.

Here, we will apply the *Newton method*, in which we base the update of the parameters on the first and second derivatives of the function at the current position so that

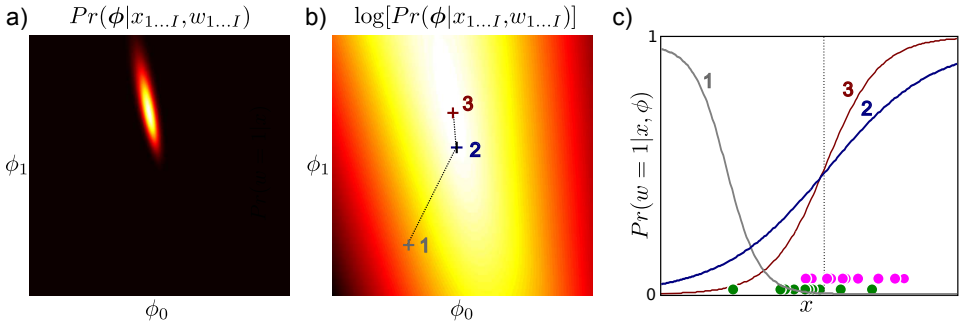


Figure 9.4 Parameter estimation for logistic regression with 1D data. a) In maximum likelihood learning, we seek the maximum of $Pr(\mathbf{w}|\mathbf{X},\phi)$ with respect to ϕ. b) In practice, we instead maximize log likelihood: notice that the peak is in the same place. Crosses show results of two iterations of optimization using Newton's method. c) The logistic sigmoid functions associated with the parameters at each optimization step. As the log likelihood increases, the model fits the data more closely: the green points represent data where $w=0$ and the purple points represent data where $w=1$, so we expect the best-fitting model to increase from left to right just like curve 3.

$$\boldsymbol{\phi}^{[t]} = \boldsymbol{\phi}^{[t-1]} + \alpha\left(\frac{\partial^2 L}{\partial \boldsymbol{\phi}^2}\right)^{-1}\frac{\partial L}{\partial \boldsymbol{\phi}}, \tag{9.8}$$

where $\boldsymbol{\phi}^{[t]}$ denotes the estimate of the parameters $\boldsymbol{\phi}$ at iteration t and α determines how much we change this estimate and is usually chosen by an explicit search at each iteration.

For the logistic regression model, the $D\times 1$ vector of first derivatives and the $D\times D$ matrix of second derivatives are given by

Problem 9.3

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{\phi}} &= -\sum_{i=1}^{I}(\text{sig}[a_i]-w_i)\mathbf{x}_i \\ \frac{\partial^2 L}{\partial \boldsymbol{\phi}^2} &= -\sum_{i=1}^{I}\text{sig}[a_i](1-\text{sig}[a_i])\mathbf{x}_i\mathbf{x}_i^T.\end{aligned} \tag{9.9}$$

These are known as the *gradient vector* and the *Hessian matrix*, respectively[1].

The expression for the gradient vector has an intuitive explanation. The contribution of each data point depends on the difference between the actual class w_i and the predicted probability $\lambda=\text{sig}[a_i]$ of being in class 1; points that are classified incorrectly contribute more to this expression and hence have more influence

[1]Note that these are the gradient and Hessian of the log likelihood that we aim to *maximize*. If this is implemented using a nonlinear minimization algorithm, we should multiply the objective function, gradient and Hessian by -1

on the parameter values. Figure 9.4 shows maximum likelihood learning of the parameters $\boldsymbol{\phi}$ for 1D data using a series of Newton steps.

For general functions, the Newton method only finds *local maxima*. At the end of the procedure we cannot be certain that there is not a taller peak in the likelihood function elsewhere. However, the log likelihood for logistic regression has a special property. It is a *concave* function of the parameters $\boldsymbol{\phi}$. For concave functions there are never multiple maxima and gradient-based approaches are guaranteed to find the global maximum. It is possible to establish whether a function is concave or not by examining the Hessian matrix. If this is negative definite for all $\boldsymbol{\phi}$, then the function is concave. This is the case for logistic regression as the Hessian (equation 9.9) consists of a negative weighted sum of outer products[2].

*

9.1.2 Problems with the logistic regression model

Problem 9.4

The logistic regression model works well for simple data sets, but for more complex visual data it will not generally suffice. It is limited in the following ways.

1. It is overconfident as it was learned using maximum likelihood.
2. It can only describe linear decision boundaries.
3. It is inefficient and prone to over-fitting in high dimensions.

In the remaining part of this chapter, we will extend this model to cope with these problems (figure 9.5).

9.2 Bayesian logistic regression

In the Bayesian approach, we learn a distribution $Pr(\boldsymbol{\phi}|\mathbf{X}, \mathbf{w})$ over the possible parameter values $\boldsymbol{\phi}$ that are compatible with the training data. In inference, we observe a new data example $\mathbf{x}^*$ and use this distribution to weight the predictions for the world state w^* given by each possible estimate of $\boldsymbol{\phi}$. In linear regression (section 8.2), there were closed form expressions for both of these steps. However, the nonlinear function sig[•] in logistic regression means that this is no longer the case. To get around this, we will approximate both steps so that we retain neat closed form expressions and the algorithm is tractable.

9.2.1 Learning

Algorithm 9.2

We start by defining a prior over the parameters $\boldsymbol{\phi}$. Unfortunately, there is no

[2]When we are concerned with minimizing functions we equivalently consider whether the function is *convex* and so has only a single minimum. If the Hessian matrix is positive definite everywhere then the function is convex

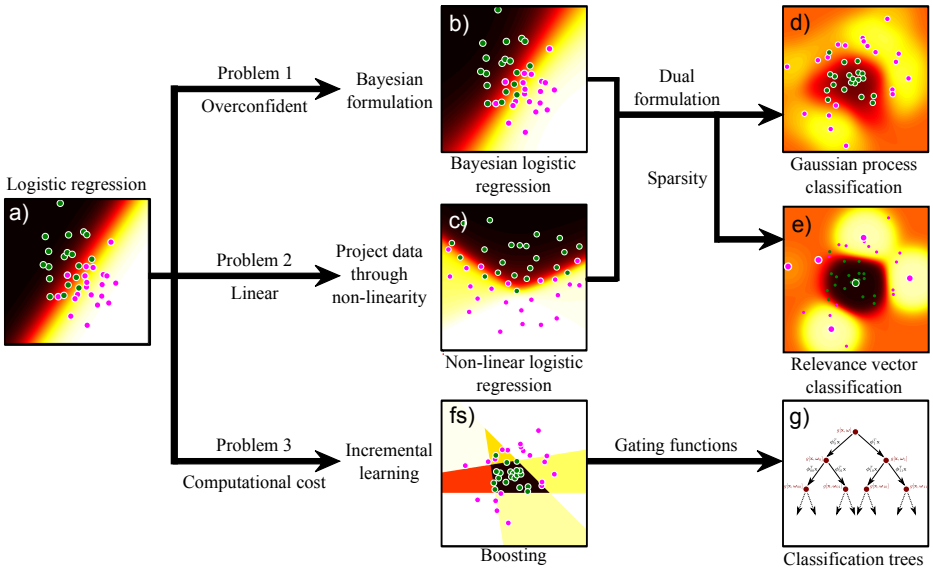


Figure 9.5 Family of classification models. a) In the remaining part of the chapter, we will address several of the limitations of logistic regression for binary classification. b) The logistic regression model with maximum likelihood learning is overconfident, and hence we develop a Bayesian version (section 9.2). c) It is unrealistic to always assume a linear relationship between the data and the world and to this end we introduce a nonlinear version (section 9.3). d) Combining the Bayesian and nonlinear versions of regression leads to Gaussian process classification. e) The logistic regression model also has many parameters and may require considerable resources to learn when the data dimension is high and so we develop relevance vector classification which encourages sparsity. f) We can also build a sparse model by incrementally adding parameters in a boosting model. g) Finally, we consider a very fast classification model based on a tree structure.

conjugate prior for the likelihood in the logistic regression model (equation 9.1); this is why there won't be closed form expressions for the likelihood and predictive distribution. With nothing else to guide us, a reasonable choice for the prior over the continuous parameters $\boldsymbol{\phi}$ is a multivariate normal distribution with zero mean and a large spherical covariance so that

$$Pr(\boldsymbol{\phi}) = \text{Norm}_{\boldsymbol{\phi}}[\mathbf{0}, \sigma_p^2\mathbf{I}]. \tag{9.10}$$

To compute the posterior probability distribution $Pr(\boldsymbol{\phi}|\mathbf{X}, \mathbf{w})$ over the parameters $\boldsymbol{\phi}$ given the training data pairs $\{\mathbf{x}_i, w_i\}$ we apply Bayes' rule,

$$Pr(\boldsymbol{\phi}|\mathbf{X}, \mathbf{w}) = \frac{Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\phi})Pr(\boldsymbol{\phi})}{Pr(\mathbf{w}|\mathbf{X})}, \tag{9.11}$$

where the likelihood and prior are given by equations 9.5 and 9.10, respectively.

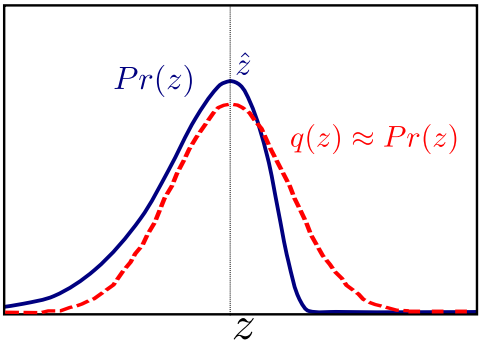


Figure 9.6 Laplace approximation. A probability density (blue curve) is approximated by a normal distribution (red curve). The mean of the normal (and hence the peak) is chosen to coincide with the peak of the original pdf. The variance of the normal is chosen so that its second derivatives at the mean match the second derivatives of the original pdf at the peak.

Since we are not using a conjugate prior, there is no simple closed form expression for this posterior and so we are forced to make an approximation of some kind.

Problem 9.5
Problem 9.6

One possibility is to use the *Laplace approximation* (figure 9.6) which is a general method for approximating complex probability distributions. The goal is to approximate the posterior distribution by a multivariate normal. We select the parameters of this normal so that (i) the mean is at the peak of the posterior distribution (i.e., at the MAP estimate) and (ii) the covariance is such that the second derivatives at the peak match the second derivatives of the true posterior distribution at its peak.

Hence, to make the Laplace approximation, we first find the MAP estimate of the parameters $\hat{\boldsymbol{\phi}}$, and to this end we use a nonlinear optimization technique such as Newton's method to maximize the criterion

$$L = \sum_{i=1}^{I} \log[Pr(w_i|\mathbf{x}_i, \boldsymbol{\phi})] + \log[Pr(\boldsymbol{\phi})]. \tag{9.12}$$

Newton's method needs the derivatives of the log posterior which are

$$\begin{aligned} \frac{\partial L}{\partial \boldsymbol{\phi}} &= -\sum_{i=1}^{I} (\text{sig}[a_i] - w_i)\mathbf{x}_i - \frac{\boldsymbol{\phi}}{\sigma_p^2} \\ \frac{\partial^2 L}{\partial \boldsymbol{\phi}^2} &= -\sum_{i=1}^{I} \text{sig}[a_i](1 - \text{sig}[a_i])\mathbf{x}_i\mathbf{x}_i^T - \frac{1}{\sigma_p^2}. \end{aligned} \tag{9.13}$$

We then approximate the posterior by a multivariate normal so that

$$Pr(\boldsymbol{\phi}|\mathbf{X}, \mathbf{w}) \approx q(\boldsymbol{\phi}) = \text{Norm}_{\boldsymbol{\phi}}[\boldsymbol{\mu}, \boldsymbol{\Sigma}], \tag{9.14}$$

where the mean $\boldsymbol{\mu}$ is set to the MAP estimate $\hat{\boldsymbol{\phi}}$, and the covariance $\boldsymbol{\Sigma}$ is chosen so that the second derivatives of the normal match those of the posterior distribution at the MAP estimate (figure 9.7) so that

Problem 9.7

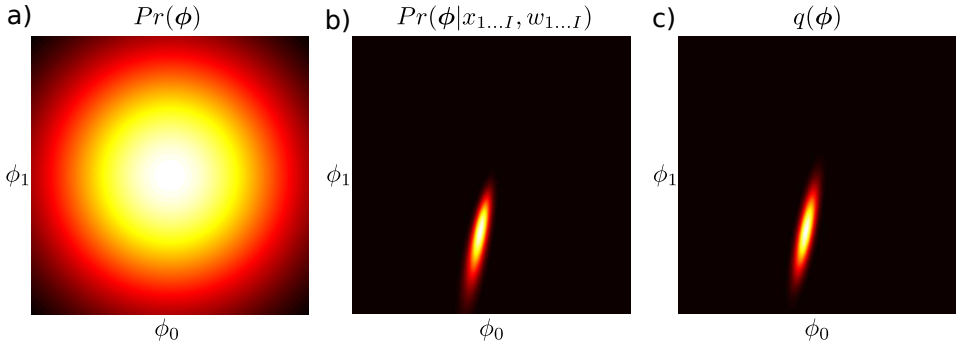


Figure 9.7 Laplace approximation for logistic regression. a) The prior $Pr(\boldsymbol{\phi})$ over the parameters is a normal distribution with mean zero and a large spherical covariance. b) The posterior distribution $Pr(\boldsymbol{\phi}|\mathbf{X}, \mathbf{w})$ represents the refined state of our knowledge after observing the data. Unfortunately, this posterior cannot be expressed in closed form. c) We approximate the true posterior with a normal distribution $q(\boldsymbol{\phi}) = \text{Norm}_{\boldsymbol{\phi}}[\boldsymbol{\mu}, \boldsymbol{\Sigma}]$ whose mean is at the peak of the posterior and whose covariance is chosen so that the second derivatives at the peak of the true posterior match the second derivatives at the peak of the normal. This is termed the Laplace approximation.

$$\begin{aligned}\boldsymbol{\mu} &= \hat{\boldsymbol{\phi}} \\ \boldsymbol{\Sigma} &= -\left(\frac{\partial^2 L}{\partial \boldsymbol{\phi}^2}\right)^{-1}\Bigg|_{\boldsymbol{\phi}=\hat{\boldsymbol{\phi}}}.\end{aligned} \tag{9.15}$$

9.2.2 Inference

In inference we aim to compute a posterior distribution $Pr(w^*|\mathbf{x}^*, \mathbf{X}, \mathbf{w})$ over the world state w^* given new observed data $\mathbf{x}^*$. To this end, we compute an infinite weighted sum (i.e., an integral) of the predictions $Pr(w^*|\mathbf{x}^*, \boldsymbol{\phi})$ given by each possible value of the parameters $\boldsymbol{\phi}$,

$$\begin{aligned}Pr(w^*|\mathbf{x}^*, \mathbf{X}, \mathbf{w}) &= \int Pr(w^*|\mathbf{x}^*, \boldsymbol{\phi}) Pr(\boldsymbol{\phi}|\mathbf{X}, \mathbf{w}) d\boldsymbol{\phi} \\ &\approx \int Pr(w^*|\mathbf{x}^*, \boldsymbol{\phi}) q(\boldsymbol{\phi}) d\boldsymbol{\phi}.\end{aligned} \tag{9.16}$$

where the weights $q(\boldsymbol{\phi})$ are given by the approximated posterior distribution over the parameters from the learning stage. Unfortunately, this integral cannot be computed in closed form either, and so we must make a further approximation.

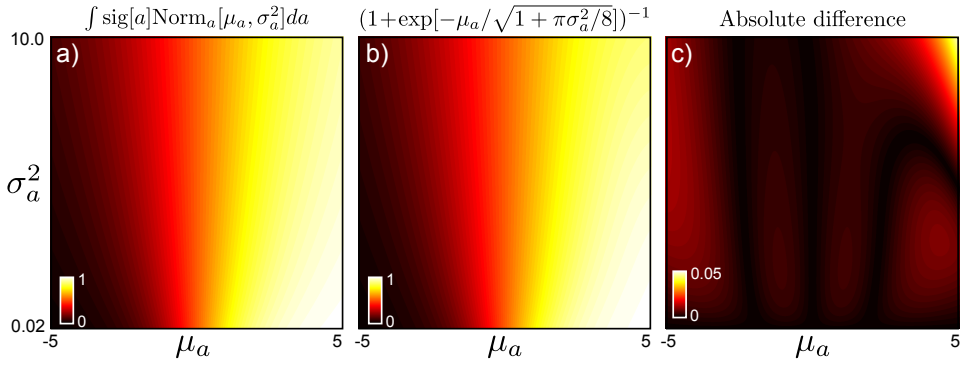


Figure 9.8 Approximation of activation integral (equation 9.19). a) Actual result of integral as a function of μ_a and σ_a^2. b) The (non-obvious) approximation from equation 9.19. c) The absolute difference between the actual result and the approximation is very small over a range of reasonable values.

We first note that the prediction $Pr(w^*|\mathbf{x}^*, \boldsymbol{\phi})$ depends only on a linear projection $a = \boldsymbol{\phi}^T\mathbf{x}^*$ of the parameters (see equation 9.4). Hence we could re-express the prediction as

$$Pr(w^*|\mathbf{x}^*, \mathbf{X}, \mathbf{w}) \approx \int Pr(w^*|a)Pr(a)da. \tag{9.17}$$

The probability distribution $Pr(a)$ can be computed using the transformation property of the normal distribution (section 5.3) and is given by

$$\begin{aligned} Pr(a) = Pr(\boldsymbol{\phi}^T\mathbf{x}^*) &= \text{Norm}_a[\boldsymbol{\mu}^T\mathbf{x}^*, \mathbf{x}^{*T}\boldsymbol{\Sigma}\mathbf{x}] \\ &= \text{Norm}_a[\mu_a, \sigma_a^2], \end{aligned} \tag{9.18}$$

where we have denoted the mean and variance of the activation by μ_a and σ_a^2, respectively. The one dimensional integration in equation 9.17 can now be computed using numerical integration over a, or we can approximate the result with a similar function such as

$$\int Pr(w^*|a)\text{Norm}_a[\mu_a, \sigma_a^2]da \approx \frac{1}{1+\exp[-\mu_a/\sqrt{1+\pi\sigma_a^2/8}]}. \tag{9.19}$$

It is not obvious by inspection that this function should approximate the integral well; however, figure 9.8 demonstrates that the approximation is quite accurate.

Figure 9.9 compares the classification predictions $Pr(w^*|\mathbf{x}^*)$ for the maximum likelihood and Bayesian approaches for logistic regression. The Bayesian approach makes more moderate predictions for the final class. This is particularly the case in regions of data space that are far from the mean.

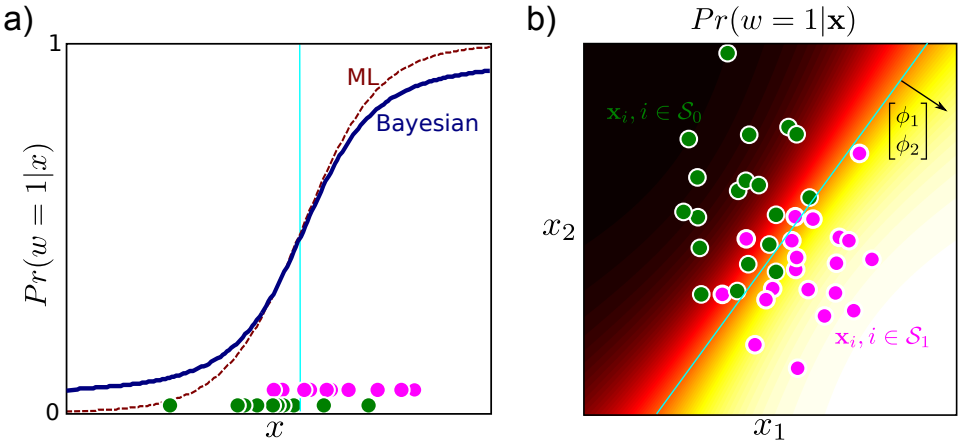


Figure 9.9 Bayesian logistic regression predictions. a) The Bayesian prediction for the class w is more moderate than the maximum likelihood prediction. b) In 2D the decision boundary in the Bayesian case (blue line) is still linear but iso-probability contours at levels other than 0.5 are curved (compare to maximum likelihood case in figure 9.3b). Here too, the Bayesian solution makes more moderate predictions than the maximum likelihood model.

9.3 Non-linear logistic regression

The logistic regression model described previously can only create linear decision boundaries between classes. To create nonlinear decision boundaries, we adopt the same approach as we did for regression (section 8.3): we compute a nonlinear transformation $\mathbf{z} = \mathbf{f}[\mathbf{x}]$ of the observed data and then build the logistic regression model substituting the original data $\mathbf{x}$ for the transformed data $\mathbf{z}$, so that

$$Pr(w=1|\mathbf{x},\boldsymbol{\phi}) = \text{Bern}_w\left[\text{sig}[\boldsymbol{\phi}^T\mathbf{z}]\right] = \text{Bern}_w\left[\text{sig}[\boldsymbol{\phi}^T\mathbf{f}[\mathbf{x}]]\right]. \tag{9.20}$$

The logic of this approach is that arbitrary nonlinear activations can be built as a linear sum of nonlinear basis functions. Typical nonlinear transformations include

- Heaviside step functions of projections: $z_k = \text{heaviside}[\boldsymbol{\alpha}_k^T\mathbf{x}]$,
- arc tangent functions of projections: $z_k = \arctan[\boldsymbol{\alpha}_k^T\mathbf{x}]$, and
- radial basis functions: $z_k = \exp[-\frac{1}{\lambda_0}(\mathbf{x}-\boldsymbol{\alpha}_k)^T(\mathbf{x}-\boldsymbol{\alpha}_k)]$

where z_k denotes the k^{th} element of the transformed vector $\mathbf{z}$ and the function heaviside[$\bullet$] returns zero if its argument is less than zero and one otherwise. In the first two cases we have attached a 1 to the start of the observed data $\mathbf{x}$ where we use projections $\boldsymbol{\alpha}^T\mathbf{x}$ to avoid having a separate offset parameter. Figures 9.10 and 9.11 show examples of nonlinear classification using arc tangent functions for one- and two-dimensional data, respectively.

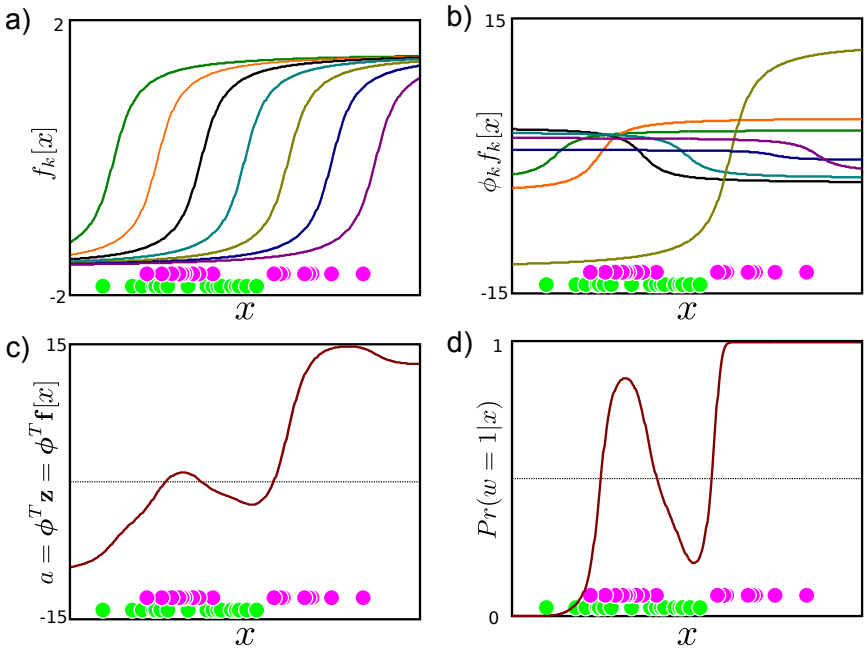


Figure 9.10 Non-linear classification in 1D using arc tangent transformation. We consider a complex 1D data set (bottom of all panels) where the posterior $Pr(w=1|x)$ cannot easily be described by a single sigmoid. Green circles represent data x_i where $w_i = 0$. Pink circles represent data x_i where $w_i = 1$. a) The seven dimensional transformed data vectors $z_1 \ldots z_I$ are computed by evaluating each data example against seven predefined arc tangent functions $z_{ik} = f_k[x_i] = \arctan[\alpha_{0k} + \alpha_{1k} x_i]$. b) When we learn the parameters $\boldsymbol{\phi}$ we are learning weights for these nonlinear arc tangent functions. The functions are shown after applying the maximum likelihood weights $\hat{\boldsymbol{\phi}}$. c) The final activation $a = \boldsymbol{\phi}^T \mathbf{z}$ is a weighted sum of the nonlinear functions. d) The probability $Pr(w=1|x)$ is computed by passing the activation a through the logistic sigmoid function.

Note that the basis functions also have parameters. For example, in the arc tangent example, there are the projection directions $\{\boldsymbol{\alpha}_k\}_{k=1}^K$ each of which contains an offset and a set of gradients. These can also be optimized during the fitting procedure together with the weights $\boldsymbol{\phi}$. We form a new vector of unknowns $\boldsymbol{\theta} = [\boldsymbol{\phi}^T, \boldsymbol{\alpha}_1^T, \boldsymbol{\alpha}_2^T, \ldots, \boldsymbol{\alpha}_K^T]^T$ and optimize the model with respect to all of these unknowns together. The gradient vector and the Hessian matrix depend on the chosen transformation $\mathbf{f}[\bullet]$ but can be computed using the expressions

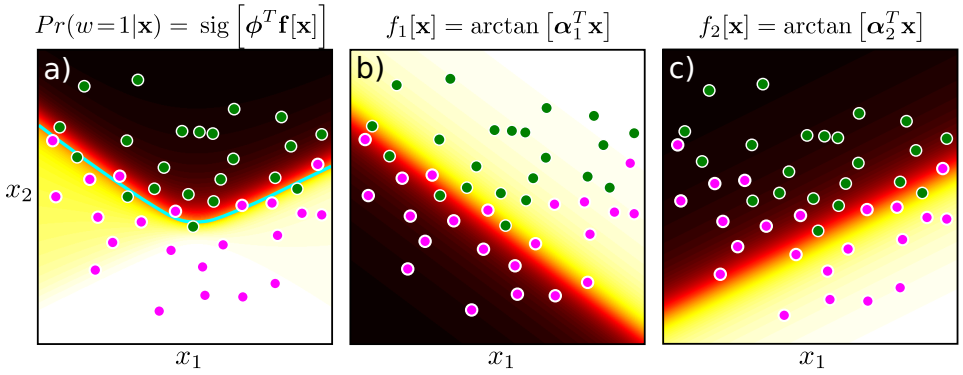


Figure 9.11 Non-linear classification in 2D using arc tangent transform. a) These data have been successfully classified with nonlinear logistic regression. Note the nonlinear decision boundary (cyan line). To compute the posterior $Pr(w=1|\mathbf{x})$, we transform the data to a new two-dimensional space $\mathbf{z} = \mathbf{f}[\mathbf{x}]$ where the elements of $\mathbf{z}$ are computed by evaluating $\mathbf{x}$ against the 1D arc tangent functions in b) and c). The arc tangent activations are weighted (the first by a negative number) and summed and the result is put through the logistic sigmoid to compute $Pr(w=1|\mathbf{x})$.

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{\theta}} &= -\sum_{i=1}^{I}(w_i - \text{sig}[a_i])\frac{\partial a_i}{\partial \boldsymbol{\theta}} \\ \frac{\partial^2 L}{\partial \boldsymbol{\theta}^2} &= -\sum_{i=1}^{I}\text{sig}[a_i](\text{sig}[a_i]-1)\frac{\partial a_i}{\partial \boldsymbol{\theta}}\frac{\partial a_i}{\partial \boldsymbol{\theta}}^T - (w_i - \text{sig}[a_i])\frac{\partial^2 a_i}{\partial \boldsymbol{\theta}^2}, \end{aligned} \quad (9.21)$$

where $a_i = \boldsymbol{\phi}^T\mathbf{f}[\mathbf{x}_i]$. These relations were established using the chain rule for derivatives. Unfortunately, this joint optimization problem is generally not convex and will be prone to terminating in local maxima. In the Bayesian case, it would be typical to marginalize over the parameters $\boldsymbol{\phi}$ but maximize over the function parameters.

9.4 Dual logistic regression

There is a potential problem with the logistic regression models as described earlier: in the original linear model, there is one element of the gradient vector $\boldsymbol{\phi}$ corresponding to each dimension of the observed data $\mathbf{x}$, and in the nonlinear extension there is one element corresponding to each transformed data dimension $\mathbf{z}$. If the relevant data $\mathbf{x}$ (or $\mathbf{z}$) is very high-dimensional, then the model will have a large number of parameters: this will render the Newton update slow or even intractable.

Algorithm 9.3

To solve this problem, we switch to the *dual* representation. For simplicity, we will develop this model using the original data $\mathbf{x}$, but all of the ideas transfer directly to the nonlinear case where we use transformed data $\mathbf{z}$.

In the *dual* parameterization, we express the gradient parameters $\boldsymbol{\phi}$ as a weighted sum of the observed data (see figure 8.12) so that

$$\boldsymbol{\phi} = \mathbf{X}\boldsymbol{\psi}, \tag{9.22}$$

where $\boldsymbol{\psi}$ is an $I \times 1$ variable where each element weights one of the data examples. If the number of data points I is less than the dimensionality D of the data $\mathbf{x}$, then the number of parameters has been reduced.

The price that we pay for this reduction is that we can now only choose gradient vectors $\boldsymbol{\phi}$ that are in the space spanned by the data examples. However, the gradient vector represents the direction in which the final probability $Pr(w=1|\mathbf{x})$ changes fastest, and this should not point in a direction in which there was no variation in the training data anyway, so this is not a limitation.

Substituting equation 9.22 into the original logistic regression model leads to the dual logistic regression model,

$$Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\psi}) = \prod_{i=1}^{I} \text{Bern}_{w_i}\left[\text{sig}[a_i]\right] = \prod_{i=1}^{I} \text{Bern}_{w_i}\left[\text{sig}[\boldsymbol{\psi}^T\mathbf{X}^T\mathbf{x}_i]\right]. \tag{9.23}$$

The resulting learning and inference algorithms are very similar to those for the original logistic regression model, so we cover them only in brief:

- In the maximum likelihood method, we learn the parameters $\boldsymbol{\psi}$ by nonlinear optimization of the log likelihood $L = \log[Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\psi})]$ using the Newton method. This optimization requires the derivatives of the log likelihood, which are

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{\psi}} &= -\sum_{i=1}^{I}\left(\text{sig}[a_i] - w_i\right)\mathbf{X}^T\mathbf{x}_i \\ \frac{\partial^2 L}{\partial \boldsymbol{\psi}^2} &= -\sum_{i=1}^{I}\text{sig}[a_i]\left(1-\text{sig}[a_i]\right)\mathbf{X}^T\mathbf{x}_i\mathbf{x}_i^T\mathbf{X}.\end{aligned} \tag{9.24}$$

- In the Bayesian approach, we use a normal prior over the parameters $\boldsymbol{\psi}$,

$$Pr(\boldsymbol{\psi}) = \text{Norm}_{\boldsymbol{\psi}}[\mathbf{0}, \sigma_p^2\mathbf{I}]. \tag{9.25}$$

  The posterior distribution $Pr(\boldsymbol{\psi}|\mathbf{X}, \mathbf{w})$ over the new parameters is found using Bayes' rule, and once more this cannot be written in closed form, so we apply the Laplace approximation; we find the MAP solution $\hat{\boldsymbol{\psi}}$ using nonlinear optimization, which requires the derivatives of the log posterior $L = \log[Pr(\boldsymbol{\psi}|\mathbf{X}, \mathbf{w})]$:

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{\psi}} &= -\sum_{i=1}^{I}\left(\text{sig}[a_i]-w_i\right)\mathbf{X}^T\mathbf{x}_i - \frac{\boldsymbol{\psi}}{\sigma_p^2} \\ \frac{\partial^2 L}{\partial \boldsymbol{\psi}^2} &= -\sum_{i=1}^{I}\text{sig}[a_i]\left(1-\text{sig}[a_i]\right)\mathbf{X}^T\mathbf{x}_i\mathbf{x}_i^T\mathbf{X} - \frac{1}{\sigma_p^2}. \end{aligned} \tag{9.26}$$

The posterior is now approximated by a multivariate normal so that

$$Pr(\boldsymbol{\psi}|\mathbf{X},\mathbf{w}) \approx q(\boldsymbol{\psi}) = \text{Norm}_{\boldsymbol{\psi}}[\boldsymbol{\mu},\boldsymbol{\Sigma}], \tag{9.27}$$

where

$$\begin{aligned}\boldsymbol{\mu} &= \hat{\boldsymbol{\psi}} \\ \boldsymbol{\Sigma} &= -\left(\frac{\partial^2 L}{\partial \boldsymbol{\psi}^2}\right)^{-1}\Bigg|_{\boldsymbol{\psi}=\hat{\boldsymbol{\psi}}}. \end{aligned} \tag{9.28}$$

In inference, we compute the distribution over the activation

$$\begin{aligned} Pr(a) = Pr(\boldsymbol{\psi}^T\mathbf{X}^T\mathbf{x}^*) &= \text{Norm}_a[\mu_a,\sigma_a^2] \\ &= \text{Norm}_a[\boldsymbol{\mu}^T\mathbf{X}^T\mathbf{x}^*, \mathbf{x}^{*T}\mathbf{X}\boldsymbol{\Sigma}\mathbf{X}^T\mathbf{x}^*], \end{aligned} \tag{9.29}$$

and then approximate the predictive distribution using equation 9.19.

Dual logistic regression gives identical results to the original logistic regression algorithm for the maximum likelihood case and very similar results in the Bayesian situation (where the difference results from the slightly different priors). However, the dual classification model is much faster to fit in high dimensions as the parameters are fewer.

9.5 Kernel logistic regression

We motivated the dual model by the reduction in the number of parameters $\boldsymbol{\psi}$ in the model when the data lies in a high dimensional space. However, now that we have developed the model, a further advantage is easy to identify: both learning and inference in the dual model rely only on inner products $\mathbf{x}_i^T\mathbf{x}_j$ of that data. Equivalently, the nonlinear version of this algorithm depends only on inner products $\mathbf{z}_i^T\mathbf{z}_j$ of the transformed data vectors. This means that the algorithm is suitable for *kernelization* (see section 8.4).

Algorithm 9.4

The idea of kernelization is to define a kernel function $k[\bullet,\bullet]$, which computes the quantity

$$\mathrm{k}[\mathbf{x}_i, \mathbf{x}_j] = \mathbf{z}_i^T \mathbf{z}_j, \tag{9.30}$$

where $\mathbf{z}_i = \mathbf{f}[\mathbf{x}_i]$ and $\mathbf{z}_j = \mathbf{f}[\mathbf{x}_j]$ are the nonlinear transformations of the two data vectors. Replacing the inner products with the kernel function means that we do not have to explicitly calculate the transformed vectors $\mathbf{z}$ and hence they may be of very high, or even infinite dimensions. See section 8.4 for a more detailed description of kernel functions.

The kernel logistic regression model (compare to equation 9.23) is hence

$$Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\psi}) = \prod_{i=1}^{I} \mathrm{Bern}_{w_i}\left[\mathrm{sig}[a_i]\right] = \prod_{i=1}^{I} \mathrm{Bern}_{w_i}\left[\mathrm{sig}[\boldsymbol{\psi}^T \mathbf{K}[\mathbf{X}, \mathbf{x}_i]\right], \tag{9.31}$$

where the notation $\mathbf{K}[\mathbf{X}, \mathbf{x}]_i$ represents a column vector of dot products where element k is given by $\mathrm{k}[\mathbf{x}_k, \mathbf{x}_i]$.

For maximum likelihood learning, we simply optimize the log posterior probability L with respect to the parameters, which requires the derivatives:

$$\begin{aligned} \frac{\partial L}{\partial \boldsymbol{\psi}} &= -\sum_{i=1}^{I} \left(\mathrm{sig}[a_i] - w_i\right) \mathbf{K}[\mathbf{X}, \mathbf{x}_i] \\ \frac{\partial^2 L}{\partial \boldsymbol{\psi}^2} &= -\sum_{i=1}^{I} \mathrm{sig}[a_i]\left(1 - \mathrm{sig}[a_i]\right) \mathbf{K}[\mathbf{X}, \mathbf{x}_i] \mathbf{K}[\mathbf{x}_i, \mathbf{X}]. \end{aligned} \tag{9.32}$$

The Bayesian formulation of kernel logistic regression, which is sometimes known as *Gaussian process classification*, proceeds along similar lines; we follow the dual formulation, replacing each of the dot products between data examples with the kernel function.

Problem 9.8

A very common example of a kernel function is the radial basis kernel in which the nonlinear transformation and inner product operations are replaced by

$$\mathrm{k}[\mathbf{x}_i, \mathbf{x}_j] = \exp\left[-0.5\left(\frac{(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j)}{\lambda^2}\right)\right]. \tag{9.33}$$

This is equivalent to computing transformed vectors $\mathbf{z}_i$ and $\mathbf{z}_j$ of infinite length, where each entry evaluates the data $\mathbf{x}$ against a radial basis function at a different position, and then computing the inner product $\mathbf{z}_i^T \mathbf{z}_j$. Examples of the kernel logistic regression with a radial basis kernel are shown in figures 9.12 and 9.13.

9.6 Relevance vector classification

The Bayesian version of the kernel logistic regression model is powerful, but computationally expensive as it requires us to compute dot products between the new data example and the all of the training examples (in the kernel function in equation 9.31). It would be more efficient if the model depended only sparsely on the

Algorithm 9.5

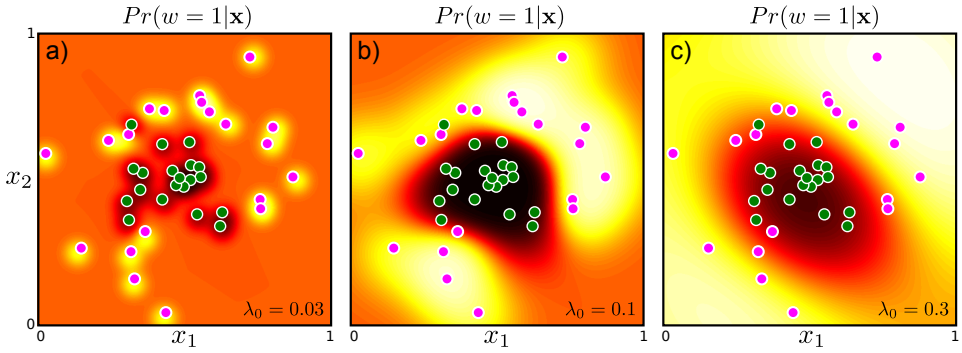


Figure 9.12 Kernel logistic regression using RBF kernel and maximum likelihood learning. a) With a small length scale λ, the model does not interpolate much from the data examples. b) With a reasonable length scale, the classifier does a good job of modeling the posterior $Pr(w=1|\mathbf{x})$. c) With a large length scale, the estimated posterior is very smooth and the model interpolates confident decisions into regions such as the top-left where there is no data.

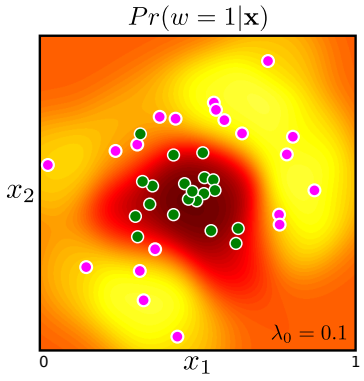


Figure 9.13 Kernel logistic regression with RBF kernel in a Bayesian setting: we now take account of our uncertainty in the dual parameters $\boldsymbol{\psi}$ by approximating their posterior distribution using Laplace's method and marginalizing them out of the model. This produces a very similar result to the maximum likelihood case with the same length scale (figure 9.12b). However, as is typical with Bayesian implementations, the confidence is (appropriately) somewhat lower.

training data. To achieve this, we impose a penalty for every non-zero weighted training example. As in the relevance regression model (section 8.8), we replace the normal prior over the dual parameters $\boldsymbol{\psi}$ (equation 9.25) with a product of one-dimensional t-distributions so that

$$Pr(\boldsymbol{\psi}) = \prod_{i=1}^{I} \text{Stud}_{\boldsymbol{\psi}_i}[0, 1, \nu]. \tag{9.34}$$

Applying the Bayesian approach to this model with respect to the parameters $\boldsymbol{\Psi}$ is known as *relevance vector classification*.

Following the argument of section 8.6, we re-write each student t-distribution as a marginalization of a joint distribution $Pr(\psi_i, h_i)$

$$\begin{aligned} Pr(\boldsymbol{\psi}) &= \prod_{i=1}^{I} \int \text{Norm}_{\psi_i}\left[0, \frac{1}{h_i}\right] \text{Gam}_{h_i}\left[\frac{\nu}{2}, \frac{\nu}{2}\right] \, dh_i \\ &= \int \text{Norm}_{\boldsymbol{\psi}}[0, \mathbf{H}^{-1}] \prod_{d=1}^{D} \text{Gam}_{h_d}[\nu/2, \nu/2] \, d\mathbf{H}, \end{aligned} \tag{9.35}$$

where the matrix $\mathbf{H}$ contains the hidden variables $\{h_i\}_{i=1}^{I}$ on its diagonal and zeros elsewhere. Now we can write the model likelihood as

$$\begin{aligned} &Pr(\mathbf{w}|\mathbf{X}) \\ &= \int Pr(\mathbf{w}|\mathbf{X}, \boldsymbol{\psi}) Pr(\boldsymbol{\psi}) \, d\boldsymbol{\psi} \\ &= \iint \prod_{i=1}^{I} \text{Bern}_{w_i}\Big[\text{sig}[\boldsymbol{\psi}^T \mathbf{K}[\mathbf{X}, \mathbf{x}_i]]\Big] \text{Norm}_{\boldsymbol{\psi}}[0, \mathbf{H}^{-1}] \prod_{d=1}^{D} \text{Gam}_{h_d}[\nu/2, \nu/2] \, d\mathbf{H} d\boldsymbol{\psi}. \end{aligned} \tag{9.36}$$

Now we make two approximations. First, we use the Laplace approximation to describe the first two terms in this integral as a normal distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ centered at the MAP parameters, and use the following result for the integral over $\boldsymbol{\psi}$:

$$\begin{aligned} \int q(\boldsymbol{\psi}) \, d\boldsymbol{\psi} &\approx q(\boldsymbol{\mu}) \int \exp\left[-\frac{1}{2}(\boldsymbol{\psi} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{\psi} - \boldsymbol{\mu})\right] \, d\boldsymbol{\psi} \\ &= q(\boldsymbol{\mu}) (2\pi)^{D/2} |\boldsymbol{\Sigma}|^{1/2}. \end{aligned} \tag{9.37}$$

This yields the expression

$$\begin{aligned} &Pr(\mathbf{w}|\mathbf{X}) \approx \\ &\int \prod_{i=1}^{I} (2\pi)^{I/2} |\boldsymbol{\Sigma}|^{0.5} \text{Bern}_{w_i}\big[\text{sig}[\boldsymbol{\mu}^T \mathbf{K}[\mathbf{X}, \mathbf{x}_i]]\big] \text{Norm}_{\boldsymbol{\mu}}\big[0, \mathbf{H}^{-1}\big] \text{Gam}_{h_i}\Big[\frac{\nu}{2}, \frac{\nu}{2}\Big] \, d\mathbf{H}, \end{aligned} \tag{9.38}$$

where the matrix $\mathbf{H}$ contains the hidden variables $\{h_i\}_{i=1}^{I}$ on the diagonal, and we have used the general result for the Laplace approximation.

In the second approximation, we maximize over the hidden variables, rather than integrate over them. This yields the expression:

$$\begin{aligned} &Pr(\mathbf{w}|\mathbf{X}) \approx \\ &\max_{\mathbf{H}} \left[\prod_{i=1}^{I} (2\pi)^{I/2} |\boldsymbol{\Sigma}|^{0.5} \text{Bern}_{w_i}\big[\text{sig}[\boldsymbol{\mu}^T \mathbf{K}[\mathbf{X}, \mathbf{x}_i]]\big] \text{Norm}_{\boldsymbol{\mu}}\big[0, \mathbf{H}^{-1}\big] \text{Gam}_{h_i}\Big[\frac{\nu}{2}, \frac{\nu}{2}\Big] \right]. \end{aligned} \tag{9.39}$$

To learn the model, we now alternate between updating the mean and variance $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ of the posterior distribution and updating the hidden variables $\{h_i\}$. To update the mean and variance parameters, we find the solution $\hat{\boldsymbol{\psi}}$ that maximizes

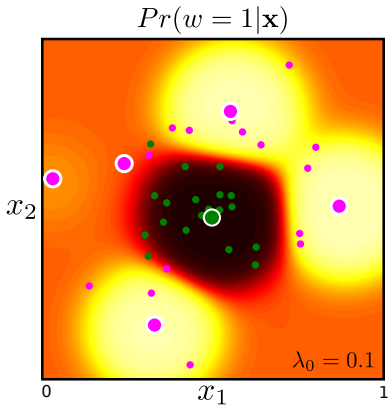


Figure 9.14 Relevance vector regression with RBF kernel. We place a prior over the dual parameters $\boldsymbol{\psi}$ that encourages sparsity. After learning, the posterior distribution over most of the parameters is tightly centered around zero and they can be dropped from the model. Large points indicate data examples associated with non-zero dual parameters. The solution here can be computed from just 6 of the 40 data points but nonetheless classifies the data almost as well as the full kernel approach (figure 9.13).

$$L = \sum_{i=1}^{I} \log\left[\text{Bern}_{w_i}\left[\text{sig}[\boldsymbol{\psi}^T\mathbf{K}[\mathbf{X},\mathbf{x}_i]]\right]\right] + \log\left[\text{Norm}_{\boldsymbol{\psi}}[0,\mathbf{H}^{-1}]\right] \tag{9.40}$$

using the derivatives

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{\psi}} &= -\sum_{i=1}^{I}\left(\text{sig}[a_i]-w_i\right)\mathbf{K}[\mathbf{X},\mathbf{x}_i] - \mathbf{H}\psi \\ \frac{\partial^2 L}{\partial \boldsymbol{\psi}^2} &= -\sum_{i=1}^{I}\text{sig}[a_i]\left(1-\text{sig}[a_i]\right)\mathbf{K}[\mathbf{X},\mathbf{x}_i]\mathbf{K}[\mathbf{x}_i\mathbf{X}] - \mathbf{H},\end{aligned} \tag{9.41}$$

and then set

$$\begin{aligned}\boldsymbol{\mu} &= \hat{\boldsymbol{\psi}} \\ \boldsymbol{\Sigma} &= -\left(\frac{\partial^2 L}{\partial \boldsymbol{\psi}^2}\right)^{-1}\Bigg|_{\boldsymbol{\psi}=\hat{\boldsymbol{\psi}}}.\end{aligned} \tag{9.42}$$

To update the hidden variables h_i we use the same expression as for relevance vector regression:

$$h_i^{new} = \frac{1-h_i\Sigma_{ii}+\nu}{\mu_i^2+\nu}. \tag{9.43}$$

As this optimization proceeds, some of the hidden variables h_i will become very large. This means that the prior over the relevant parameter becomes very concentrated around zero and that the associated data points contribute nothing to the final solution. These can be removed, leaving a kernelized classifier that depends only sparsely on the data and can hence be evaluated very efficiently.

In inference, we aim to compute the distribution over the world state w^* given a new data example $\mathbf{x}^*$. We take the familiar strategy of approximating the posterior distribution over the activation as

$$\begin{aligned} Pr(a) = Pr(\boldsymbol{\psi}^T\mathbf{K}[\mathbf{X}^T, \mathbf{x}^*]) &= \text{Norm}_a[\mu_a, \sigma_a^2]r \\ &= \text{Norm}_a[\boldsymbol{\mu}^T\mathbf{K}[\mathbf{X}, \mathbf{x}^*], \mathbf{K}[\mathbf{x}^*, \mathbf{X}]\boldsymbol{\Sigma}\mathbf{K}[\mathbf{X}, \mathbf{x}^*]], \end{aligned} \tag{9.44}$$

and then approximate the predictive distribution using equation 9.19.

An example of relevance vector classification is shown in figure 9.14, which shows that the data set can be discriminated based on 6 of the original 40 data points. This results in a considerable computational saving and the simpler solution guards against over-fitting of the training set.

9.7 Incremental fitting and boosting

In the previous section, we developed the *relevance vector classification* model in which we applied a prior that encourages sparsity in the dual logistic regression parameters $\boldsymbol{\psi}$ and hence encouraged the model to depend on only a subset of the training data. It is similarly possible to develop a *sparse logistic regression method* by placing a prior that encourages sparsity in the original parameters $\boldsymbol{\phi}$ and hence encourages the classifier to depend only on a subset of the data dimensions. This is left as an exercise to the reader.

Algorithm 9.6

In this section we will investigate a different approach to inducing sparsity; we will add one parameter at a time to the model in a greedy fashion; in other words, we add the parameter that improves the objective function most at each stage and then consider this fixed. As the most discriminative parts of the model are added first, it is possible to truncate this process after only a small fraction of the parameters are added and still achieve good results. The remaining, unused parameters can be considered as having a value of zero and so this model also provides a sparse solution. We term this approach *incremental fitting*. We will work with the original formulation (so that the sparsity is over the data dimensions), although these ideas can equally be adapted to the dual case.

To describe the incremental fitting procedure, let us work with the nonlinear formulation of logistic regression (section 9.3) where the probability of the class given the data was described as

$$Pr(w_i|\mathbf{x}_i) = \text{Bern}_{w_i}[\text{sig}[a_i]], \tag{9.45}$$

where $\text{sig}[\bullet]$ is the logistic sigmoid function and the activation a_i is given by

$$a_i = \boldsymbol{\phi}^T\mathbf{z}_i = \boldsymbol{\phi}^T\mathbf{f}[\mathbf{x}_i], \tag{9.46}$$

and $\mathbf{f}[\bullet]$ is a nonlinear transformation that returns the transformed vector $\mathbf{z}_i$.

To simplify the subsequent description we will now write the activation term in a slightly different way so that the dot product is described explicitly as a weighted sum of individual nonlinear functions of the data

$$a_i = \phi_0 + \sum_{k=1}^{K} \phi_k f[\mathbf{x}_i, \boldsymbol{\xi}_k]. \tag{9.47}$$

Here $f[\bullet, \bullet]$ is a fixed nonlinear function that takes the data vector $\mathbf{x}_i$ and some parameters $\boldsymbol{\xi}_k$ and returns a scalar value. In other words the k^{th} entry of the transformed vector $\mathbf{z}$ arises by passing the data $\mathbf{x}$ through the function with the k^{th} parameters $\boldsymbol{\xi}_k$. Example functions $f[\bullet, \bullet]$ might include

- radial basis functions, $\boldsymbol{\xi} = \{\boldsymbol{\alpha}, \lambda_0\}$

$$f[\mathbf{x}, \boldsymbol{\xi}] = \exp\left[-\frac{(\mathbf{x}-\boldsymbol{\alpha})^T(\mathbf{x}-\boldsymbol{\alpha})}{\lambda_0^2}\right], \quad \text{and} \tag{9.48}$$

- arc tan functions, $\boldsymbol{\xi} = \{\boldsymbol{\alpha}\}$

$$f[\mathbf{x}, \boldsymbol{\xi}] = \arctan[\boldsymbol{\alpha}^T\mathbf{x}]. \tag{9.49}$$

In incremental learning, we construct the activation term in equation 9.47 piecewise. At each stage we add a new term, leaving all of the previous terms unchanged except the additive constant ϕ_0. So, at the first stage we use the activation

$$a_i = \phi_0 + \phi_1 f[\mathbf{x}_i, \boldsymbol{\xi}_1] \tag{9.50}$$

and learn the parameters ϕ_0, ϕ_1 and $\boldsymbol{\xi}_1$ using the maximum likelihood approach. At the second stage, we fit the function

$$a_i = \phi_0 + \phi_1 f[\mathbf{x}_i, \boldsymbol{\xi}_1] + \phi_2 f[\mathbf{x}_i, \boldsymbol{\xi}_2] \tag{9.51}$$

and learn the parameters ϕ_0, ϕ_2 and $\boldsymbol{\xi}_2$, while keeping the remaining parameters ϕ_1 and $\boldsymbol{\xi}_1$ constant. At the K^{th} stage, we fit a model with activation

$$a_i = \phi_0 + \sum_{k=1}^{K} \phi_k f[\mathbf{x}_i, \boldsymbol{\xi}_k] \tag{9.52}$$

and learn the parameters ϕ_0, ϕ_K and $\boldsymbol{\xi}_K$, while keeping the remaining parameters $\phi_1 \ldots \phi_{k-1}$ and $\boldsymbol{\xi}_1 \ldots \boldsymbol{\xi}_{k-1}$ constant.

At each stage, the learning is carried out using the maximum likelihood approach. We use a nonlinear optimization procedure to maximize the log posterior probability L with respect to the relevant parameters. The derivatives required by the optimization procedure depend on the choice of nonlinear function, but can be computed using the chain rule relations (equation 9.21).

This procedure is obviously sub-optimal as we do not learn the parameters together or even revisit early parameters once they have been set. However, it has three nice properties.

1. It creates sparse models: the weights ϕ_k tend to decrease as we move through the sequence and each subsequent basis function tends to have less influence on the model. Consequently, the series can be truncated to the desired length and the associated performance is likely to remain good.

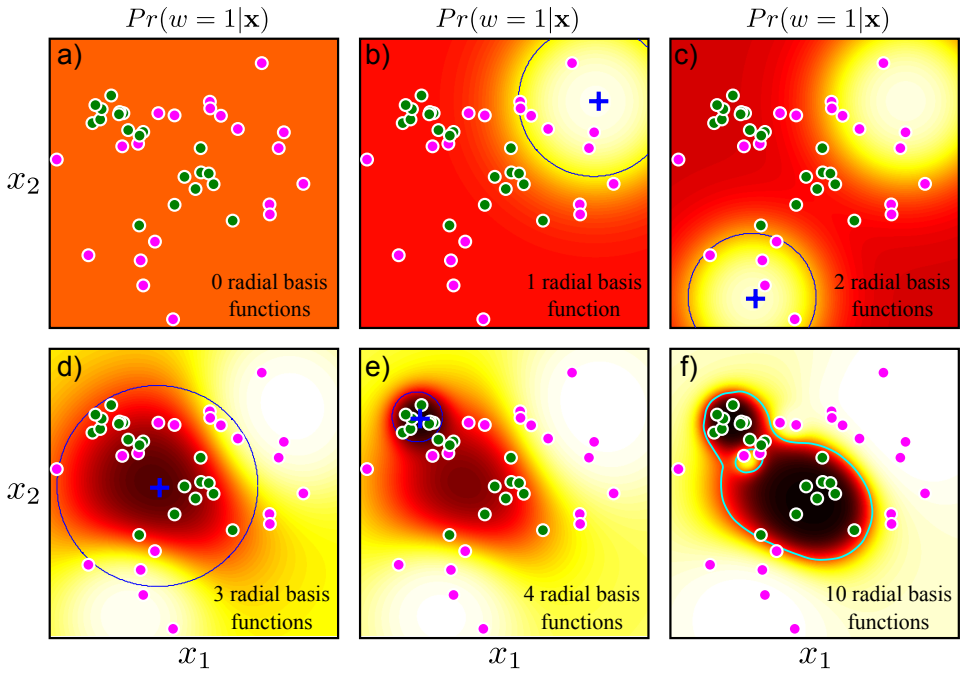


Figure 9.15 Incremental approach to fitting nonlinear logistic regression model with RBF functions. a) Before fitting, the activation (and hence the posterior probability) is uniform. b) Posterior probability after fitting one function (mean and scale of RBF shown in blue). c-e) After fitting two, three and four RBFs. f) After fitting ten RBFs. The data are now all classified correctly as can be seen from the decision boundary (cyan line).

2. The previous logistic regression models have been suited to cases where either the dimensionality D of the data is small (original formulation) or the number of training examples I is small (dual formulation). However, it is quite possible that neither of these things is true. A strong advantage of incremental fitting is that it is still practical when the data are high dimensional *and* there are a large number of training examples. During training, we do not need to hold all of the transformed vectors $\mathbf{z}$ in memory at once: at the K^{th} stage, we need only the K^{th} dimension of the transformed parameters $z_K = f[\mathbf{x}, \boldsymbol{\xi}_K]$ and the aggregate of the previous contributions to activation term $\sum_{k=1}^{K-1} \phi_k f[\mathbf{x}_i, \boldsymbol{\xi}_k]$.

3. Learning is relatively inexpensive because we only optimize a few parameters at each stage.

Figure 9.15 illustrates the incremental approach to learning a 2D data set using radial basis functions. Notice that even after only a few functions have been added to the sequence, the classification is substantially correct. Nonetheless, it is worth

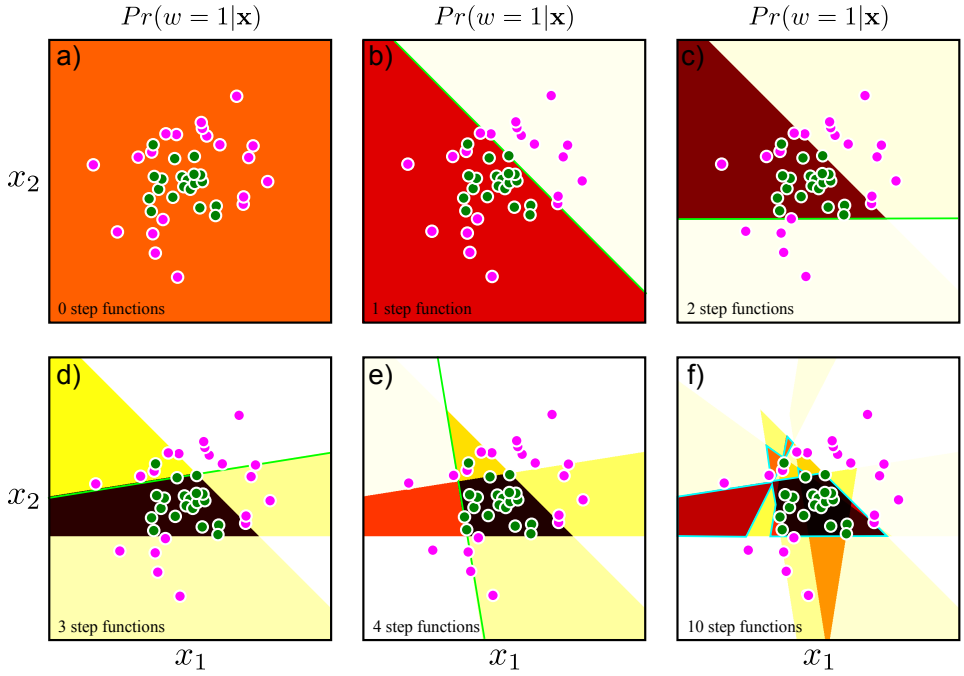


Figure 9.16 Boosting. a) We start with a uniform prediction $Pr(w=1|\mathbf{x})$ and b) incrementally add a step function to the activation (green line indicates position of step). In this case the parameters of the step function were chosen greedily from a pre-determined set containing 20 angles each with 40 offsets. c)-e) As subsequent functions are added the overall classification improves. f) However, the final decision surface (cyan line) is complex and does not interpolate smoothly between regions of high confidence.

continuing to train this model even after the training data are classified correctly. Usually the model continues to improve and the classification performance on test data will continue to increase for some time.

9.7.1 Boosting

There is a special case of the incremental approach to fitting nonlinear logistic regression that is commonly used in vision applications. Consider a logistic regression model based on a sum of step functions

Algorithm 9.7

$$a_i = \phi_0 + \sum_{k=1}^{K} \phi_k \text{heaviside}[\boldsymbol{\alpha}_k^T \mathbf{x}_i], \tag{9.53}$$

where the function heaviside[•] returns 0 if its argument is less than 0 and 1 otherwise. As usual, we have attached a 1 to the start of the data $\mathbf{x}$ so that the

parameters $\boldsymbol{\alpha}_k$ contain both a direction $[\alpha_{k1}, \alpha_{k2}, \ldots, \alpha_{kD}]$ in the D dimensional space (which determines the direction of the step function) and an offset α_{k0} (that determines where the step occurs).

One way to think about the step functions is as *weak classifiers*; they return 0 or 1 depending on the value of x_i so each classifies the data. The model combines these weak classifiers to compute a final *strong classifier*. Schemes for combining weak classifiers in this way are generically known as *boosting* and this particular model is called *logitboost*.

Unfortunately, we cannot simply fit this model using a gradient-based optimization approach because the derivative of the heaviside step function with respect to the parameters $\boldsymbol{\alpha}_k$ is not smooth. Consequently it is usual to predefine a large set of J weak classifiers and assume that each parameter vector $\boldsymbol{\alpha}_k$ is taken from this set so that $\boldsymbol{\alpha}_k \in \{\boldsymbol{\alpha}^{(1)} \ldots \boldsymbol{\alpha}^{(J)}\}$.

As before, we learn the logitboost model incrementally by adding one term at a time to the activation (equation 9.53). However, now we exhaustively search over the weak classifiers $\{\boldsymbol{\alpha}^{(1)} \ldots \boldsymbol{\alpha}^{(J)}\}$ and for each, we use nonlinear optimization to estimate the weights ϕ_0 and ϕ_k. We choose the combination $\{\boldsymbol{\alpha}_k, \phi_0, \phi_k\}$ that improves the log likelihood the most. This procedure may be made even more efficient (but more approximate) by choosing the weak classifier based on the log likelihood after just a single Newton or gradient descent step in the nonlinear optimization stage. When we have selected the best weak classifier $\boldsymbol{\alpha}_k$ we can return and perform the full optimization over the offset ϕ_0 and weight ϕ_k.

Note that after each classifier is added, the relative importance of each data point is effectively changed: the data points contribute to the derivative according to how well they are currently predicted (equation 9.9). Consequently, the later weak classifiers become more specialized to the more difficult parts of the data set that are not well classified by the early ones. Usually, these are close to the final decision boundary.

Figure 9.16 shows several iterations of the boosting procedure. Because the model is composed from step functions, the final classification boundary is irregular and does not interpolate smoothly between the data examples. This is a potential disadvantage of this approach. In general, a classifier based on arc tangent functions (which are roughly smooth step functions) will have superior generalization and can also be fit using continuous optimization.

9.8 Classification trees

In the nonlinear logistic regression model, we created complex decision boundaries using an activation function that is a linear combination $\boldsymbol{\phi}^T\mathbf{z}$ of nonlinear functions $\mathbf{z} = \mathbf{f}[\mathbf{x}]$ of the data $\mathbf{x}$. We now investigate an alternative method to induce complex decision boundaries: we partition data space into distinct regions and apply a different classifier in each region.

The *branching logistic regression model* has activation,

$$a_i = (1 - g[\mathbf{x}_i, \boldsymbol{\omega}])\boldsymbol{\phi}_0^T\mathbf{x}_i + g[\mathbf{x}_i, \boldsymbol{\omega}]\boldsymbol{\phi}_1^T\mathbf{x}_i. \tag{9.54}$$

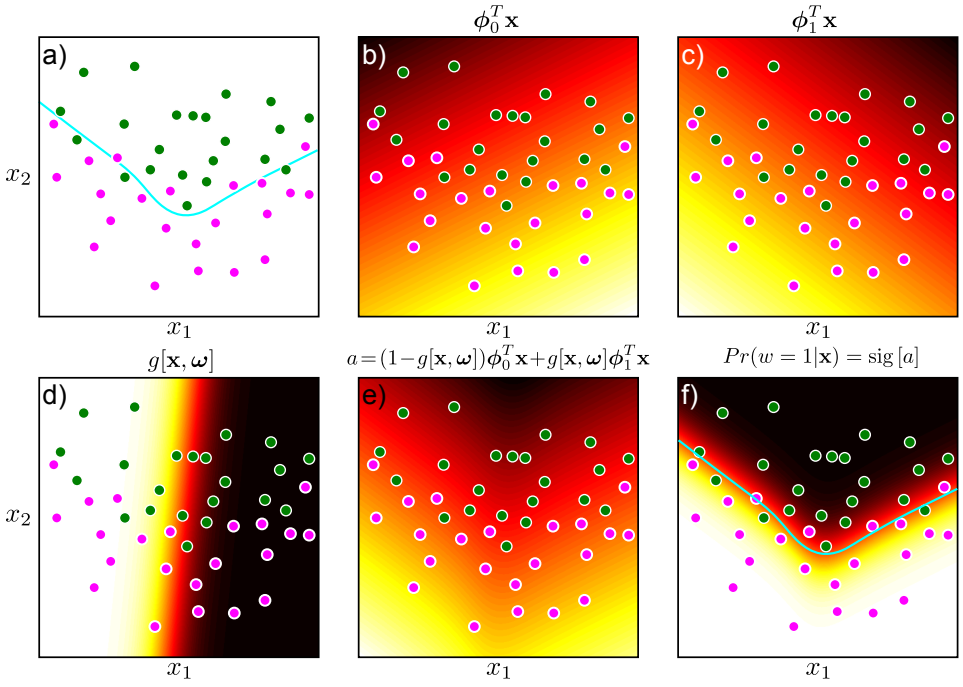


Figure 9.17 Branching logistic regression. a) This data set needs a nonlinear decision surface (cyan line) to classify the data reasonably. b) This linear activation is an *expert* that is specialized to describing the right-hand side of the data. c) This linear activation is an expert that describes the left-hand side of the data. d) A gating function takes the data vector $\mathbf{x}$ and returns a number between 0 and 1, which we will use to decide which expert contributes at each decision. e) The final activation consists of a weighted sum of the activation indicated by the two experts where the weight comes from the gating function. f) The final classifier predictions $Pr(w=1|\mathbf{x})$ are generated by passing this activation through the logistic sigmoid function.

The term $g[\bullet,\bullet]$ is a *gating function* that returns a number between 0 and 1. If this gating function returns 0, then the activation will be $\boldsymbol{\phi}_0\mathbf{x}_i$, whereas if it returns 1, the activation will be $\boldsymbol{\phi}_1\mathbf{x}_i$. If the gating returns an intermediate value, then the activation will be a weighted sum of these two components. The gating function itself depends on the data $\mathbf{x}_i$ and takes parameters $\boldsymbol{\omega}$. This model induces a complex nonlinear decision boundary (figure 9.17) where the two linear functions $\boldsymbol{\phi}_0\mathbf{x}_i$ and $\boldsymbol{\phi}_1\mathbf{x}_i$ are specialized to different regions of the data space. In this context, they are sometimes referred to as *experts*.

The gating function could take many forms, but an obvious possibility is to use a second logistic regression model. In other words, we compute a linear function $\boldsymbol{\omega}^T\mathbf{x}_i$ of the data that is passed through a logistic sigmoid so that

$$g[\mathbf{x}_i,\boldsymbol{\omega}] = \text{sig}[\boldsymbol{\omega}^T\mathbf{x}_i]. \tag{9.55}$$

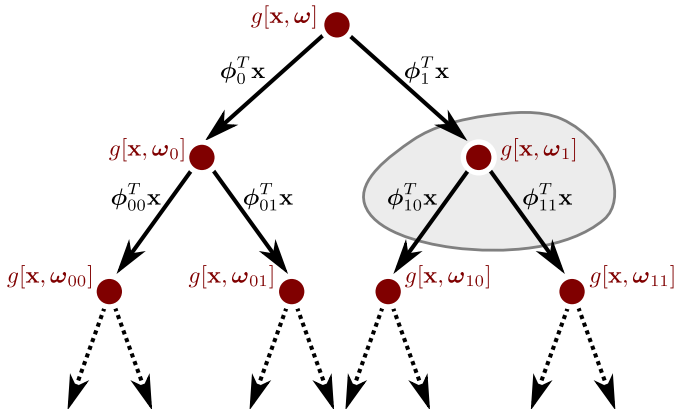


Figure 9.18 Logistic classification tree. Data flows from the root to the leaves. Each node is a gating function that weights the contributions of terms in the sub-branches in the final activation. The gray region indicates variables that would be learned together in an incremental training approach.

To learn this model we maximize the log probability $L = \sum_i \log[Pr(w_i|\mathbf{x}_i)]$ of the training data pairs $\{\mathbf{x}_i, w_i\}_{i=1}^I$ with respect to all of the parameters $\boldsymbol{\theta} = \{\boldsymbol{\phi}_0, \boldsymbol{\phi}_1, \boldsymbol{\omega}\}$. As usual this can be accomplished using a nonlinear optimization procedure. The parameters can be estimated simultaneously or using a coordinate ascent approach in which we alternately update the three sets of parameters.

Problem 9.9

We can extend this idea to create a hierarchical tree structure by nesting gating functions (figure 9.18). For example, consider the activation

$$\begin{aligned} a_i = \; & (1 - g[\mathbf{x}_i, \boldsymbol{\omega}]) \left[\boldsymbol{\phi}_0^T\mathbf{x}_i + (1 - g[\mathbf{x}_i, \boldsymbol{\omega}_0])\boldsymbol{\phi}_{00}^T\mathbf{x}_i + g[\mathbf{x}_i, \boldsymbol{\omega}_0]\boldsymbol{\phi}_{01}^T\mathbf{x}_i\right] \\ & + g[\mathbf{x}_i, \boldsymbol{\omega}] \left[\boldsymbol{\phi}_1^T\mathbf{x}_i + (1 - g[\mathbf{x}_i, \boldsymbol{\omega}_1])\boldsymbol{\phi}_{10}^T\mathbf{x}_i + g[\mathbf{x}_i, \boldsymbol{\omega}_1]\boldsymbol{\phi}_{11}^T\mathbf{x}_i\right]. \end{aligned} \tag{9.56}$$

This is an example of a *classification tree*.

To learn the parameters $\boldsymbol{\theta} = \{\boldsymbol{\phi}_0, \boldsymbol{\phi}_1, \boldsymbol{\phi}_{00}, \boldsymbol{\phi}_{01}, \boldsymbol{\phi}_{10}, \boldsymbol{\phi}_{11}, \boldsymbol{\omega}, \boldsymbol{\omega}_0, \boldsymbol{\omega}_1\}$ we could take an incremental approach. At the first stage we fit the top part of the tree (equation 9.54), setting parameters $\boldsymbol{\omega}, \boldsymbol{\phi}_0, \boldsymbol{\phi}_1$. Then we fit the left branch, setting parameters $\boldsymbol{\omega}_0, \boldsymbol{\phi}_{00}, \boldsymbol{\phi}_{01}$ and subsequently the right branch, setting parameters $\boldsymbol{\omega}_1, \boldsymbol{\phi}_{10}, \boldsymbol{\phi}_{11}$ and so on.

The classification tree has the potential advantage of speed. If each gating function produces a binary output (like the heaviside step function), then each data point passes down just one of the outgoing edges from each node and ends up at a single leaf. When each branch in the tree is a linear operation (as in this example), these operations can be aggregated to a single linear operation at each leaf. Since each data point receives specialized processing, the tree need not usually be deep, and new data can be classified very efficiently.

9.9 Multi-class logistic regression

Throughout this chapter, we have discussed binary classification. We now discuss how to extend these models to handle $N > 2$ world states. One possibility is to build N *one-against-all* binary classifiers each of which computes the probability that the n^{th} class is present as opposed to any of the other classes. The final label is assigned according to the one-against-all classifier with the highest probability. Algorithm 9.8

The one-against-all approach works in practice but is not very elegant. A more principled way to cope with multi-class classification problems is to describe the the posterior $Pr(w|\mathbf{x})$ as a categorical distribution, where the parameters $\boldsymbol{\lambda} = [\lambda_1 \ldots \lambda_N]$ are functions of the data $\mathbf{x}$

$$Pr(w|\mathbf{x}) = \text{Cat}_w[\boldsymbol{\lambda}[\mathbf{x}]], \tag{9.57}$$

where the parameters are in the range $\lambda_n \in [0, 1]$ and sum to one, $\sum_n \lambda_n = 1$. In constructing the function $\boldsymbol{\lambda}[\mathbf{x}]$ we must ensure that we obey these constraints.

As for the two class logistic regression case, we will base the model on linear functions of the data $\mathbf{x}$ and pass these through a function that enforces the constraints. To this end, we define N activations (one for each class),

$$a_n = \boldsymbol{\phi}_n^T \mathbf{x}, \tag{9.58}$$

where $\boldsymbol{\phi}_1 \ldots \boldsymbol{\phi}_N$ are parameter vectors. We assume that as usual we have prepended a 1 to each of the data vectors $\mathbf{x}_i$ so that the first entry of each parameter vectors $\boldsymbol{\phi}_n$ represents an offset. The n^{th} entry of the final categorical distribution is now defined by

$$\lambda_n = \text{softmax}_n[a_1, a_2 \ldots a_N] = \frac{\exp[a_n]}{\sum_{m=1}^{N} \exp[a_m]}. \tag{9.59}$$

The function $\textbf{softmax}[\bullet]$ takes the N activations $a_1 \ldots a_N$, which can take any real number, and maps them to the N parameters $\lambda_1 \ldots \lambda_N$ of the categorical distribution, which are constrained to be positive and sum to one (figure 9.19). Problem 9.10

To learn the parameters $\boldsymbol{\theta} = \{\boldsymbol{\phi}_1 \ldots \boldsymbol{\phi}_N\}$ given training pairs $(w_i, \mathbf{x}_i)$ we optimize the log likelihood of the training data

$$L = \sum_{i=1}^{I} \log \left[Pr(w_i|\mathbf{x}_i) \right]. \tag{9.60}$$

As for the two class case, there is no closed form expression for the maximum likelihood parameters. However, this is a convex function, and the maximum can be found using a nonlinear optimization technique such as the Newton method. These techniques require the first and second derivatives of the log likelihood with respect to the parameters, which are given by: Problem 9.11

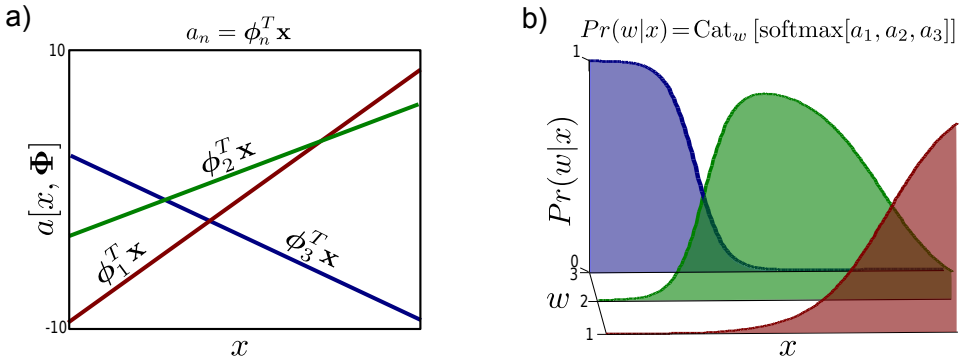


Figure 9.19 Multi-class logistic regression. a) We form one activation for each class based on linear functions of the data. b) We pass these activations through the softmax function to create the distribution $Pr(w|x)$ which is shown here as a function of x. The softmax function takes the three real-valued activations and returns three positive values that sum to one, ensuring that the distribution $Pr(w|x)$ is a valid probability distribution for all x.

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{\phi}_n} &= -\sum_{i=1}^{I}(y_{in} - \delta[w_i - n])\mathbf{x}_i \\ \frac{\partial^2 L}{\partial \boldsymbol{\phi}_m \boldsymbol{\phi}_n} &= -\sum_{i=1}^{I} y_{im}(\delta[m-n] - y_{in})\mathbf{x}_i\mathbf{x}_i^T,\end{aligned} \tag{9.61}$$

where we define the term

$$y_{in} = Pr(w_i = n|\mathbf{x}_i) = \mathrm{softmax}_n[a_{i1}, a_{i2} \ldots a_{iN}]. \tag{9.62}$$

It is possible to extend multi-class logistic regression in all of the ways that we extended the two class model. We can construct Bayesian, nonlinear, dual and kernelized versions. It is possible to train incrementally and combine weak classifiers in a boosting framework. Here, we will consider tree-structured models as these are very common in modern vision applications.

9.10 Random trees, forests, and ferns

Algorithm 9.9

In section 9.8 we introduced the idea of tree-structured classifiers, in which the processing for each data example is different and becomes steadily more specialized. This idea has recently become extremely popular for multi-class problems in the form of *random classification trees*.

As for the two-class case, the key idea is to construct a binary tree where at each node, the data are evaluated to determine whether it will pass to the left or the

right branch. Unlike in section 9.8, we will assume that each data point passes into just one branch. In a random classification tree, the data are evaluated against a function $q[\mathbf{x}]$ that was randomly chosen from a predefined family of possible functions. For example, this might be the response of a randomly chosen filter. The data proceeds one way in the tree if the response of this function exceeds a threshold τ and the other way if not. While the functions are chosen randomly, the threshold is carefully selected.

We select the threshold that maximizes the log-likelihood L of the data:

$$\begin{aligned} L &= \sum_{i=1}^{I}(1-\text{heaviside}[q[\mathbf{x}_i]-\tau])\log\left[\text{Cat}_{w_i}\left[\boldsymbol{\lambda}^{[l]}\right]\right] \\ &\qquad +\text{heaviside}[q[\mathbf{x}_i]-\tau]\log\left[\text{Cat}_{w_i}\left[\boldsymbol{\lambda}^{[r]}\right]\right]. \end{aligned} \tag{9.63}$$

Here the first term represents the contribution of the data that passes down the left branch, and the second term represents the contribution of the data that passes down the right branch. In each case, the data are evaluated against a categorical distribution with parameters $\boldsymbol{\lambda}^{[l]}$ and $\boldsymbol{\lambda}^{[r]}$, respectively. These parameters are set using maximum likelihood:

$$\begin{aligned} \lambda_k^{[l]} &= \frac{\sum_{i=1}^{I}\delta[w_i-k](1-\text{heaviside}[q[\mathbf{x}_i]-\tau])}{\sum_{i=1}^{I}(1-\text{heaviside}[q[\mathbf{x}_i]-\tau])} \\ \lambda_k^{[r]} &= \frac{\sum_{i=1}^{I}\delta[w_i-k](\text{heaviside}[q[\mathbf{x}_i]-\tau])}{\sum_{i=1}^{I}(\text{heaviside}[q[\mathbf{x}_i]-\tau])}. \end{aligned} \tag{9.64}$$

The log likelihood is not a smooth function of the threshold τ, and so in practice we maximize the log likelihood by empirically trying a number of different threshold values and choosing the one that gives the best result.

We then perform this same procedure recursively; the data that pass to the left branch has a new randomly chosen classifier applied to them and a new threshold is chosen that splits it again. This can be done without recourse to the data in the right branch. When we classify a new data example $\mathbf{x}^*$, we pass it down the tree until it reaches one of the leaves. The posterior distribution $Pr(w^*|\mathbf{x}^*)$ over the world state w^* is set to $\text{Cat}_{w^*}[\boldsymbol{\lambda}]$ where the parameters $\boldsymbol{\lambda}$ are the categorical parameters associated with this leaf during the training process.

The random classification tree is attractive because it is very fast to train – after all, most of its parameters are chosen randomly. It can also be trained with very large amounts of data as its complexity is linear in the number of data examples.

There are two important variations on this model:

1. A *fern* is a tree where the randomly chosen functions at each level of the tree are constrained to be the same. In other words, the data that pass through the left and right branches at any node are subsequently acted on by the same function (although the threshold level may optionally be different in each branch). In practice, this means that every data point is acted on by the same sequence of functions. This can make implementation extremely efficient when we are evaluating the classifier repeatedly.

2. A *random forest* is a collection of random trees, each of which uses a different randomly chosen set of functions. By averaging together the probabilities $Pr(w^*|\mathbf{x}^*)$ predicted by these trees, a more robust classifier is produced. One way to think of this is as approximating the Bayesian approach; we are constructing the final answer by taking a weighted sum of the predictions suggested by different sets of parameters.

9.11 Relation to non-probabilistic models

In this chapter, we have described a family of probabilistic algorithms for classification. Each is based on maximizing either the log Bernoulli probability of the training class labels given the data (two-class case) or the log categorical probability of the training class labels given the data (multi-class case).

However, it is more common in the computer vision literature to use non-probabilistic classification algorithms such as the multilayer perceptron, adaboost or support vector classification. At their core, these algorithms optimize different objective functions and so are neither directly equivalent to each other, nor to the models in this chapter.

We chose to describe the less common probabilistic algorithms because

- they have no serious disadvantages relative to non-probabilistic techniques,
- they naturally produce estimates of certainty,
- they are easily extensible to the multi-class case whereas non-probabilistic algorithms usually rely on one-against-all formulations, and
- they are more easily related to one another and to the rest of the book.

In short, it can reasonably be argued that the dominance of non-probabilistic approaches to classification is largely for historical reasons. We will now briefly describe the relationship between our models and common non-probabilistic approaches.

The *multi-layer perceptron* or *neural network* is very similar to our nonlinear logistic regression model in the special case where the nonlinear transform consists of a set of sigmoid functions applied to linear projections of data (e.g., $z_k = \arctan[\boldsymbol{\alpha}_k^T\mathbf{x}]$). In the MLP, learning is known as *back propagation* and the transformed variable $\mathbf{z}$ is known as the *hidden layer*.

Adaboost is very closely related to the the logitboost model described in this chapter, but adaboost is not probabilistic. Performance of the two algorithms is similar.

The *support vector machine* (SVM) is similar to relevance vector classification; it is a kernelized classifier that depends sparsely on the data. It has the advantage that its objective function is convex, whereas the objective function in relevance vector classification is non-convex and only guarantees to converge to a local minimum. However, the SVM has several disadvantages: it does not assign certainty to its class predictions, it is not so easily extended to the multi-class case, it produces

solutions that are less sparse than relevance vector classification, and it places more restrictions on the form of the kernel function. In practice, classification performance of the two models is again similar.

9.12 Applications

We now present a number of examples of the use of classification in computer vision from the research literature. In many of the examples, the method used was non-probabilistic (e.g., adaboost), but is very closely related to the algorithms in this chapter, and one would not expect the performance to differ significantly if these were substituted.

9.12.1 Gender classification

The algorithms in this chapter were motivated by the problem of gender detection in unconstrained facial images. The goal is to assign a label $w \in \{0, 1\}$ indicating whether a small patch of an image $\mathbf{x}$ contains a male or a female face. Prince & Aghajanian (2009) developed a system of this type. First, a bounding box around the face was identified using a face detector (see next section). The data within this bounding box was resized to 60×60, converted to grayscale and histogram equalized. The resulting image was convolved with a bank of Gabor functions and the filtered images sampled at regular intervals that were proportionate to the wavelength to create a final feature vector of length 1064. Each dimension was whitened to have mean zero and unit standard deviation. Chapter 13 contains information about these and other preprocessing methods.

A training database of 32000 examples was used to learn a nonlinear logistic regression model of the form

$$Pr(w_i|\mathbf{x}_i) = \text{Bern}_{w_i}\left[\frac{1}{1+\exp\left[-\phi_0 - \sum_{k=1}^{K}\phi_k \text{f}[\mathbf{x}_i, \boldsymbol{\xi}_k]\right]}\right], \tag{9.65}$$

where the nonlinear functions $\text{f}[\bullet]$ were arc tangents of linear projections of the data so that

$$\text{f}[\mathbf{x}_i, \boldsymbol{\xi}_k] = \arctan[\boldsymbol{\xi}_k^T \mathbf{x}_i]. \tag{9.66}$$

As usual the data were augmented by prepending a 1 so the projection vectors $\{\boldsymbol{\xi}_k\}$ were of length $D+1$. This model was learned using an incremental approach so that at each stage the parameters $\boldsymbol{\phi}_0, \phi_k$ and $\boldsymbol{\xi}_k$ were modified.

The system achieved 87.5% performance with $K = 300$ arc tangent functions on a challenging real-world database that contained large variations in scale, pose, lighting and expression similar to the faces in figure 9.1. Human observers managed only 95% performance on the same database using the resized face region alone.

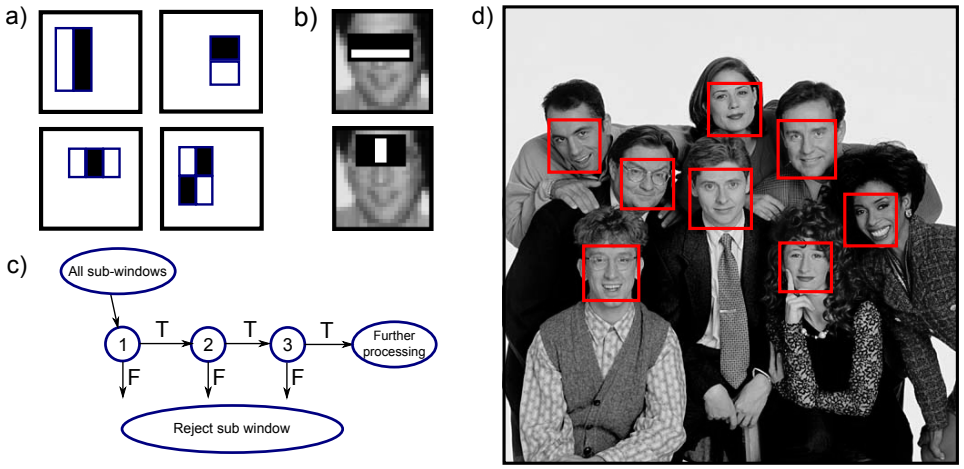


Figure 9.20 Fast face detection using a boosting method (Viola & Jones 2004). a) Each weak classifier consists of the response of the image to a Haar-like filter, which is then passed through a step function. b) The first two weak classifiers learned in this implementation have clear interpretations: the first responds to the dark horizontal region belonging to the eyes and the second responds to the relative brightness of the bridge of the nose. c) The data passes through a cascade: most regions can be quickly rejected after evaluating only a few weak classifiers as they look nothing like faces. More ambiguous regions undergo further preprocessing. d) Example results. Adapted from Viola & Jones (2004).

9.12.2 Face and pedestrian detection

Before we can determine the gender of a face, we must first find it. In face detection (figure 7.1), we assign a label $w \in \{0,1\}$ to a small region of the image $\mathbf{x}$ indicating whether a face is present ($w=1$) or not ($w=0$). To ensure that the face is found, this process is repeated at every position and scale in the image and consequently the classifier must be very fast.

Viola & Jones (2004) presented a face detection system based on *adaboost* (figure 9.20). This is a non-probabilistic analogue of the boosting methods described in section 9.7.1. The final classification is based on the sign of a sum of nonlinear functions of the data

$$a = \phi_0 + \sum_{k=1}^{K} \phi_k \mathrm{f}[\mathbf{x}, \boldsymbol{\xi}_k], \tag{9.67}$$

where the nonlinear functions $f[\bullet]$ are heaviside step functions of projections of the data (weak classifiers giving a response of zero or one for each possible data vector $\mathbf{x}$) so that

$$\mathrm{f}[\mathbf{x}, \boldsymbol{\xi}_k] = \mathrm{heaviside}[\boldsymbol{\xi}_k^T \mathbf{x}]. \tag{9.68}$$

As usual, the data vector $\mathbf{x}$ was prepended with a 1 to account for an offset.

The system was trained on 5,000 faces and 10,000 non-face regions, each of which was represented as a 24×24 image patch. Since the model is not smooth (due to the step function), gradient-based optimization is unsuitable, and so Viola & Jones (2004) exhaustively searched through a very large number of pre-defined projections $\boldsymbol{\xi}_k$.

There were two aspects of the design that ensured that the system ran quickly.

1. The structure of the classifier was exploited: training in boosting is incremental – the 'weak classifiers' (nonlinear functions of the data) are incrementally added to create an increasingly sophisticated strong classifier. Viola & Jones (2004) exploited this structure when they ran the classifier: they reject regions that are very unlikely to be faces based on responses of the first few weak classifiers and only subject more ambiguous regions to further processing. This is known as a *cascade* structure. During training, the later stages of the cascade are trained with new negative examples that were not rejected by the earlier stages.

2. The projections $\boldsymbol{\xi}_k$ were carefully chosen so that they were very fast to evaluate: they consisted of Haar-like filters (section 11.4.4) which require only a few operations to compute.

The final system consisted of 4297 weak classifiers divided into a 32 stage cascade. It found 91.1% of 507 frontal faces across 130 images, with a false positive rate of less than 1 per frame, and processed images in fractions of a second.

Viola *et al.* (2005) developed a similar system for detecting pedestrians in video sequences (figure 9.21). The main modification was to extend the set of weak classifiers to encompass features that span more than one frame and hence select for the particular temporal patterns associated with human motion. To this end their system used not only the image data itself, but also the difference image between adjacent frames and similar difference images when taken after offsetting the frames in each of four directions. The final system achieved an 80% detection rate with a false alarm rate of 1/400,000 which corresponds to one false positive for every two frames.

9.12.3 Semantic segmentation

The goal of semantic segmentation is to assign a label $w \in \{1 \ldots M\}$ to each pixel indicating which of M objects is present, based on the local image data $\mathbf{x}$. Shotton *et al.* (2009) developed a system known as *textonboost* that was based on a non-probabilistic boosting algorithm called *jointboost* (Torralba *et al.* 2007). The decision was based on a one-against-all strategy in which M binary classifiers are computed based on the weighted sums

$$a_m = \phi_{0m} + \sum_{k=1}^{K} \phi_{km} \mathrm{f}[\mathbf{x}, \boldsymbol{\xi}_k], \tag{9.69}$$

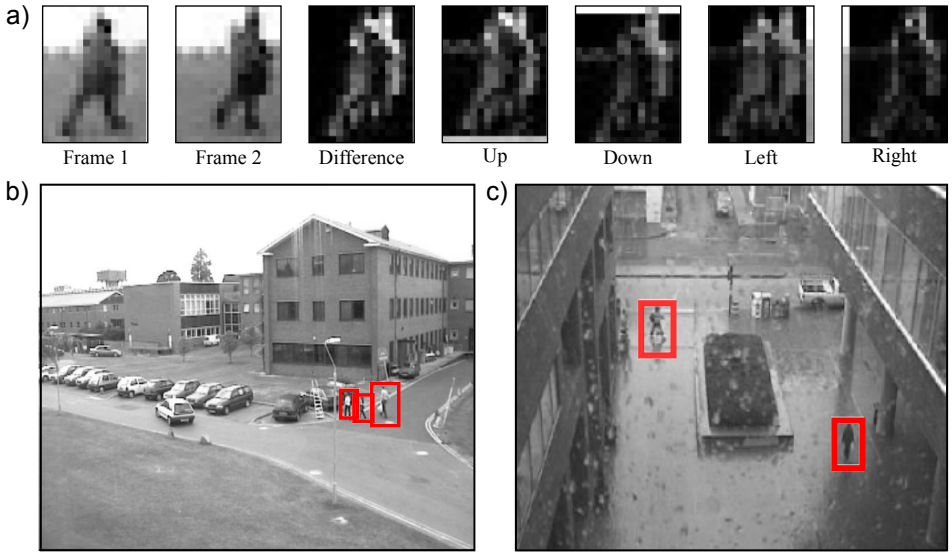


Figure 9.21 Boosting methods based on the thresholded responses of Haar functions have also been used for pedestrian detection in video footage. a) To improve detection rates two subsequent frames are used. The absolute difference between the frames is computed as is the difference when one of the frames is offset in each of four directions. The set of potential weak classifiers consists of Haar functions applied to all six of these representations. b,c) Example results. Adapted from Viola *et al.* (2005). ©2005 Springer.

where the nonlinear functions f[•] were once more based on heaviside step functions. Note that the weighted sums associated with each object class share the same nonlinear functions, but weight them differently. After computing these series, the decision is assigned based on the activation a_m that is the greatest.

Shotton *et al.* (2009) based the nonlinear functions on a *texton* representation of the image: each pixel in the image is replaced by a discrete index indicating the 'type' of texture present at that position (see section 13.1.5). Each nonlinear function considers one of these texton types and computes the number of times that it is found within a rectangular area. This area has a fixed spatial displacement from the pixel under consideration (figure 9.22c-f). If this displacement is zero, then the function provides evidence about the pixel directly (e.g., it looks like grass). If the spatial displacement is larger, then the function provides evidence of the local context (e.g., there is grass nearby, so this pixel may belong to a cow).

For each nonlinear function, an offset is added to the texton count and the result is passed through a step function. The system was learned incrementally by assessing each of a set of randomly chosen classifiers (defined by the choice of texton, rectangular region and offset) and choosing the best at the current stage.

The full system also included a post-processing step in which the result was refined using a conditional random field model (see chapter 12). It achieved 72.2%

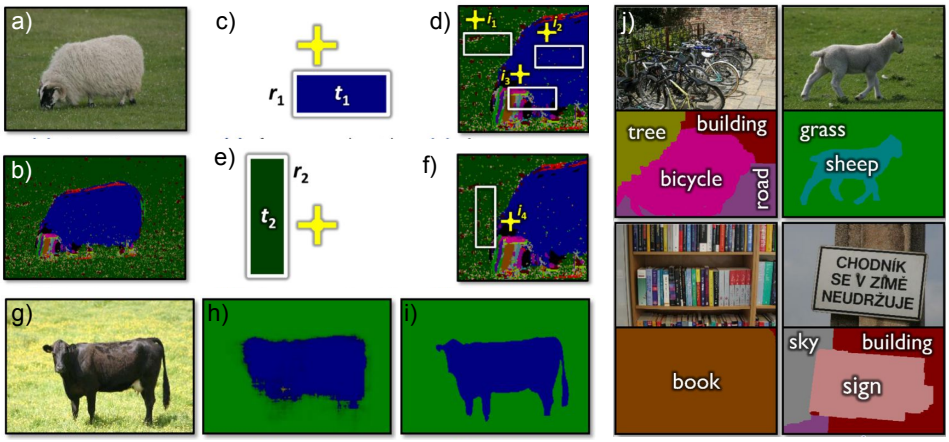


Figure 9.22 Semantic image labeling using "TextonBoost". a) Original image. b) Image converted to textons – a discrete value at each pixel indicating the type of texture that is present. c) The system was based on weak classifiers that count the number of textons of a certain type within a rectangle that is offset from the current position (yellow cross). d) This provides both information about the object itself (contains sheep-like textons) and nearby objects (near to grass-like textons). e,f) Another example of a weak classifier. g) Test image. h) Per-pixel classification is not very precise at the edges of objects and so i) a conditional random field is used to improve the result. j) Examples of results and ground truth. Adapted from Shotton *et al.* (2009) ©2009 Springer.

performance on the challenging MRSC database that includes 21 diverse object classes including wiry objects such as bicycles and objects with a large degree of variation such as dogs.

9.12.4 Recovering surface layout

To recover the *surface layout* of a scene we assign a label $w \in \{1, \ldots 3\}$ to each pixel in the image indicating whether the pixel contains a support object (e.g., floor), a vertical object (e.g., building), or the sky. This decision is based on local image data $\mathbf{x}$. Hoiem *et al.* (2007) constructed a system of this type using a one-against-all principle. Each of the three binary classifiers was based on logitboosted classification trees; different classification trees are treated as weak classifiers and the results are weighted together to compute the final probability.

Hoiem *et al.* (2007) worked with the intermediate representation of *superpixels* – an over-segmentation of the scene into small homogeneous regions, which are assumed to belong to the same object. Each superpixel was assigned a label w using the classifier based on a data vector $\mathbf{x}$, which contained location, appearance, texture and perspective information associated with the superpixel.

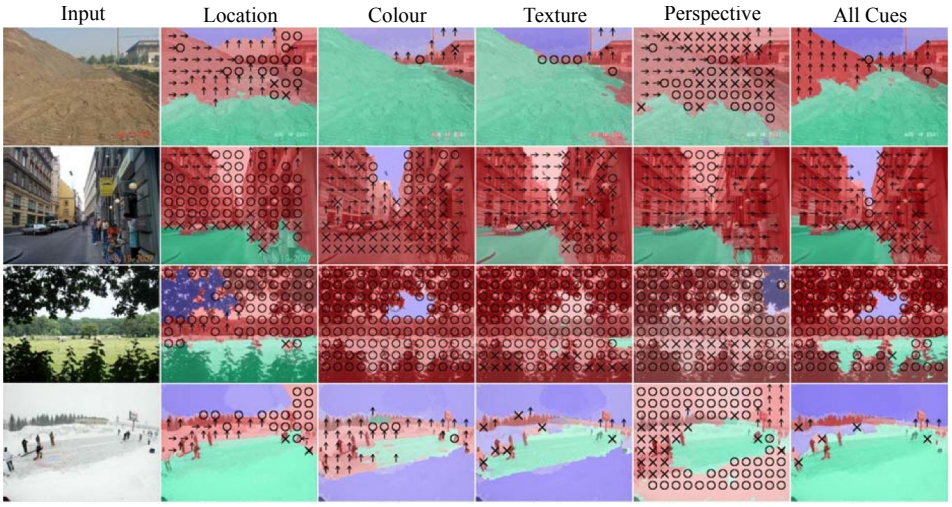


Figure 9.23 Recovering surface layout. The goal is to take an image and return a label indicating whether the pixel is part of a support surface (green pixels) vertical surface (red pixels) or the sky (blue pixels). Vertical surfaces were sub-classified into planar objects at different orientations (left arrows, upward arrows and right arrows denote left-facing, fronto-parrallel and right-facing surfaces) and non-planar objects which can be porous (marked as 'o') or non-porous (marked as 'x'. The final classification was based on (i) location cues (which include position in the image and position relative to the horizon), (ii) color cues, (iii) texture cues, and (iv) perspective cues, which were based on the statistics of line segments in the region. The figure shows example classifications for each of these cues alone and when combined. Adapted from Hoiem *et al.* (2007). ©2007 Springer.

To mitigate against the possibility that the original superpixel segmentation was wrong, multiple segmentations were computed and the results merged to provide a final per-pixel classification (figure 9.23). In the full system, regions that were classified as vertical were sub-classified into left-facing planar surfaces, fronto-parallel planar surfaces, or right-facing planar surfaces, or non-planar surfaces, which may be porous (e.g., trees) or solid. The system was trained and tested on a data set consisting of images collected from the web including diverse environments (forests, cities, roads etc.) and conditions (snowy, sunny, cloudy etc.). The data set was pruned to remove photos where the horizon was not within the image.

The system correctly labeled 88.1% of pixels correctly with respect to the main three classes and 61.5% correctly with respect to the subclasses of the vertical surface. This algorithm was the basis of a remarkable system for creating a 3D model from a single 2D photograph (Hoiem *et al.* 2005).

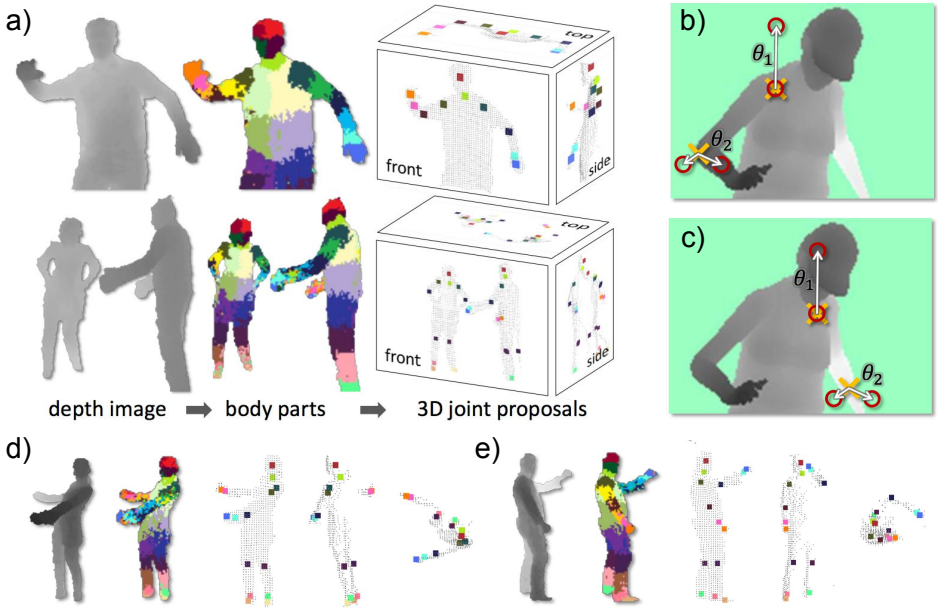


Figure 9.24 Identifying human parts. a) The goal of the system is to take a depth image $\mathbf{x}$ and assign a discrete label w to each pixel indicating which of 31 possible body parts is present. These depth labels are used to form proposals about the position of 3D joints. b) The classification is based on decision trees. At each point in the tree, the data are divided according to the relative depth at two points (red circles) offset relative to the current pixel (yellow crosses). In this example, this difference is large in both cases, whereas in c) this difference is small - hence these differences provide information about the pose. d,e) Two more examples of depth image, labeling and hypothesized pose. Adapted from Shotton *et al.* (2011) ©2011 IEEE

.

9.12.5 Identifying human parts

Shotton *et al.* (2011) describe a system that assigns a discrete label $w \in \{1, \ldots, 31\}$, indicating which of 31 body parts is present at each pixel based on a depth image $\mathbf{x}$. The resulting distribution of labels is an intermediate representation in a system that proposes a possible configuration of the 3D joint positions in the Microsoft Kinect gaming system (figure 9.24).

The classification was based on a *forest* of decision trees: the final probability $Pr(w|\mathbf{x})$ is an average (i.e., a mixture) of the predictions from a number of different classification trees. The goal is to mitigate against biases introduced by the greedy method with which a single tree is trained.

Within each tree, the decision about which branch a data point travels down is based on the difference in measured depths at two points, each of which is spatially offset from the current pixel. The offsets are inversely scaled by the distance to

the pixel itself, which ensures that they address the same relative positions on the body when the person moves closer or further away to the depth camera.

The system was trained from a very large data set of 900,000 depth images which were synthesized based on motion capture data and consisted of three trees of depth 20. Remarkably, the system is capable of assigning the correct label 59%, of the time and this provides a very solid basis for the subsequent joint proposals.

Discussion

In this chapter we have considered classification problems. We note that all of the ideas that were applied to regression models in chapter 8 are also applicable to classification problems. However, for classification the model includes a nonlinear mapping between the data $\mathbf{x}$ and the parameters of the distribution $Pr(w|\mathbf{x})$ over the world w. This means that we cannot find the maximum likelihood solution in closed form (although the problem is still convex) and we cannot compute a full Bayesian solution without making approximations.

Classification techniques have many uses in machine vision. Notice though that these models have no domain specific information about the problem other than that provided by the preprocessing of the data. This is both an advantage (they find many applications) and a disadvantage (they cannot take advantage of a priori information about the problem). In the remaining part of the book we will explore models that introduce increasing amounts of domain specific information to the problem.

Notes

Classification in vision: Classification techniques such as those discussed in this chapter have been applied to many problems in vision including face detection (Viola & Jones 2004), surface layout estimation (Hoiem *et al.* 2007), boundary detection (Dollár *et al.* 2006), keypoint matching (Lepetit *et al.* 2005), body part classification (Hoiem *et al.* 2007), semantic segmentation (He *et al.* 2004), object recognition (Csurka *et al.* 2004), and gender classification (Kumar *et al.* 2008).

Probabilistic classification: More information about logistic regression can be found in Bishop (2006) and many other statistics textbooks. Kernel logistic regression (or Gaussian process regression) was presented in Williams & Barber (1998), and more information can be found in Rasmussen & Williams (2006). A sparse version of kernel logistic regression (relevance vector classification) was presented by Tipping (2001) and a sparse multi-class variant was developed by Brishnapuram *et al.* (2005). Probabilistic interpretations of boosting were introduced by Friedman *et al.* (2000). Random forests of multinomial regressors were introduced in Prinzie & Van den Poel (2008).

Other classification schemes: In this chapter, we have presented a family of probabilistic classification models based on logistic regression. There are other non-probabilistic techniques for classification, and these include single and multi-layer perceptrons (Rosenblatt 1958; Rumelhart *et al.* 1986), support vector machines (Vapnik 1995; Cristianini & Shawe-Taylor 2000), and adaboost (Freund & Schapire 1995). A critical difference between these techniques is the underlying objective function. Logistic regression models optimize the log Bernoulli probability, but the other models optimize different criteria, such as the hinge loss (support vector machines) or exponential error (adaboost). It is difficult to make general statements about the relative merits of these approaches, but it is probably fair to say that (i) there is no major disadvantage to using the probabilistic techniques in this chapter and (ii) the choice of classification method is usually less important in vision problems than the preprocessing of the data. Methods based on boosting and classification trees are particularly popular in vision because of their speed.

Boosting: Adaboost was introduced by Freund & Schapire (1995). Since then there have been a large number of variations, most of which have been used in computer vision. These include discrete adaboost (Freund & Schapire 1996), real adaboost (Schapire & Singer 1998), gentleboost (Friedman *et al.* 2000), logitboost (Friedman *et al.* 2000), floatboost (Li *et al.* 2003), KLBoost (Liu & Shum 2003), asymmetric boost (Viola & Jones 2002), and statboost (Pham & Chan 2007a). Boosting has also been applied to the multi-class case (Schapire & Singer 1998; Torralba *et al.* 2007) and for regression (Friedman 1999). A review of boosting approaches can be found in Meir & Mätsch (2003).

Classification trees: Classification trees have a long history in computer vision, dating back to at least Shepherd (1983). Modern interest was stimulated by Amit & Geman (1997) and Breiman (2001) who investigated the use of random forests. Since this time classification trees and forests have been applied to keypoint matching (Lepetit *et al.* 2005), segmentation (Yin *et al.* 2007), human pose detection (Rogez *et al.* 2006; Shotton *et al.* 2011), object detection (Bosch *et al.* 2007), image classification (Moosmann *et al.* 2006; Moosmann *et al.* 2008), deciding image suitability (Mac Aodha *et al.* 2010), detecting occlusions (Humayun *et al.* 2011), and semantic image segmentation (Shotton *et al.* 2009).

Gender classification: Automatic determination of gender from a facial image has variously been tackled with neural networks (Golomb *et al.* 1990), support vector machines (Moghaddam & Yang 2002), linear discriminant analysis (Bekios-Calfa *et al.* 2011) and

both adaboost (Baluja & Rowley 2003), and logitboost (Prince & Aghajanian 2009). A review is provided by Mäkinen & Raisamo (2008b) and quantative comparisons are presented in Mäkinen & Raisamo (2008a). Representative examples of the state of the art can be found in Kumar *et al.* (2008) and Shan (in press).

Face detection: The application of boosting to face detection (Viola & Jones 2004) usurped earlier techniques (e.g., Osuna *et al.* 1997; Schneiderman & Kanade 2000). Since then, many boosting variants have been applied to the problem including floatboost (Li *et al.* 2002; Li & Zhang 2004), gentleboost (Lienhart *et al.* 2003), realboost (Huang *et al.* 2007a; Wu *et al.* 2007), asymboost (Pham & Chan 2007b; Viola & Jones 2002) and statboost (Pham & Chan 2007a). A recent review of this area can be found in Zhang & Zhang (2010).

Semantic segmentation: The authors of the system described in the text (Shotton *et al.* 2008b) subsequently presented a much faster system based on classification trees (Shotton *et al.* 2009). A recent comparison of quantitative performance can be found in Ranganathan (2009). Other work has investigated the imposition of prior knowledge such as the co-presence of object classes (He *et al.* 2006) and likely spatial configurations of objects (He *et al.* 2004).

Problems

Problem 9.1 The logistic sigmoid function is defined as

$$\text{sig}[a] = \frac{1}{1+\exp[-a]}.$$

Show that (i) $\text{sig}[-\infty] = 0$, (ii) $\text{sig}[0] = 0.5$, (iii) $\text{sig}[\infty] = 1$.

Problem 9.2 Show that the derivative of the log posterior probability for the logistic regression model

$$L = \sum_{i=1}^{I} w_i \log\left[\frac{1}{1+\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]}\right] + \sum_{i=1}^{I}(1-w_i)\log\left[\frac{\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]}{1+\exp[-\boldsymbol{\phi}^T\mathbf{x}_i]}\right]$$

with respect to the parameters $\boldsymbol{\phi}$ is given by

$$\frac{\partial L}{\partial \boldsymbol{\phi}} = -\sum_{i=1}^{I}(\text{sig}[a_i] - w_i)\,\mathbf{x}_i.$$

Problem 9.3 Show that the second derivative of the log likelihood of the logistic regression model is given by

$$\frac{\partial^2 L}{\partial \boldsymbol{\phi}^2} = -\sum_{i=1}^{I}\text{sig}[a_i](1-\text{sig}[a_i])\mathbf{x}_i\mathbf{x}_i^T.$$

Problem 9.4 Consider fitting a logistic regression model to 1D data x where the two classes are perfectly separable. For example, perhaps all the data x where the world state $w = 0$ takes values less than 0 and all the data x where the world state is $w = 1$ takes

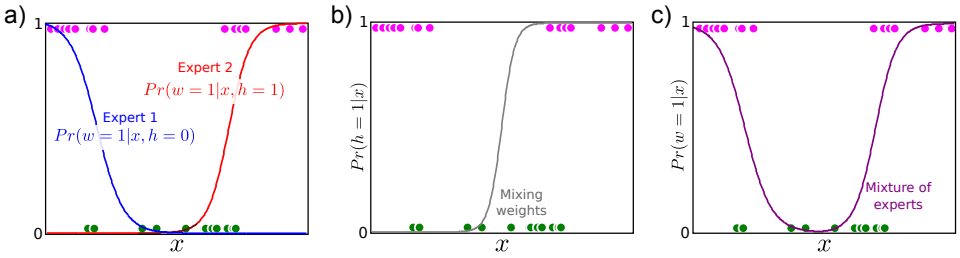


Figure 9.25 Mixture of two experts model for 1D data. Pink circles indicate positive examples. Green circles indicate negative examples. a) Two expert is specialized to model the left and right sides of the data, respectively. b) The mixing weights change as a function of the data. c) The final output of the model is mixture of the two constituent experts and fits the data well.

values greater than 1. Hence it is possible to classify the training data perfectly. What will happen to the parameters of the model during learning? How could you rectify this problem?

Problem 9.5 Compute the Laplace approximation to a beta distribution with parameters $\alpha = 1.0$, $\beta = 1.0$.

Problem 9.6 Show that the Laplace approximation to a univariate normal distribution with mean μ and variance σ^2 is the normal distribution itself.

Problem 9.7 Show that the second derivative of the logarithm $L = \log[\text{Norm}_{\mathbf{x}}[\boldsymbol{\mu}, \boldsymbol{\Sigma}]]$ of a normal distribution evaluated at the mean $\boldsymbol{\mu}$ is given by

$$\left|\frac{\partial L}{\partial \phi^2}\right|_{\boldsymbol{\mu}} = -\boldsymbol{\Sigma}^{-1}.$$

Problem 9.8 Devise a method to choose the scale parameter λ_0 in the radial basis function in kernel logistic regression (equation 9.33).

Problem 9.9 A *mixture of experts* (Jordan & Jacobs 1994) divides space into different regions, each of which receives specialized attention (figure 9.25) . For example, we could describe the data as a mixture of logistic classifiers so that

$$Pr(w_i|\mathbf{x}_i) = \sum_{k=1}^{K} \lambda_k[\mathbf{x}_i]\text{Bern}_{w_i}\left[\text{sig}[\boldsymbol{\phi}_k^T\mathbf{x}_i]\right].$$

Each logistic classifier is considered as an expert and the mixing weights decide the combination of experts that are applied to the data. The mixing weights, which are positive and sum to one, depend on the data $\mathbf{x}$: for a two-component model they could be based on a second logistic regression model with activation $\boldsymbol{\omega}^T\mathbf{x}$. This model can be expressed as the marginalization of a joint distribution between $\mathbf{w}_i$ and a hidden variable h_i so that

$$Pr(w_i|\mathbf{x}_i) = \sum_{k=1}^{K} Pr(w_i, h_i = k|\mathbf{x}_i) = \sum_{k=1}^{K} Pr(w_i|h_i = k, \mathbf{x}_i)Pr(h_i = k|\mathbf{x}_i),$$

where

$$\begin{aligned} Pr(w_i|h_i = k, \mathbf{x}_i) &= \text{Bern}_{w_i}\left[\text{sig}[\boldsymbol{\phi}_k^T \mathbf{x}_i]\right] \\ Pr(h_i = k|\mathbf{x}_i) &= \text{Bern}_{h_i}\left[\text{sig}[\boldsymbol{\omega}^T \mathbf{x}_i]\right]. \end{aligned}$$

How does this model differ from branching logistic regression (section 9.8)? Devise a learning algorithm for this model.

Problem 9.10 The $\textbf{softmax}[\bullet, \bullet, \ldots, \bullet]$ function is defined to return a multivariate quantity where the k^{th} element is given by

$$s_k = \text{softmax}_k[a_1, a_2, \ldots a_K] = \frac{\exp[a_k]}{\sum_{j=1}^K \exp[a_j]}.$$

Show that $0 < s_k < 1$ and that $\sum_{k=1}^K s_k = 1$.

Problem 9.11 Show that the first derivative of the log-probability of the multi-class logistic regression model is given by equation 9.61.

Problem 9.12 The classifiers in this chapter have all been based on continuous data $\mathbf{x}$. Devise a model that can distinguish between M world states $w \in \{1 \ldots M\}$ based on a discrete observation $x \in \{1 \ldots K\}$ and discuss potential learning algorithms.

Part III

Connecting local models

Part III: Connecting local models

The models in chapters 6-9 describe the relationship between a set of measurements and the world state. They work well when the measurements and the world state are both low dimensional. However, there are many situations where this is not the case, and these models are unsuitable.

For example, consider the semantic image labeling problem in which we wish to assign a label to each pixel in the image where the label denotes the object class. For example, in a road scene we might wish to label pixels as 'road', 'sky', 'car', 'tree', 'building' or 'other'. For an image with $N = 10000$ pixels, this means we need to build a model relating the 10000 measured RGB triples to 6^{10000} possible world states. None of the models discussed so far can cope with this challenge: the number of parameters involved (and hence the amount of training data and the computational requirements of the learning and inference algorithms) is far beyond what current machines can handle.

One possible solution to this problem would be to build a set of independent local models: for example, we could build models that relate each pixel label separately to the nearby RGB data. However, this is not ideal as the image may be locally ambiguous. For example, a small blue image patch might result from a variety of semantically different classes: sky, water, a car door or a person's clothing. In general, it is insufficient to build independent local models.

The solution to this problem is to build local models that are *connected* to one another. Consider again the semantic labeling example: given the whole image, we can see that when the image patch is blue and is found above trees and mountains and alongside similar patches across the top of the image, then the correct class is probably sky. Hence, to solve this problem, we still model the relationship between the label and its local image region, but we also connect these models so that nearby elements can help to disambiguate one another.

In chapter 10 we introduce the idea of conditional independence, which is a way of characterizing redundancies in the model (i.e., the lack of direct dependence between variables). We show how conditional independence relations can be visualized with graphical models. We distinguish between directed and undirected graphical models. In chapter 11, we discuss models in which the local units are combined together to form chains or trees. In chapter 12, we extend this to the case where they have more general connections.

Chapter 10

Graphical models

The previous chapters discussed models that relate the observed measurements to some aspect of the world that we wish to estimate. In each case, this relationship depended on a set of parameters and for each model we presented a learning algorithm that estimated these parameters.

Unfortunately, the utility of these models is limited because every element of the model depends on every other. For example, in generative models we model the joint probability of the observations and the world state. In many problems both of these quantities may be high dimensional. Consequently, the number of parameters required to characterize their joint density accurately is very large. Discriminative models suffer from the same pathology: if every element of the world state depends on every element of the data, a large number of parameters will be required to characterize this relationship. In practice, the required amount of training data and the computational burden of learning and inference reach impractical levels.

The solution to this problem is to reduce the dependencies between variables in the model by identifying (or asserting) some degree of redundancy. To this end, we introduce the idea of *conditional independence*, which is a way of characterizing these redundancies. We then introduce *graphical models* which are graph-based representations of the conditional independence relations. We discuss two different types of graphical models — directed and undirected — and we consider the implications for learning, inference, and drawing samples.

This chapter does not develop specific models or discuss vision applications. The goal is to provide the theoretical background for the models in subsequent chapters. We will illustrate the ideas with probability distributions where the constituent variables are discrete; however, almost all of the ideas transfer directly to the continuous case.

10.1 Conditional independence

When we first discussed probability distributions, we introduced the notion of independence (section 2.6). Two variables x_1 and x_2 are independent if their joint

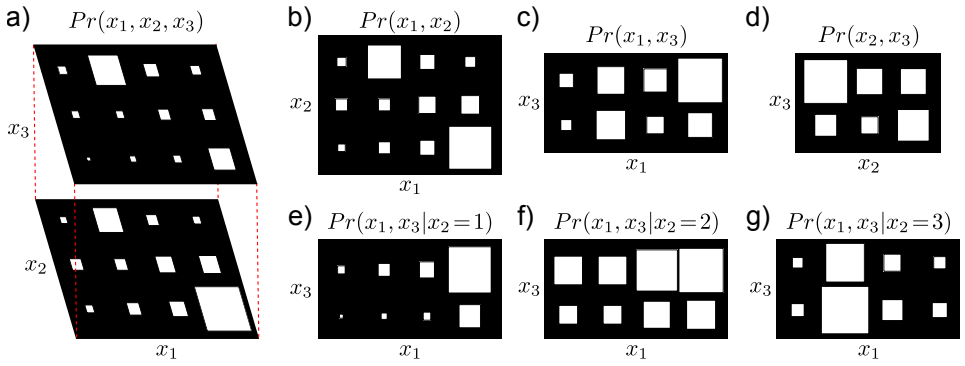


Figure 10.1 Conditional independence. a) Joint pdf of three discrete variables x_1, x_2, x_3, which take four, three, and two possible values, respectively. All 24 probability values sum to one. b) Marginalizing, we see that variables x_1 and x_2 are dependent; the conditional distribution of x_1 is different for different values of x_2 (the elements in each row are not in the same proportions)and vice-versa. c) Variables x_1 and x_3 are also dependent. d) Variables x_2 and x_3 are also dependent. e-g) However, x_1 and x_3 are conditionally independent *given* x_2. For fixed x_2, x_1 tells us nothing more about x_3 and vice-versa.

probability distribution factorizes as $Pr(x_1, x_2) = Pr(x_1)Pr(x_2)$. In layman's terms, one variable provides no information about the other if they are independent.

With more than two random variables, independence relations become more complex. The variable x_1 is said to be *conditionally independent of variable* x_3 *given variable* x_2 when x_1 and x_3 are independent for fixed x_2 (figure 10.1). In mathematical terms, we have

$$\begin{aligned} Pr(x_1|x_2, x_3) &= Pr(x_1|x_2) \\ Pr(x_3|x_1, x_2) &= Pr(x_3|x_2). \end{aligned} \tag{10.1}$$

Note that conditional independence relations are always symmetric; if x_1 is conditionally independent of x_3 given x_2, then it is also true that x_3 is independent of x_1 given x_2.

Confusingly, the conditional independence of x_1 and x_3 given x_2 does not mean that x_1 and x_3 are themselves independent. It merely implies that if we know variable x_2 then x_1 provides no further information about x_3 and vice-versa. One way that this can occur is in a chain of events: if event x_1 causes event x_2 and x_2 causes x_3 then the dependence of x_3 on x_1 might be entirely mediated by x_2.

Now consider decomposing the joint probability distribution $Pr(x_1, x_2, x_3)$ into the product of conditional probabilities. When x_1 is independent of x_3 given x_2, we find that

$$\begin{aligned} Pr(x_1, x_2, x_3) &= Pr(x_3|x_2, x_1)Pr(x_2|x_1)Pr(x_1) \\ &= Pr(x_3|x_2)Pr(x_2|x_1)Pr(x_1). \end{aligned} \tag{10.2}$$

The conditional independence relation means that the probability distribution factorizes in a certain way (and is hence redundant). This redundancy implies that we can describe the distribution with fewer parameters and so working with models with large numbers of variables becomes more tractable.

Throughout this chapter, we will explore the relationship between factorization of the distribution and conditional independence relations. To this end, we will introduce graphical models. These are graph-based representations that make both the factorization and the conditional independence relations easy to establish. In this book we will consider two different types of graphical model – directed and undirected graphical models – each of which corresponds to a different type of factorization.

10.2 Directed graphical models

A *directed graphical model* or *Bayesian network* represents the factorization of the joint probability distribution into a product of conditional distributions that take the form of a directed acyclic graph (DAG) so that

$$Pr(x_{1\ldots N}) = \prod_{n=1}^{N} Pr(x_n|x_{\mathrm{pa}[n]}), \tag{10.3}$$

where $\{x_n\}_{n=1}^N$ represent the constituent variables of the joint distribution and the function $\mathrm{pa}[n]$ returns the indices of variables that are parents of variable x_n.

We can visualize the factorization as a directed graphical model (figure 10.2) by adding one node per random variable and drawing an arrow to each variable x_n from each of its parents $x_{\mathrm{pa}[n]}$. This directed graphical model should never contain cycles. If it does, then the original factorization was not a valid probability distribution.

Problem 10.1
Problem 10.2

To retrieve the factorization from the graphical model, we introduce one factorization term per variable in the graph. If variable x_n is independent of all others (has no parents), then we write $Pr(x_n)$. Otherwise, we write $Pr(x_n|x_{\mathrm{pa}[n]})$ where the parents $x_{\mathrm{pa}[n]}$ consist of the set of variables with arrows that point to x_n.

10.2.1 Example 1

The graphical model in figure 10.2 represents the factorization

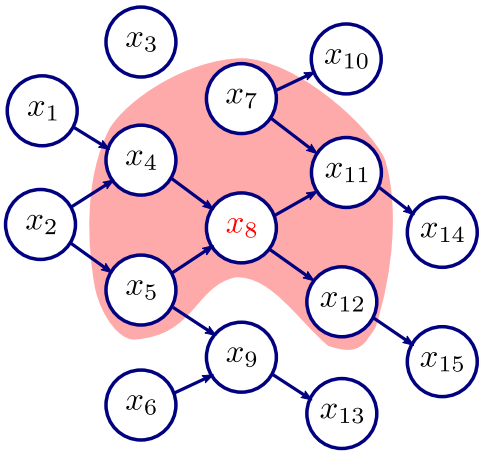


Figure 10.2 Example 1. A directed graphical model has one node per term in the factorization of the joint probability distribution. A node x_n with no incoming connections represents the term $Pr(x_n)$. A node x_n with incoming connections $\mathbf{x}_{\text{pa}[n]}$ represents the term $Pr(x_n|x_{\text{pa}[n]})$. Variable x_n is conditionally independent of all of the others given its *Markov blanket*. This comprises its parents, its children and other parents of its children. For example, the Markov blanket for variable x_8 is indicated by the shaded region.

$$\begin{aligned} Pr(x_1 \ldots x_{15}) \;=\; & Pr(x_1)Pr(x_2)Pr(x_3)Pr(x_4|x_1,x_2)Pr(x_5|x_2)Pr(x_6) \\ & Pr(x_7)Pr(x_8|x_4,x_5)Pr(x_9|x_5,x_6)Pr(x_{10}|x_7)Pr(x_{11}|x_7,x_8) \\ & Pr(x_{12}|x_8)Pr(x_{13}|x_9)Pr(x_{14}|x_{11})Pr(x_{15}|x_{12}). \end{aligned} \tag{10.4}$$

The graphical model (or factorization) implies a set of independence and conditional independence relations between the variables. Some statements about these relations can be made based on a superficial look at the graph. First, if there is no directed path between two variables following the arrow directions and they have no common ancestors, then they are independent. So, variable x_3 in figure 10.2 is independent of all of the other variables, and variables x_1 and x_2 are independent of each other. Variables x_4 and x_5 are not independent as they share an ancestor. Second, any variable is conditionally independent of all the other variables given its parents, children, and the other parents of its children (its *Markov blanket*). So, for example, variable x_8 in figure 10.2 is conditionally independent of the remaining variables given those in the shaded area.

For vision applications, these rules are usually sufficient to gain an understanding of the main properties of a graphical model. However, occasionally we may wish to test whether one arbitrary set of nodes is independent of another given a third. This is not easily established by looking at the graph, but can be tested using the following criterion:

The variables in set $\mathcal{A}$ are conditionally independent of those in set $\mathcal{B}$ given set $\mathcal{C}$ if all routes from $\mathcal{A}$ to $\mathcal{B}$ are blocked. A route is blocked at a node if (i) this node is in $\mathcal{C}$ and the arrows meet head to tail or tail to tail or (ii) neither this node nor any of its descendants are in $\mathcal{C}$ and the arrows meet head to head.

See Koller & Friedman 2009 for more details of why this is the case.

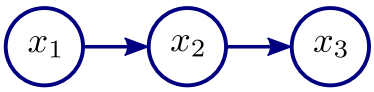


Figure 10.3 Example 2. Directed graphical model relating variables x_1, x_2, x_3 from figure 10.1. This model implies that the joint probability can be broken down as $Pr(x_1, x_2, x_3) = Pr(x_1)Pr(x_2|x_1)Pr(x_3|x_2)$.

10.2.2 Example 2

Figure 10.3 tells us that

$$Pr(x_1, x_2, x_3) = Pr(x_1)Pr(x_2|x_1)Pr(x_3|x_2). \tag{10.5}$$

In other words, this is the graphical model corresponding to the distribution in figure 10.1.

If we condition on x_2, then the only route from x_1 to x_3 is blocked at x_2 (the arrows meet head to tail here) and so x_1 must be conditionally independent of x_3 given x_2. We could have reached the same conclusion by noticing that the Markov blanket for variable x_1 is just variable x_2.

In this case, it is easy to prove this conditional independence relation algebraically. Writing out the conditional probability of x_1 given x_2 and x_3

$$\begin{aligned} Pr(x_1|x_2, x_3) &= \frac{Pr(x_1, x_2, x_3)}{Pr(x_2, x_3)} \\ &= \frac{Pr(x_1)Pr(x_2|x_1)Pr(x_3|x_2)}{\int Pr(x_1)Pr(x_2|x_1)Pr(x_3|x_2)dx_1} \\ &= \frac{Pr(x_1)Pr(x_2|x_1)}{\int Pr(x_1)Pr(x_2|x_1)dx_1}, \end{aligned} \tag{10.6}$$

we see that the final expression does not depend on x_3 and so we deduce that x_1 is conditionally independent of x_3 given x_2 as required.

Notice that the factorized distribution is more efficient to represent than the full version. The original distribution $Pr(x_1, x_2, x_3)$ (figure 10.1a) contains $4\times3\times2=24$ entries. However, the terms $Pr(x_1)$, $Pr(x_2|x_1)$, and $Pr(x_3|x_2)$ contain 4, $4\times3=12$, and $3\times2=6$ entries, respectively, giving a total of 22 entries. In this case, this is not a dramatic reduction, but in more practical situations it would be. For example, if each variable took 10 possible values, the full joint distribution would have $10\times10\times10=1000$ values, but the factorized distribution would have only $10+100+100=210$ values. For even larger systems, this can make a huge saving. One way to think about conditional independence relations is to consider them as redundancies in the full joint probability distribution.

10.2.3 Example 3

Finally, in figure 10.4 we present graphical models for the mixture of Gaussians, t-distribution and factor analysis models from chapter 7. These depictions imme-

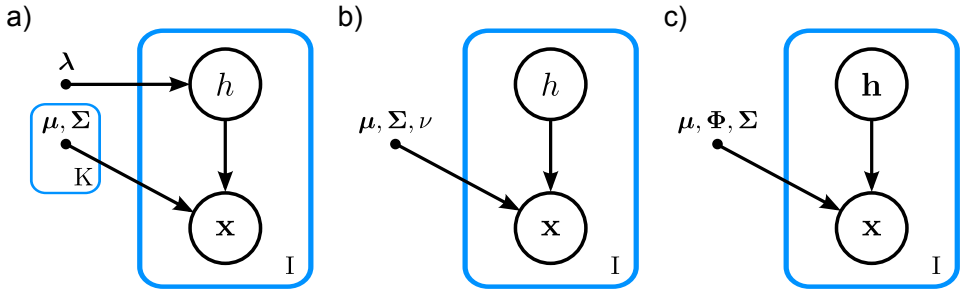


Figure 10.4 Example 3. Graphical models for a) mixture of Gaussians b) t-distribution and c) factor analysis. A node (black circle) represents a random variable. In a graphical model a bullet • represents a variable whose value is considered to be fixed. Each variable may be repeated many times, and this is indicated by a plate (blue rectangle) where the number of copies is indicated in the lower right corner. For example, in a) there are I copies of the training data $\{\mathbf{x}_i\}_{i=1}^{I}$ and I copies of the variable $\{h_i\}_{i=1}^{I}$. Similarly, there are K sets of parameters $\{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^{K}$, but just one weight vector $\boldsymbol{\lambda}$.

diately demonstrate that these models have very similar structures.

They also add several new features to the graphical representation. First, they include multidimensional variables. Second, they include variables that are considered as fixed and these are marked by a bullet •. We condition on the fixed variables, but do not define a probability distribution over them. Figure 10.4c depicts the factorization $Pr(\mathbf{h}_i, \mathbf{x}_i) = Pr(\mathbf{h}_i)Pr(\mathbf{x}_i|\mathbf{h}_i, \boldsymbol{\mu}, \boldsymbol{\Phi}, \boldsymbol{\Sigma})$.

Finally, we have also used *plate* notation. A plate is depicted as a rectangle with a number in the corner. It indicates that the quantities inside the rectangle should be repeated the given number of times. For example, in figure 10.4c there are I copies $\{\mathbf{x}_i, \mathbf{h}_i\}_{i=1}^{I}$ of the variables $\mathbf{x}$ and $\mathbf{h}$ but only one set of parameters $\boldsymbol{\mu}, \boldsymbol{\Phi}$, and $\boldsymbol{\Sigma}$.

10.2.4 Summary

To summarize, we can think about the structure of the joint probability distribution in three ways. First, we can consider the way that the probability distribution factorizes. Second, we can examine the directed graphical model. Third, we can think about the conditional independence relations.

There is a one-to-one mapping between directed graphical models (acyclic directed graphs of conditional probability relations) and factorizations. However, the relationship between the graphical model (or factorization) and the conditional independence relations is more complicated. A directed graphical model (or its equivalent factorization) determines a set of conditional independence relations. However, as we shall see later in this chapter, there are some sets of conditional independence relations that cannot be represented by directed graphical models.

10.3 Undirected graphical models

In this section we introduce a second family of graphical models. Undirected graphical models represent probability distributions over variables $\{x_n\}_{n=1}^N$ that take the form of a product of *potential functions* $\phi[x_{1\ldots N}]$ so that

Problem 10.3
Problem 10.4
Problem 10.5

$$Pr(x_{1\ldots N}) = \frac{1}{Z}\prod_{c=1}^{C}\phi_c[x_{1\ldots N}], \tag{10.7}$$

where the potential function $\phi_c[x_{1\ldots N}]$ always returns a positive number. Since the probability increases when $\phi_c[x_{1\ldots N}]$ increases, each of these functions modulates the tendency for the variables $x_{1\ldots N}$ to take certain values. The probability is greatest where all of the functions $\phi_{1\ldots C}$ return high values. However, it should be emphasized that potential functions are *not* the same as conditional probabilities, and there is not usually a clear way to map from one to the other.

The term Z is known as the *partition function* and normalizes the product of these positive functions so that the total probability is one. In the discrete case, it would be computed as

$$Z = \sum_{x_1}\sum_{x_2}\cdots\sum_{x_N}\prod_{c=1}^{C}\phi_c[x_{1\ldots N}]. \tag{10.8}$$

For realistically sized systems, this sum will be intractable; we will not be able to compute Z and hence will only be able to compute the overall probability up to an unknown scale factor.

We can equivalently write equation 10.7 as

$$Pr(x_{1\ldots N}) = \frac{1}{Z}\exp\left[-\sum_{c=1}^{C}\psi_c[x_{1\ldots N}]\right], \tag{10.9}$$

where $\psi_c[x_{1\ldots N}] = -\log[\phi_c[x_{1\ldots N}]]$. When written in this form, the probability is referred to as a *Gibbs distribution*. The terms $\psi_c[x_{1\ldots N}]$ are functions that may return any real number and can be thought of as representing a cost for every combination of labels $x_{1\ldots N}$. As the cost increases, the probability decreases. The total cost $\sum_{c=1}^{C}\psi_c[x_{1\ldots N}]$ is sometimes known as the *energy*, and the process of fitting the model (increasing the probability) is hence sometimes termed *energy minimization*.

When each potential function $\phi[\bullet]$ (or alternatively each cost function $\psi[\bullet]$) addresses all of the variables $x_{1\ldots N}$ the undirected graphical model is known as a *product of experts*. However, in computer vision it is more common for each potential function to operate on a subset of the variables $\mathcal{S} \subset \{x_n\}_{n=1}^N$. These subsets are called *cliques* and it is the choice of these cliques that determines the conditional independence relations. Denoting the c^{th} clique by $\mathcal{S}_c$ we can rewrite equation 10.7 as

$$Pr(x_{1\ldots N}) = \frac{1}{Z}\prod_{c=1}^{C}\phi_c[\mathcal{S}_c]. \tag{10.10}$$

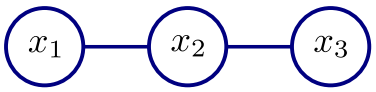


Figure 10.5 Example 1. Undirected graphical model relating variables x_1, x_2, and x_3. This model implies that the joint probability can be factorized as $Pr(x_1, x_2, x_3) = \frac{1}{Z}\phi_1[x_1, x_2]\phi_2[x_2, x_3]$.

In other words, the probability distribution is factorized into a product of terms, each of which only depends on a subset of variables. In this situation, the model is sometimes referred to as a *Markov random field*.

To visualize the undirected graphical model, we draw one node per random variable. Then, for every clique $\mathcal{S}_c$ we draw a connection from every member variable $x_i \in \mathcal{S}_c$ to every other member variable.

Moving in the opposite direction, we can take a graphical model and establish the underlying factorization using the following method. We add one term to the factorization per *maximal clique* (see figure 10.6). A maximal clique is a fully connected subset of nodes (i.e., a subset where every node is connected to every other) where it is not possible to add another node and remain fully connected.

It is much easier to establish the conditional independence relations from an undirected graphical model than for directed graphical models. They can be found using the following property:

One set of nodes is conditionally independent of another given a third if the third set separates them (prevents a path from the first node to the second).

It follows that a node is conditionally independent of all other nodes given its set of immediate neighbors, and so these neighbors form the Markov blanket.

10.3.1 Example 1

Consider the graphical model in figure 10.5. This represents the factorization

$$Pr(x_1, x_2, x_3) = \frac{1}{Z}\phi_1[x_1, x_2]\phi_2[x_2, x_3]. \tag{10.11}$$

We can immediately see that variable x_1 is conditionally independent of variable x_3 given x_2 because x_2 separates the other two variables: it blocks the path from x_1 to x_3. In this case, the conditional independence relation is easy to prove:

$$\begin{aligned} Pr(x_1|x_2, x_3) &= \frac{Pr(x_1, x_2, x_3)}{Pr(x_2, x_3)} \\ &= \frac{\frac{1}{Z}\phi_1[x_1, x_2]\phi_2[x_2, x_3]}{\int \frac{1}{Z}\phi_1[x_1, x_2]\phi_2[x_2, x_3]dx_1} \\ &= \frac{\phi_1[x_1, x_2]}{\int \phi_1[x_1, x_2]dx_1}. \end{aligned} \tag{10.12}$$

The final expression does not depend on x_3 and so we conclude that x_1 is conditionally independent of x_3 given x_2.

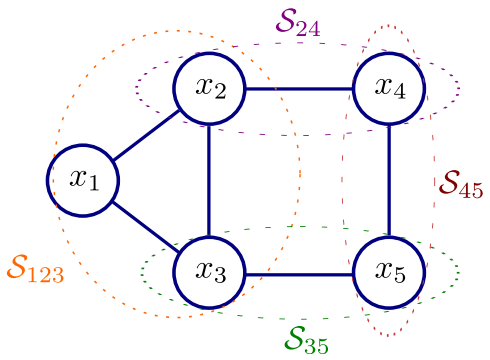


Figure 10.6 Example 2. Undirected graphical model representing variables $\{x_i\}_{i=1}^{5}$. The associated probability distribution factorizes into a product of one potential function per maximal clique. The clique $\mathcal{S}_{45} = \{x_4, x_5\}$ is a maximal clique as there is no other node that we can add that connects to every node in the clique. The clique $\mathcal{S}_{23} = \{x_2, x_3\}$ is not a maximal clique as it is possible to add node x_1 and all three nodes in the new clique are connected to each other.

10.3.2 Example 2

Consider the graphical model in figure 10.6. There are four maximal cliques in this graph, and so it represents the factorization

$$Pr(x_{1\ldots5}) = \frac{1}{Z}\phi_1[x_1, x_2, x_3]\phi_2[x_2, x_4]\phi_3[x_3, x_5]\phi_4[x_4, x_5]. \tag{10.13}$$

We can deduce various conditional independence relations from the graphical representation. For example, variable x_1 is conditionally independent of variables x_4 and x_5 given x_2 and x_3, and variable x_5 is independent of variables x_1 and x_2 given x_3 and x_4, and so on.

Note also that the factorization

$$Pr(x_{1\ldots5}) = \frac{1}{Z}\left(\phi_1[x_1, x_2]\phi_2[x_2, x_3]\phi_3[x_1, x_3]\right)\phi_4[x_2, x_4]\phi_5[x_3, x_5]\phi_6[x_4, x_5] \tag{10.14}$$

creates the same graphical model; there is a many-to-one mapping from factorizations to undirected graphical models (as opposed to the one-to-one mapping for directed graphical models). When we compute a factorization from the graphical model based on the maximal cliques we do so in a conservative way. It is possible that there are further redundancies which were not made explicit by the undirected graphical model.

10.4 Comparing directed and undirected graphical models

In sections 10.2 and 10.3 we have discussed directed and undirected graphical models, respectively. Each graphical model represents a factorization of the probability distribution. We have presented methods to extract the conditional independence relations from each type of graphical model. The purpose of this section is to argue that these representations are not equivalent. There are patterns of conditional independence that can be represented by directed graphical models but not undirected graphical models and vice versa.

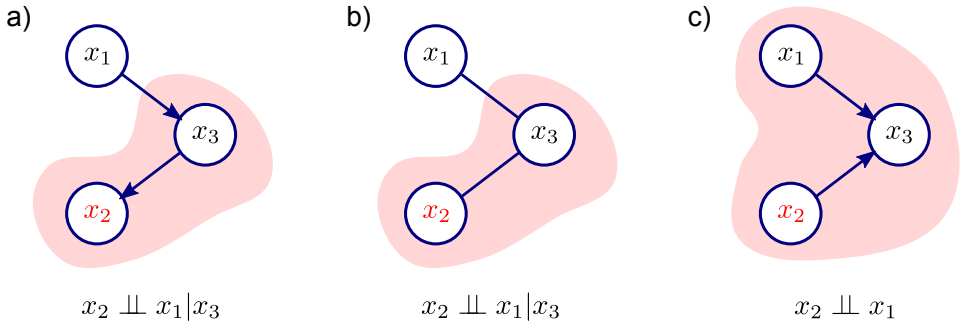


Figure 10.7 Directed vs. undirected graphical models. a) Directed graphical model with three nodes. There is only one conditional independence relation implied by this model: the node x_3 is the Markov blanket of node x_2 (shaded area) and so $x_2 \perp\!\!\!\perp x_1 | x_3$, where the notation $\perp\!\!\!\perp$ can be read as 'is independent of.' b) This undirected graphical model implies the same conditional independence relation. c) Second directed graphical model. The relation $x_2 \perp\!\!\!\perp x_1 | x_3$ is no longer true, but x_1 and x_2 are independent if we don't condition on x_3 so we can write $x_2 \perp\!\!\!\perp x_1$. There is no undirected graphical model with three variables that has this pattern of independence and conditional independence.

Problem 10.6
Problem 10.7
Problem 10.8

Figures 10.7a-b show an undirected and directed graphical model that do represent the same conditional independence relations. However, figure 10.7c shows a directed graphical model for which there is no equivalent undirected graphical model. There is simply no way to induce the same pattern of independence and conditional independence with an undirected graphical model.

Conversely, figure 10.8a shows an undirected graphical model that induces a pattern of conditional independence relations that cannot be replicated by any directed graphical model. Figure 10.8b shows a directed graphical model that is close, but still not equivalent; the Markov blanket of x_2 is different in each model and so are its conditional independence relations.

We conclude from this brief argument that directed and undirected graphical models do not represent the same subset of independence and conditional independence relations, and so we cannot eliminate one or the other from our consideration. In fact, there are other patterns of conditional independence that cannot be represented by either type of model. However, these will not be considered in this book. For further information concerning the families of distributions that can be represented by different types of graphical model, consult Barber (2012) or Koller & Friedman (2009).

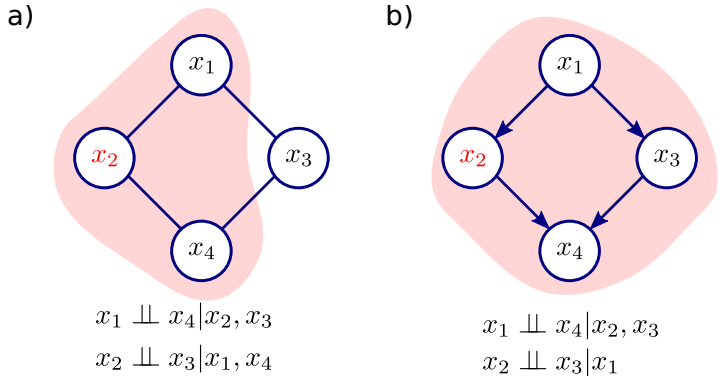


Figure 10.8 Directed vs. undirected models. a) This undirected graphical model induces two conditional independence relations. However, there is no equivalent directed graphical model that produces the same pattern. b) This directed graphical model also induces two conditional independence relations but they are not the same. In both cases, the shaded region represents the Markov blanket of variable x_2.

10.5 Graphical models in computer vision

We will now introduce a number of common vision models and look at their associated graphical models. We will discuss each of these in detail in subsequent chapters. However, it is instructive to see them together.

Problem 10.9
Problem 10.10

Figure 10.9a shows the graphical model for a *hidden Markov model* or *HMM*. We observe a sequence of measurements $\{\mathbf{x}_n\}_{n=1}^N$ each of which tells us something about the corresponding discrete world state $\{\mathbf{w}_n\}_{n=1}^N$. Adjacent world states are connected together so that the previous world state influences the current one and potentially resolves situations where the measurements are ambiguous. A prototypical application would be tracking sequences of sign language gestures (figure 10.9b). There is information at each frame about which gesture is present, but it may be ambiguous. However, we can impose prior knowledge that certain signs are more likely to follow others using the HMM and get an improved result.

Figure 10.9c represents a *Markov tree*. Again we observe a number of measurements, each of which provides information about the associated discrete world state. However, the world states are now connected in a tree structure. A prototypical application would be human body fitting (figure 10.9d) where each unknown world state represents a body part. The parts of the body naturally have a tree structure and so it makes sense to build a model that exploits this.

Figure 10.9e illustrates the use of a *Markov random field* or *MRF* as a prior. The MRF here describes the world state as a grid of undirected connections. Each node might correspond to a pixel. There is also a measurement variable associated with each world state variable. These pairs are connected with directed links, so overall this is a mixed model (partly directed and partly undirected). A prototypical application of an MRF in vision would be for semantic labeling (figure 10.9f). The

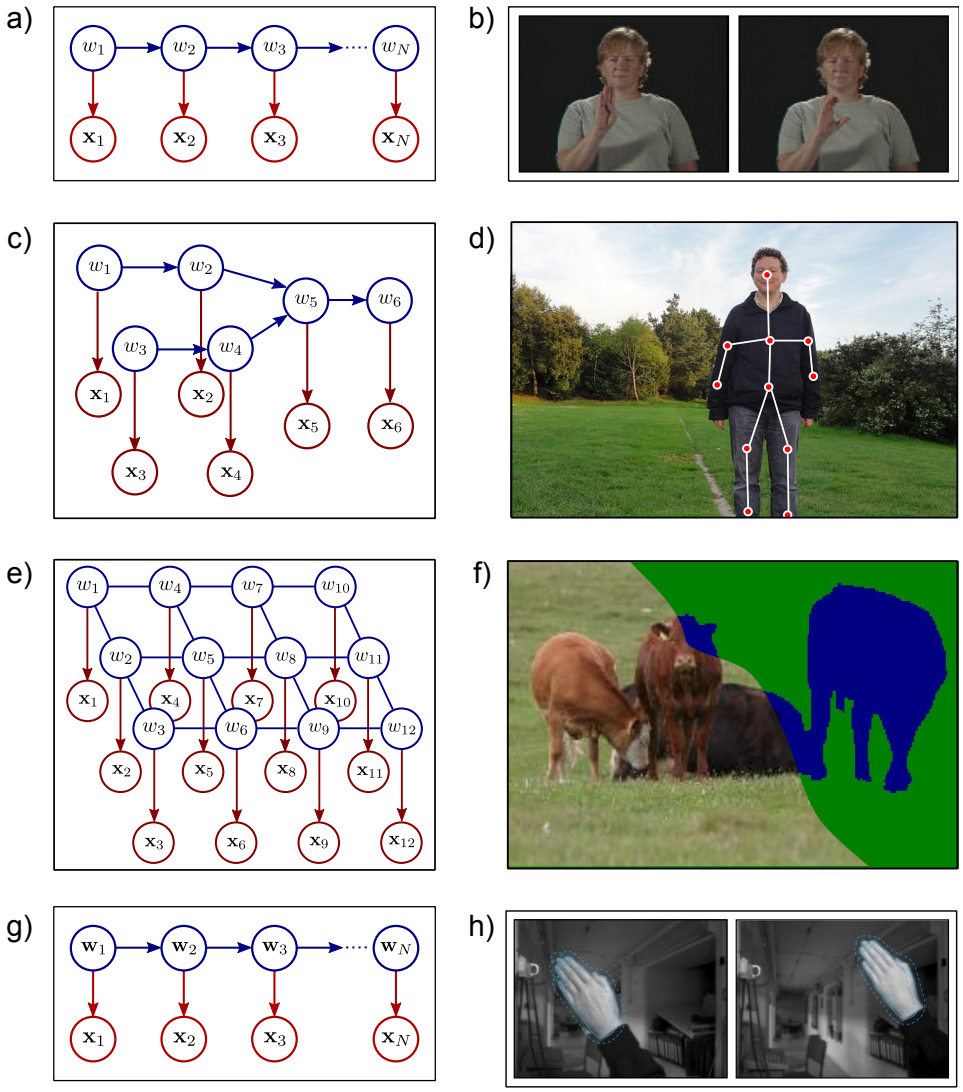


Figure 10.9 Commonly used graphical models in computer vision. a) Hidden Markov model (HMM). b) One possible application of the HMM is interpreting sign language sequences. The choice of sign at time n depends on the sign at time $n-1$. c) Markov tree. d) An example application is fitting a tree-structured body model e) Markov random field (MRF) prior with independent observations. f) The MRF is often used as a prior distribution in semantic labeling tasks. Here the goal is to infer a binary label at each pixel determining whether it belongs to the cow or the grass. g) Kalman filter. An example application is tracking an object through a sequence. It has the same graphical model as the HMM, but the unknown quantities are continuous as opposed to discrete.

measurements constitute the RGB values at each position. The world state at each pixel is a discrete variable that determines the class of object present (i.e., cow vs. grass). The Markov random field prior ties together all of the individual classifiers to help yield a solution that makes global sense.

Finally, figure 10.9g shows the *Kalman filter*. This has the same graphical model as the hidden Markov model but in this case the world state is continuous rather than discrete. A prototypical application of the Kalman filter is for tracking objects through a time sequence (figure 10.9h). At each time, we might want to know the 2D position, size, and orientation of the hand. However, in a given frame the measurements might be poor: the frame may be blurred or the object may be temporarily occluded. By building a model that connects the estimates from adjacent frames, we can increase the robustness to these factors; earlier frames can resolve the uncertainty in the current ambiguous frame.

Notice that all of these graphical models have directed links from the world $\mathbf{w}$ to the data $\mathbf{x}$ that indicate a relationship of the form $Pr(\mathbf{x}|\mathbf{w})$. Hence, they all construct a probability distribution over the data and are generative models. We will also consider discriminative models, but, historically speaking, generative models of this kind have been more important. Each model is quite sparsely connected: each data variable $\mathbf{x}$ connects only to one world state variable $\mathbf{w}$, and each world state variable connects to only a few others. The result of this is that there are many conditional independence relations in the model. We will exploit these redundancies to develop efficient algorithms for learning and inference.

We will return to all of these models later in the book. We investigate the hidden Markov model and the Markov tree in chapter 11. We discuss the Markov random field in chapter 12, and we will present the Kalman filter in chapter 19. The remaining part of this chapter answers two questions: (i) How can we perform inference when there are a large number of unknown world variables? (ii) What are the implications of using a directed graphical model vs. an undirected one?

10.6 Inference in models with many unknowns

We will now consider inference in these models. Ideally, we would compute the full posterior distribution $Pr(w_{1\ldots N}|\mathbf{x}_{1\ldots N})$ using Bayes' rule. However, the unknown world states in the preceding models are generally much larger than previously considered in this book and this makes inference challenging.

For example, consider the space of world states in the HMM example. If we are given 1000 frames of video and there are 500 common signs in the sign language, then there are 500^{1000} possible states. It is clearly not practical to compute and store the posterior probability associated with each. Even when the world states are continuous, computing and storing the parameters of a high-dimensional probability model is still problematic. Fortunately, there are alternative approaches to inference, which we now consider in turn.

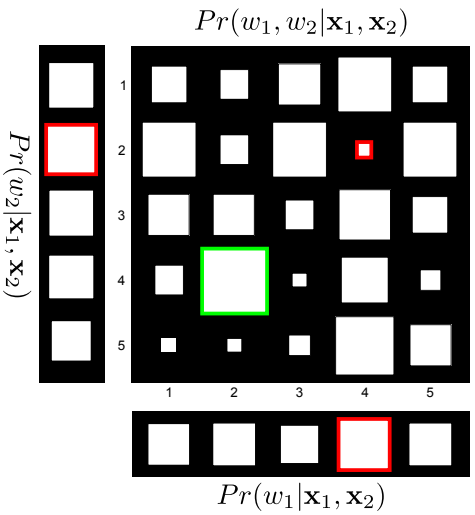


Figure 10.10 MAP solution vs. max marginals solution. The main figure shows the joint posterior distribution $Pr(w_1, w_2|\mathbf{x}_1, \mathbf{x}_2)$. The MAP solution is at the peak of this distribution at $w_1 = 2, w_2 = 4$ (highlighted in green). The figure also shows the two marginal distributions $Pr(w_1|\mathbf{x}_1, \mathbf{x}_2)$ and $Pr(w_2|\mathbf{x}_1, \mathbf{x}_2)$. The maximum marginals solution is computed by individually finding the maximum of each marginal distributions, which gives the solution $w_1 = 4, w_2 = 2$ (highlighted in red). For this distribution, this is very unrepresentative; although these labels are individually likely, they rarely co-occur and the joint posterior for this combination has low probability.

10.6.1 Finding the MAP solution

One obvious possibility is to find the maximum a posteriori (MAP) solution:

$$\begin{aligned}\hat{w}_{1\ldots N} &= \underset{w_{1\ldots N}}{\operatorname{argmax}}\left[Pr(w_{1\ldots N}|\mathbf{x}_{1\ldots N})\right] \\ &= \underset{w_{1\ldots N}}{\operatorname{argmax}}\left[Pr(\mathbf{x}_{1\ldots N}|w_{1\ldots N})Pr(w_{1\ldots N})\right].\end{aligned}$$

This is still far from trivial. The number of world states is extremely large so we cannot possibly explore every one and take the maximum. We must employ intelligent and efficient algorithms that exploit the redundancies in the distribution to find the correct solution where possible. However, as we shall see, for some models there is no known polynomial algorithm to find the MAP solution.

10.6.2 Finding the marginal posterior distribution

An alternative strategy is to find the marginal posterior distributions:

$$Pr(w_n|\mathbf{x}_{1\ldots N}) = \int\int Pr(w_{1\ldots N}|\mathbf{x}_{1\ldots N}) dw_{1\ldots n-1}\, dw_{n+1\ldots N}. \tag{10.15}$$

Since each of these distributions is over a single label, it is not implausible to compute and store each one separately. Obviously it is not possible to do this by directly computing the (extremely large) joint distribution and marginalizing it directly. We must use algorithms that exploit the conditional independence relations in the distribution to efficiently compute the marginals.

10.6.3 Maximum marginals

If we want a single estimate of the world state, we could return the maximum values of the marginal distributions, giving the criterion

$$\hat{w}_n = \underset{w_n}{\operatorname{argmax}} \left[Pr(w_n|\mathbf{x}_{1\ldots N}) \right]. \tag{10.16}$$

This produces estimates of each world state that are individually most probable, but which may not reflect the joint statistics. For example, world state $w_n = 4$ might be the most probable value for the n^{th} world state and $w_m = 6$ might be the most probable value for the m^{th} world state, but it could be that the posterior probability for this configuration is zero: although the states are individually probable, they never co-occur (figure 10.10).

10.6.4 Sampling the posterior

For some models, it is intractable to compute either the MAP solution or the marginal distributions. One possibility in this circumstance is to draw samples from the posterior distribution. Methods based on sampling the posterior would fall under the more general category of *approximate inference* as they do not normally return the true answer.

Having drawn a number of samples from the posterior, we can approximate the posterior probability distribution as a mixture of delta functions where there is one delta function at each of the sample positions. Alternatively, we could make estimates of marginal statistics such as the mean and variance based on the sampled values or select the sample with the highest posterior probability as an estimate of the MAP state; this latter approach has the advantage of being consistent with the full posterior distribution (as opposed to maximum marginals which is not) even if we cannot be sure that we have the correct answer.

An alternative way to compute a point estimate from a set of samples from the posterior is to compute the *empirical max-marginals*. We estimate the marginal probability distributions by looking at the marginal statistics of the samples. In other words, we consider one variable w_n at a time and look at the distribution of different values observed. For a discrete distribution, this information is captured in a histogram. For a continuous distribution, we could fit a univariate model such as a normal distribution to these values to summarize them.

10.7 Drawing samples

We have seen that some of the approaches to inference require us to draw samples from the posterior distribution. We will now discuss how to do this for both directed and undirected models and we will see that this is generally more straightforward in directed models.

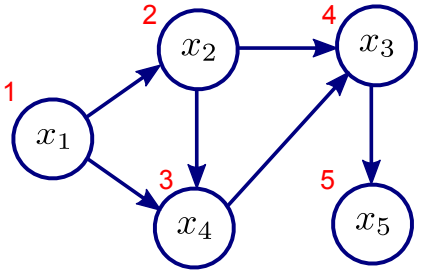


Figure 10.11 Ancestral sampling. We work our way through the graph in an order (red number) that guarantees that the parents of every node are visited before the node itself. At each step we draw a sample conditioned on the values of the samples at the parents. This is guaranteed to produce a valid sample from the full joint distribution.

10.7.1 Sampling from directed graphical models

Directed graphical models take the form of directed acyclic graphs of conditional probability relations that have the following algebraic form:

$$Pr(x_{1\ldots N}) = \prod_{n=1}^{I} Pr(x_n|x_{\mathrm{pa}[n]}). \tag{10.17}$$

It is relatively easy to sample from a directed graphical model using a technique known as *ancestral sampling*. The idea is to sample each variable in the network in turn, where the order is such that all parents of a node are sampled before the node itself. At each node, we condition on the observed sample values of the parents.

The simplest way to understand this is with an example. Consider the directed graphical model in figure 10.11 whose probability distribution factorizes as

$$Pr(x_1, x_2, x_3, x_4, x_5) = Pr(x_1)Pr(x_2|x_1)Pr(x_3|x_4, x_2)Pr(x_4|x_2, x_1)Pr(x_5|x_3). \tag{10.18}$$

To sample from this model we first identify x_1 as a node with no parents and draw a sample from the distribution $Pr(x_1)$. Let us say the observed sample at x_1 took value α_1.

We now turn to the remaining nodes. Node x_2 is the only node in the network where all of the parents have been processed, and so we turn our attention here next. We draw a sample from the distribution $Pr(x_2|x_1 = \alpha_1)$ to yield a sample α_2. We now see that we are not yet ready to sample from x_3 as not all of its parents have been sampled, but we can sample x_4 from the distribution $Pr(x_4|x_1 = \alpha_1, x_2 = \alpha_2)$ to yield the value α_4. Continuing this process we draw x_3 from $Pr(x_3|x_2 = \alpha_2, x_4 = \alpha_4)$ and finally x_5 from $Pr(x_5|x_3 = \alpha_3)$.

The resulting vector $\mathbf{w}^* = [\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5]$ is guaranteed to be a valid sample from the full joint distribution $Pr(x_1, x_2, x_3, x_4, x_5)$. An equivalent way to think about this algorithm is to consider it as working through the terms in the factorized joint distribution (right hand side of equation 10.18) sampling from each in turn conditioned on the previous values.

10.7.2 Sampling from undirected graphical models

Unfortunately, it is much harder to draw samples from undirected models except in certain special cases (e.g., where the variables are continuous and Gaussian or where the graph structure takes the form of a tree). In general graphs we cannot use ancestral sampling because (i) there is no sense in which any variable is a parent to any other so we don't know which order to sample in and (ii) the terms $\phi[\bullet]$ in the factorization are not probability distributions anyway.

Algorithm 10.1

One way to generate samples from any complex high-dimensional probability distribution is to use a *Markov chain Monte Carlo* (MCMC) method. The principle is to generate a series (chain) of samples from the distribution, so that each sample depends directly on the previous one (hence "Markov"). However, the generation of the sample is not completely deterministic (hence "Monte Carlo").

One of the simplest MCMC methods is *Gibbs sampling* which proceeds as follows. First, we randomly choose the initial state $\mathbf{x}^{[0]}$ using any method. We generate the next sample in the chain $\mathbf{x}^{[1]}$ by updating the state at each dimension $\{x_n\}_{n=1}^N$ in turn (in any order). To update the n^{th} dimension x_n we fix the other $N-1$ dimensions and draw from the conditional distribution $Pr(x_n|x_{1\ldots N\backslash n})$ where the set $x_{1\ldots N\backslash n}$ denotes all of the N variables $x_1, x_2 \ldots x_N$ *except* x_n. Having modified every dimension in this way, we obtain the second sample in the chain. This idea is illustrated in figure 10.12 for the multivariate normal distribution.

When this procedure is repeated a very large number of times, so that the initial conditions are forgotten, a sample from this sequence can be considered as a draw from the distribution $Pr(x_{1\ldots N})$. Although this is not immediately obvious (and a proof is beyond the scope of this book) this procedure does clearly have some sensible properties: since we are sampling from the conditional probability distribution at each pixel, we are more likely to change the current value to one which has an overall higher probability. However, the stochastic update rule provides the possibility of (infrequently) visiting less probable regions of the space.

For undirected graphical models, the conditional distribution $Pr(x_n|x_{1\ldots N\backslash n})$ can be quite efficient to evaluate because of the conditional independence properties: variable x_n is conditionally independent of the rest of the nodes given its immediate neighbors, and so computing this term only involves the immediate neighbors. However, overall this method is extremely inefficient: it requires a large amount of computational effort to generate even a single sample. Sampling from directed graphical models is far easier.

10.8 Learning

In the section 10.7 we argued that sampling from directed graphical models is considerably easier than sampling from undirected graphical models. In this section, we consider learning in each type of model and come to a similar conclusion. Note that we are not discussing the learning of the graph structure here; we are talking about learning the parameters of the model itself. For directed graphical models, these parameters would determine the conditional distributions $Pr(x_n|x_{\mathrm{pa}[n]})$

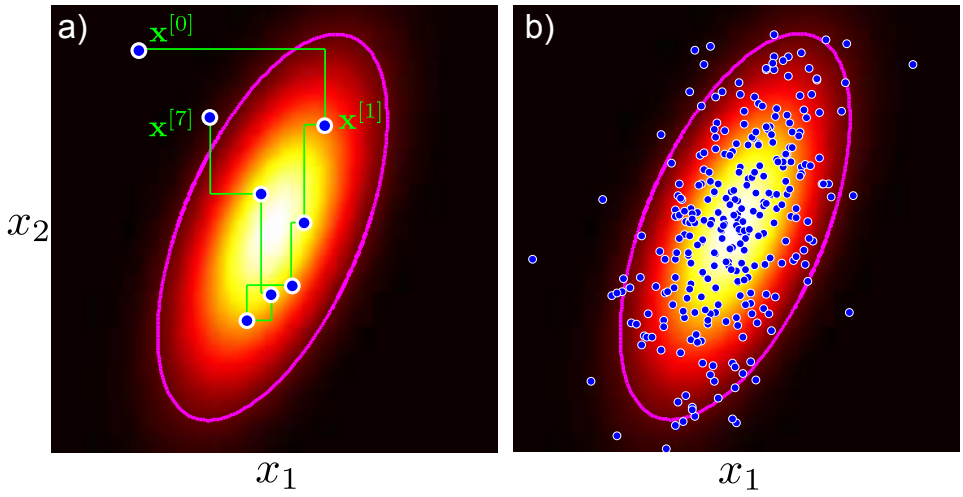


Figure 10.12 Gibbs sampling. We generate a chain of samples by cycling through each dimension in turn and drawing a sample from the conditional distribution of that dimension given that the others are fixed. a) For this 2D multivariate normal distribution we start at a random position $\mathbf{x}^{[0]}$. We alternately draw samples from the conditional distribution of the first dimension keeping the second fixed (horizontal changes) and the second dimension keeping the first fixed (vertical changes). For the multivariate normal, these conditional distributions are themselves normal (section 5.5). Each time we cycle through both of the dimensions, we create a new sample $\mathbf{x}^{[t]}$. b) Many samples generated using this method.

and for undirected graphical models they would determine the potential functions $\phi_c[\mathbf{x}_{1\ldots N}]$.

10.8.1 Learning in directed graphical models

Any directed graphical model can be written in the factorized form

$$Pr(x_1 \ldots x_N) = \prod_{n=1}^{N} Pr(x_n | x_{\mathrm{pa}[n]}, \boldsymbol{\theta}), \tag{10.19}$$

where the conditional probability relations form a directed acyclic graph, and $\boldsymbol{\theta}$ denotes the parameters of the model. For example, in the discrete distributions that we have focused on in this chapter, an individual conditional model might be

$$Pr(x_2 | x_1 = k) = \mathrm{Cat}_{x_2}[\boldsymbol{\lambda}_k] \tag{10.20}$$

where the parameters here are $\{\boldsymbol{\lambda}_k\}_{k=1}^{K}$. In general, the parameters can be learned using the maximum likelihood method by finding

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\left[\prod_{i=1}^{I}\prod_{n=1}^{N} Pr(x_{i,n}|x_{i,\mathrm{pa}[n]},\boldsymbol{\theta})\right] \tag{10.21}$$

$$= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\left[\sum_{i=1}^{I}\sum_{n=1}^{N} \log[Pr(x_{i,n}|x_{i,\mathrm{pa}[n]},\boldsymbol{\theta})]\right], \tag{10.22}$$

where $x_{i,n}$ represents the n^{th} dimension of the i^{th} training example. This criterion leads to simple learning algorithms and often the maximum likelihood parameters can be computed in closed form.

10.8.2 Learning in undirected graphical models

An undirected graphical model is written as

$$Pr(\mathbf{x}) = \frac{1}{Z}\prod_{c=1}^{C}\phi_c[\mathbf{x},\boldsymbol{\theta}], \tag{10.23}$$

where $\mathbf{x} = [x_1, x_2, \ldots, x_N]$ and we have assumed that the training samples are independent. However, in this form we must constrain the parameters so that they ensure that each $\phi_c[\bullet]$ always returns a positive number. A more practical approach is to re-parameterize the undirected graphical model in the form of the Gibbs distribution,

$$Pr(\mathbf{x}) = \frac{1}{Z}\exp\left[-\sum_{c=1}^{C}\psi_c[x_{1\ldots N},\boldsymbol{\theta}]\right] \tag{10.24}$$

so that we do not have to worry about constraints on the parameters.

Given I training examples $\{\mathbf{x}_i\}_{i=1}^{I}$, we aim to fit parameters $\boldsymbol{\theta}$. Assuming that the training examples are independent, the maximum likelihood solution is

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\left[\frac{1}{Z(\boldsymbol{\theta})^I}\exp\left[-\sum_{i=1}^{I}\sum_{c=1}^{C}\psi_c(\mathbf{x}_i,\boldsymbol{\theta})\right]\right]$$

$$= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\left[-I\log[Z(\boldsymbol{\theta})] - \sum_{i=1}^{I}\sum_{c=1}^{C}\psi_c(\mathbf{x}_i,\boldsymbol{\theta})\right], \tag{10.25}$$

where as usual we have taken the log to simplify the expression.

To maximize this expression we calculate the derivative of the log likelihood L with respect to the parameters $\boldsymbol{\theta}$:

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = -I\frac{\partial \log[Z(\boldsymbol{\theta})]}{\partial \boldsymbol{\theta}} - \sum_{i=1}^{I}\sum_{c=1}^{C}\frac{\partial \psi_c(\mathbf{x}_i,\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{10.26}$$

$$= -I\frac{\partial \log\left[\sum_{\mathbf{x}_i}\exp\left[-\sum_{c=1}^{C}\psi_c(\mathbf{x}_i,\boldsymbol{\theta})\right]\right]}{\partial \boldsymbol{\theta}} - \sum_{i=1}^{I}\sum_{c=1}^{C}\frac{\partial \psi_c(\mathbf{x}_i,\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}.$$

The second term is readily computable but the first term involves an intractable sum over all possible values of the variable $\mathbf{x}$: we cannot compute the derivative with respect to the parameters for reasonable-sized models and so learning is difficult. Moreover, we cannot evaluate the original probability expression (equation 10.23) as this too contains an intractable sum. Consequently, we can't compute the derivative using finite differences either.

In short, we can neither find an algebraic solution nor use a straightforward optimization technique as we cannot compute the gradient. The best that we can do is to approximate the gradient.

Contrastive divergence

Algorithm 10.2

One possible solution to this problem is the *contrastive divergence* algorithm. This is a method for approximating the gradient of the log likelihood with respect to parameters $\boldsymbol{\theta}$ for functions with the general form,

$$Pr(\mathbf{x}) = \frac{1}{Z(\boldsymbol{\theta})} f[\mathbf{x}, \boldsymbol{\theta}], \tag{10.27}$$

where $Z(\boldsymbol{\theta}) = \sum_{\mathbf{x}} f[\mathbf{x}, \boldsymbol{\theta}]$ is the normalizing constant and the derivative of the log likelihood is

$$\frac{\partial \log[Pr(\mathbf{x})]}{\partial \boldsymbol{\theta}} = -\frac{\partial \log[Z(\boldsymbol{\theta})]}{\partial \boldsymbol{\theta}} + \frac{\partial \log[f[\mathbf{x}, \boldsymbol{\theta}]]}{\partial \boldsymbol{\theta}}. \tag{10.28}$$

The main idea behind contrastive divergence follows from some algebraic manipulation of the first term:

$$\begin{aligned}
\frac{\partial \log[Z(\boldsymbol{\theta})]}{\partial \boldsymbol{\theta}} &= \frac{1}{Z(\boldsymbol{\theta})} \frac{\partial Z(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\
&= \frac{1}{Z(\boldsymbol{\theta})} \frac{\partial \sum_{\mathbf{x}} f[\mathbf{x}, \boldsymbol{\theta}]}{\partial \boldsymbol{\theta}} \\
&= \frac{1}{Z(\boldsymbol{\theta})} \sum_{\mathbf{x}} \frac{\partial f[\mathbf{x}, \boldsymbol{\theta}]}{\partial \boldsymbol{\theta}} \\
&= \frac{1}{Z(\boldsymbol{\theta})} \sum_{\mathbf{x}} f[\mathbf{x}, \boldsymbol{\theta}] \frac{\partial \log[f[\mathbf{x}, \boldsymbol{\theta}]]}{\partial \boldsymbol{\theta}} \\
&= \sum_{\mathbf{x}} Pr(\mathbf{x}) \frac{\partial \log[f[\mathbf{x}, \boldsymbol{\theta}]]}{\partial \boldsymbol{\theta}}.
\end{aligned} \tag{10.29}$$

where we have used the relation $\partial \log f[\mathbf{x}]/\partial x = (\partial f[\mathbf{x}]/\partial x)/f[\mathbf{x}]$ between the third and fourth lines.

The final term in equation 10.29 is the expectation of the derivative of $\log[f[\mathbf{x}, \boldsymbol{\theta}]]$. We cannot compute this exactly, but we can approximate it by drawing J independent samples $\mathbf{x}^*$ from the distribution to yield

$$\frac{\partial \log[Z(\boldsymbol{\theta})]}{\partial \boldsymbol{\theta}} = \sum_{\mathbf{x}} Pr(\mathbf{x}) \frac{\partial \log[f[\mathbf{x}, \boldsymbol{\theta}]]}{\partial \boldsymbol{\theta}} \approx \frac{1}{J} \sum_{j=1}^{J} \frac{\partial \log[f[\mathbf{x}_j^*, \boldsymbol{\theta}]]}{\partial \boldsymbol{\theta}}. \tag{10.30}$$

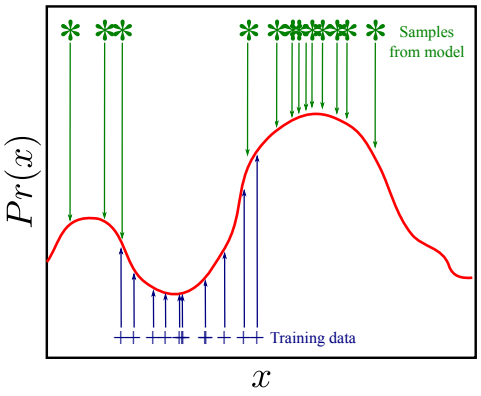


Figure 10.13 The contrastive divergence algorithm changes the parameters so that the un-normalized distribution increases at the observed data points (blue crosses) but decreases at sampled data points from the model. These two components counterbalance one another and ensure that the likelihood increases. When the model fits the data these two forces will cancel out and the parameters will remain constant.

With I training examples $\{\mathbf{x}_i\}_{i=1}^{I}$, the gradient of the log likelihood L is hence

$$\frac{\partial L}{\partial \boldsymbol{\theta}} \approx -\frac{I}{J}\sum_{j=1}^{J}\frac{\partial \log[f(\mathbf{x}_j^*, \boldsymbol{\theta})]}{\partial \boldsymbol{\theta}} + \sum_{i=1}^{I}\frac{\partial \log[f(\mathbf{x}_i, \boldsymbol{\theta})]}{\partial \boldsymbol{\theta}}. \tag{10.31}$$

A visual explanation of this expression is presented in figure 10.13. The gradient points in a direction that (i) increases the logarithm of the un-normalized function at the data points $\mathbf{x}_i$ but (ii) decreases the same quantity in places where the model believes the density is high (i.e., the samples $\mathbf{x}_j^*$). When the model fits the data, these two forces cancel out, and the parameters will stop changing.

This algorithm requires us to draw samples $\mathbf{x}^*$ from the model at each iteration of the optimization procedure in order to compute the gradient. Unfortunately, the only way to draw samples from general undirected graphical models is to use costly Markov chain Monte Carlo methods such as Gibbs sampling (section 10.7.2), and this is impractically time consuming. In practice it has been found that even approximate samples will do: one method is to re-start $J = I$ samples at the data points at each iteration and do just a few MCMC steps. Surprisingly, this works well even with a single step. A second approach is to start with the samples from the previous iteration and perform a few MCMC steps so that the samples are free to wander without restarting. This technique is known as *persistent* contrastive divergence.

Discussion

In this chapter, we introduced directed and undirected graphical models. Each represents a different type of factorization of the joint distribution. A graphical model implies a set of independence and conditional independence relations. There are some sets that can only be represented by directed graphical models, others that can only be represented by undirected graphical models, some that can be represented by both and some that cannot be represented by either.

We presented a number of common vision models and examined their graphical

models. Each had sparse connections and hence many conditional independence relations. In subsequent chapters, we will exploit these redundancies to develop efficient learning and inference algorithms. The world state is usually very high dimensional in these models and so we discussed alternative forms of inference including maximum marginals and sampling.

Finally, we looked at the implications of choosing directed or undirected graphical models for sampling and for learning. We concluded that it is generally more straightforward to draw samples from directed graphical models. Moreover, it is also easier to learn directed graphical models. The best-known learning algorithm for general undirected graphical models requires us to draw samples, which is itself challenging.

Notes

Graphical models: For a readable introduction to graphical models, consult Jordan (2004) or Bishop (2006). For a more comprehensive overview, I would recommend Barber (2012). For an even more encyclopaedic resource, consult Koller & Friedman (2009).

Contrastive divergence: The contrastive divergence algorithm was introduced by Hinton (2002). Further information about this technique can be found in Carreira-Perpiñán. & Hinton (2005) and Bengio & Delalleau (2009).

Problems

Problem 10.1 The joint probability model between variables $\{x_n\}_{n=1}^{7}$ factorizes as

$$Pr(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = Pr(x_1)Pr(x_3)Pr(x_7)Pr(x_2|x_1, x_3)Pr(x_5|x_7, x_2)Pr(x_4|x_2)Pr(x_6|x_5, x_4).$$

Draw a directed graphical model relating these variables. Which variables form the Markov blanket of variable x_2?

Problem 10.2 Write out the factorization corresponding to the directed graphical model in figure 10.14a.

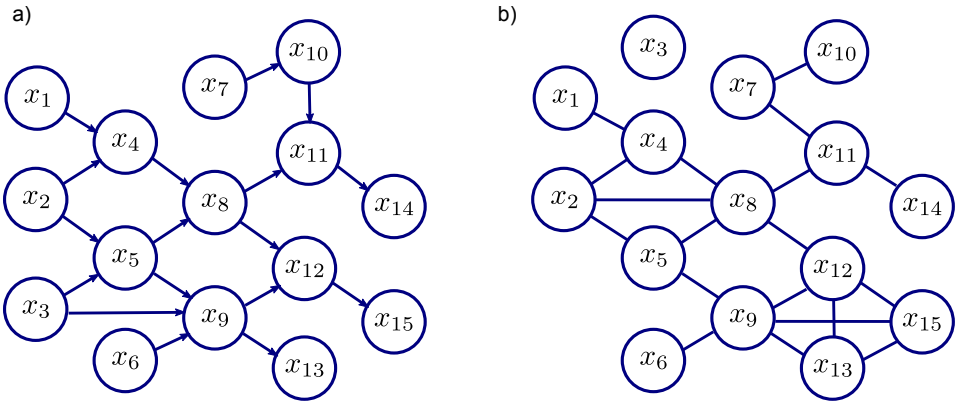


Figure 10.14 a) Graphical model for problem 10.2. b) Graphical model for problem 10.4

Problem 10.3 An undirected graphical model has the form

$$Pr(x_1 \ldots x_6) = \frac{1}{Z}\Phi_1[x_1, x_2, x_5]\Phi_2[x_2, x_3, x_4]\Phi_3[x_1 x_5]\Phi_4[x_5, x_6].$$

Draw the undirected graphical model that corresponds to this factorization.

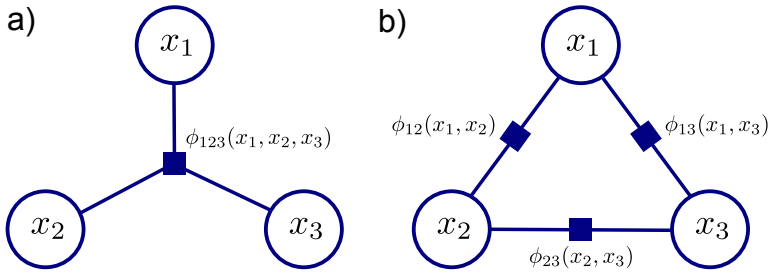


Figure 10.15 Factor graphs contain one node (square) per factor in the joint pdf as well as one node (circle) per variable. Each factor node is connected to all of the variables that belong to that factor. This type of graphical model can distinguish between the undirected graphical models a) $Pr(x_1, x_2, x_3) = \frac{1}{Z}\phi_{123}[x_1, x_2, x_3]$ and b) $Pr(x_1, x_2, x_3) = \frac{1}{Z}\phi_{12}[x_1, x_2]\phi_{23}[x_2, x_3]\phi_{13}[x_1, x_3]$.

Problem 10.4 Write out the factorization corresponding to the undirected graphical model in figure 10.14b.

Problem 10.5 Consider the undirected graphical model defined over binary values $\{x_i\}_{i=1}^{4} \in \{0, 1\}$ defined by

$$Pr(x_1, x_2, x_3, x_4) = \frac{1}{Z}\phi(x_1, x_2)\phi(x_2, x_3)\phi(x_3, x_4)\phi(x_4, x_1),$$

where the function ϕ is defined by

$$\begin{aligned} \phi(0,0) &= 1 & \phi(1,1) &= 2 \\ \phi(0,1) &= 0.1 & \phi(1,0) &= 0.1 \end{aligned}$$

Compute the probability of each of the 16 possible states of this system.

Problem 10.6 What is the Markov blanket for each of the variables in figures 10.7 and 10.8?

Problem 10.7 Show that the stated patterns of independence and conditional independence in figure 10.7 and figure 10.8 are true.

Problem 10.8 A *factor graph* is a third type of graphical model that depicts the factorization of a joint probability. As usual it contains a single node per variable, but it also contains one node per factor (usually indicated by a solid square). Each factor variable is connected to all of the variables that are contained in the associated term in the factorization by undirected links. For example, the factor node corresponding to the term $Pr(x_1|x_2, x_3)$ in a directed model would connect to all three variables x_1, x_2 and x_3. Similarly, the factor node corresponding to the term $\phi_{12}[x_1, x_2]$ in an undirected model would connect variables x_1 and x_2. Figure 10.15 shows two examples of factor graphs.

Draw the factor graphs corresponding to the graphical models in figures 10.7 and 10.8. You must first establish the factorized joint distribution associated with each graph.

Problem 10.9 What is the Markov blanket of variable w_2 in figure 10.9c?

Problem 10.10 What is the Markov blanket of variable w_8 in figure 10.9e?

Chapter 11

Models for chains and trees

In this chapter, we model the relationship between a multidimensional set of measurements $\{\mathbf{x}_n\}_{n=1}^N$ and an associated multidimensional world state $\{w_n\}_{n=1}^N$. When N is large, it is not practical to describe the full set of dependencies between all of these variables, as the number of model parameters will be too great. Instead, we construct models where we only directly describe the probabilistic dependence between variables in small neighborhoods. In particular, we will consider models in which the world variables $\{w_n\}_{n=1}^N$ are structured as chains or trees.

We define a *chain model* to be one in which the world state variables $\{w_n\}_{n=1}^N$ are connected to only the previous variable and the subsequent variable in the associated graphical model (as in figure 11.2). We define a *tree model* to be one in which the world variables have more complex connections, but so that there are no *loops* in the resulting graphical model. Importantly, we disregard the directionality of the connections when we assess whether a directed model is a tree. Hence, our definition of a tree differs from the standard computer science usage.

We will also make the following assumptions:

- The world states w_n are discrete.
- There is an observed data variable $\mathbf{x}_n$ associated with each world state w_n.
- The n^{th} data variable $\mathbf{x}_n$ is conditionally independent of all other data variables and world states given the associated world state w_n.

These assumptions are not critical for the development of the ideas in this chapter, but are typical for the type of computer vision applications that we consider. We will show that both maximum a posteriori and maximum marginals inference are tractable for this sub-class of models, and we will discuss why this is not the case when the states are not organized as a chain or a tree.

To motivate these models, consider the problem of *gesture tracking*. Here, the goal is to automatically interpret sign language from a video sequence (figure 11.1). We observe N frames $\{\mathbf{x}_n\}_{n=1}^N$ of a video sequence and wish to infer the N discrete variables $\{w_n\}_{n=1}^N$ that encode which sign is present in each of the N frames. The data at time n tells us something about the sign at time n but may be insufficient to specify it accurately. Consequently, we also model dependencies between adjacent

Figure 11.1 Interpreting sign language. We observe a sequence of images of a person using sign language. In each frame we extract a vector $\mathbf{x}_n$ describing the shape and position of the hands. The goal is to infer the sign w_n that is present. Unfortunately, the visual data in a single frame may be ambiguous. We improve matters by describing probabilistic connections between adjacent states w_n and w_{n-1}. We impose knowledge about the likely sequence of signs, and this helps disambiguate any individual frame. Images from Purdue RVL-SLLL ASL database (Wilbur & Kak 2006).

world states: we know that the signs are more likely to appear in some orders than others and we exploit this knowledge to help disambiguate the sequence. Since we model probabilistic connections only between adjacent states in the time series, this has the form of a chain model.

11.1 Models for chains

In this section, we will describe both a directed and an undirected model for describing chain structure and show that these two models are equivalent.

11.1.1 Directed model for chains

The directed model describes the joint probability of a set of continuous measurements $\{\mathbf{x}_n\}_{n=1}^N$ and a set of discrete world states $\{w_n\}_{n=1}^N$ with the graphical model shown in figure 11.2a. The tendency to observe the measurements $\mathbf{x}_n$ given that state w_n takes value k is encoded in the likelihood $Pr(\mathbf{x}_n|w_n = k)$. The prior probability of the first state w_1 is explicitly encoded in the discrete distribution $Pr(w_1)$, but, for simplicity, we assume that this is uniform and omit it from most of the ensuing discussion. The remaining states are each dependent on the previous one, and this information is captured in the distribution $Pr(w_n|w_{n-1})$. This is sometimes termed the *Markov assumption*.

Hence, the overall joint probability is

$$Pr(\mathbf{x}_{1\ldots N}, w_{1\ldots N}) = \left(\prod_{n=1}^{N} Pr(\mathbf{x}_n|w_n)\right)\left(\prod_{n=2}^{N} Pr(w_n|w_{n-1})\right). \tag{11.1}$$

This is known as a *hidden Markov model* (HMM). The world states $\{w_n\}_{n=1}^N$ in the directed model have the form of a chain, and the overall model has the form of a tree. As we shall see, these properties are critical to our ability to perform inference.

11.1.2 Undirected model for chains

The undirected model (see section 10.3) describes the joint probability of the measurements $\{\mathbf{x}_n\}_{n=1}^N$ and the world states $\{w_n\}_{n=1}^N$ with the graphical model shown in figure 11.2b. The tendency for the measurements and the data to take certain values is encoded in the potential function $\phi[\mathbf{x}_n, w_n]$. This function always returns positive values and returns larger values when the measurements and the world state are more compatible. The tendency for adjacent states to take certain values is encoded in a second potential function $\zeta[w_n, w_{n-1}]$ which returns larger values when the adjacent states are more compatible. Hence, the overall probability is

$$Pr(\mathbf{x}_{1\ldots N}, w_{1\ldots N}) = \frac{1}{Z}\left(\prod_{n=1}^{N} \phi[\mathbf{x}_n, w_n]\right)\left(\prod_{n=2}^{N} \zeta[w_n, w_{n-1}]\right) \tag{11.2}$$

Once more, the states form a chain and the overall model has the form of a tree; there are no loops.

11.1.3 Equivalence of models

When we take a directed model and make the edges undirected, we usually create a different model. However, comparing equations 11.1 and 11.2 reveals that these two models represent the same factorization of the joint probability density; in this special case, the two models are equivalent. This equivalence becomes even more apparent if we make the substitutions

$$\begin{aligned} Pr(\mathbf{x}_n|w_n) &= \frac{1}{z_n}\phi[\mathbf{x}_n, w_n] \\ Pr(w_n|w_{n-1}) &= \frac{1}{z'_n}\zeta[w_n, w_{n-1}], \end{aligned} \tag{11.3}$$

where z_n and z'_n are normalizing factors which form the partition function:

$$Z = \left(\prod_{n=1}^{N} z_n\right)\left(\prod_{n=2}^{N} z'_n\right). \tag{11.4}$$

Since the directed and undirected versions of the chain model are equivalent, we will continue our discussion in terms of the directed model alone.

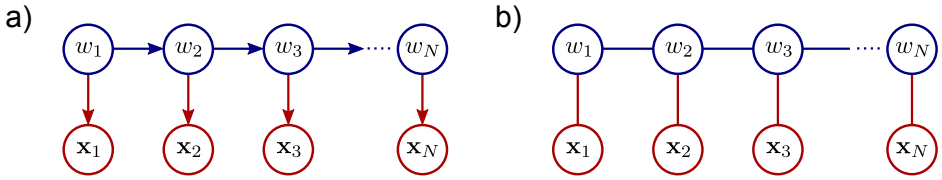


Figure 11.2 Models for chains. a) Directed model. There is one observation variable $\mathbf{x}_n$ for each state variable w_n, and these are related by the conditional probability $Pr(\mathbf{x}_n|w_n)$ (downward arrows). Each state w_n is related to the previous one by the conditional probability $Pr(w_n|w_{n-1})$ (horizontal arrows). b) Undirected model. Here, each observed variable $\mathbf{x}_n$ is related to its associated state variable w_n via the potential function $\phi[\mathbf{x}_n, w_n]$ and the neighboring states are connected via the potential function $\zeta[w_n, w_{n-1}]$.

11.1.4 Hidden Markov model for sign language application

We will now briefly describe how this directed model relates to the sign language application. We preprocess the video frame to create a vector $\mathbf{x}_n$ that represents the shape of the hands. For example, we might just extract a window of pixels around each hand and concatenate their RGB pixel values.

We now model the likelihood $Pr(\mathbf{x}_n|w_n = k)$ of observing this measurement vector given that the sign w_n in this image takes value k. A very simple model might assume that the measurements have a normal distribution with parameters that are contingent on which sign is present so that

$$Pr(\mathbf{x}_n|w_n=k) = \text{Norm}_{\mathbf{x}_n}[\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k]. \tag{11.5}$$

We model the sign w_n as a being categorically distributed, where the parameters depend on the previous sign w_{n-1}, so that

$$Pr(w_n|w_{n-1}=k) = \text{Cat}_{w_n}[\boldsymbol{\lambda}_k]. \tag{11.6}$$

This hidden Markov model has parameters $\{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \boldsymbol{\lambda}_k\}_{k=1}^K$. For most of this chapter, we will assume that these parameters are known, but we return briefly to the issue of learning in section 11.6. We now turn our focus to inference in this type of model.

11.2 MAP inference for chains

Consider a chain with N unknown variables $\{w_n\}_{n=1}^N$, each of which can take K possible values. Here, there are K^N possible states of the world. For real world problems, this means that there are far too many states to evaluate exhaustively;

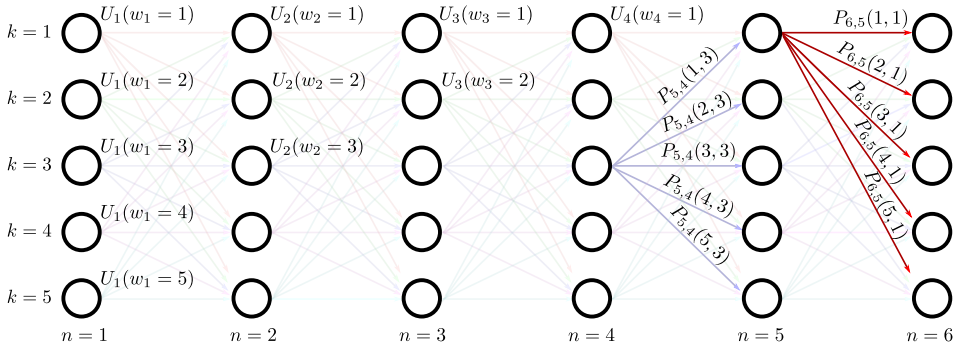


Figure 11.3 Dynamic programming formulation. Each solution is equated with a particular path from left-to-right through an acyclic directed graph. The N columns of the graph represent variables $w_{1\ldots N}$, and the K rows represent possible states $1 \ldots K$. The nodes and edges of the graph have costs associated with the unary and pairwise terms, respectively. Any path from left to right through the graph has a cost that is the sum of the costs at all of the nodes and edges that it passes through. Optimizing the function is now equivalent to finding the path with the least cost.

we can neither compute the full posterior distribution nor search directly through all of the states to find the maximum a posteriori (MAP) estimate.

Fortunately, the factorization of the joint probability distribution (the conditional independence structure) can be exploited to find more efficient algorithms for MAP inference than brute force search. The MAP solution is given by

$$\begin{aligned}
\hat{w}_{1\ldots N} &= \underset{w_{1\ldots N}}{\operatorname{argmax}} \left[Pr(w_{1\ldots N}|\mathbf{x}_{1\ldots N}) \right] \\
&= \underset{w_{1\ldots N}}{\operatorname{argmax}} \left[Pr(\mathbf{x}_{1\ldots N}, w_{1\ldots N}) \right] \\
&= \underset{w_{1\ldots N}}{\operatorname{argmin}} \left[-\log \left[Pr(\mathbf{x}_{1\ldots N}, w_{1\ldots N}) \right] \right], \qquad (11.7)
\end{aligned}$$

where line 2 follows from Bayes' rule. We have reformulated this as a minimization problem in line 3.

Substituting in the expression for the log probability (equation 11.1) we get

$$\hat{w}_{1\ldots N} = \underset{w_{1\ldots N}}{\operatorname{argmin}} \left[-\sum_{n=1}^{N} \log \left[Pr(\mathbf{x}_n|w_n) \right] - \sum_{n=2}^{N} \log \left[Pr(w_n|w_{n-1}) \right] \right], \qquad (11.8)$$

which has the general form

$$\hat{w}_{1\ldots N} = \underset{w_{1\ldots N}}{\operatorname{argmin}} \left[\sum_{n=1}^{N} U_n(w_n) + \sum_{n=2}^{N} P_n(w_n, w_{n-1}) \right], \qquad (11.9)$$

where U_n is a *unary* term and depends only on a single variable w_n and P_n is a *pairwise* term, depending on two variables w_n and w_{n-1}. In this instance, the unary and pairwise terms can be defined as

$$\begin{aligned} U_n(w_n) &= -\log[Pr(\mathbf{x}_n|w_n)] \\ P_n(w_n, w_{n-1}) &= -\log[Pr(w_n|w_{n-1})]. \end{aligned} \tag{11.10}$$

Any problem that has the form of equation 11.9 can be solved in polynomial time using the *Viterbi algorithm* which is an example of *dynamic programming*.

11.2.1 Dynamic programming (Viterbi algorithm)

Algorithm 11.1

To optimize the cost function in equation 11.9, we first visualize the problem with a 2D graph with vertices $\{V_{n,k}\}_{n=1,k=1}^{N,K}$. The vertex $V_{n,k}$ represents choosing the k^{th} world state at the n^{th} variable (figure 11.3). Vertex $V_{n,k}$ is connected by a directed edge to each of the vertices $\{V_{n+1,k}\}_{k=1}^{K}$ at the next pixel position. Hence, the organization of the graph is such that each valid horizontal path from left to right represents a possible solution to the problem; it corresponds to assigning one value $k \in [1 \ldots K]$ to each variable w_n.

We now attach the costs $U_n(w_n = k)$ to the vertices $V_{n,k}$. We also attach the costs $P_n(w_n = k, w_{n-1} = l)$ to the edges joining vertices $V_{n-1,l}$ to $V_{n,k}$. We define the total cost of a path from left to right as the sum of the costs of the edges and vertices that make up the path. Now, every horizontal path represents a solution and the cost of that path is the cost for that solution; we have reformulated the problem as finding the minimum cost path from left to right across the graph.

Finding the minimum cost

The approach to finding the minimum cost path is simple. We work through the graph from left to right, computing at each vertex the minimum possible cumulative cost $S_{n,k}$ to arrive at this point *by any route*. When we reach the right hand side, we compare the K values $S_{N,\bullet}$ and choose the minimum. This is the lowest possible cost for traversing the graph. We now retrace the route we took to reach this point, using information that was cached during the forward pass.

The easiest way to understand this method is with a concrete example (figures 11.4-11.5) and the reader is encouraged to scrutinize these figures before continuing.

A more formal description is as follows. Our goal is to assign the minimum possible cumulative cost $S_{n,k}$ for reaching vertex $V_{n,k}$. Starting at the left hand side, we set the first column of vertices to the unary costs for the first variable:

$$S_{1,k} = U_1(w_1 = k). \tag{11.11}$$

The cumulative total $S_{2,k}$ for the k^{th} vertex in the second column should represent the minimum possible cumulative cost to reach this point. To calculate this, we consider the K possible predecessors, and compute the cost for reaching this vertex by each possible route. We set $S_{2,k}$ to the minimum of these values and store the route by which we reached this vertex so that

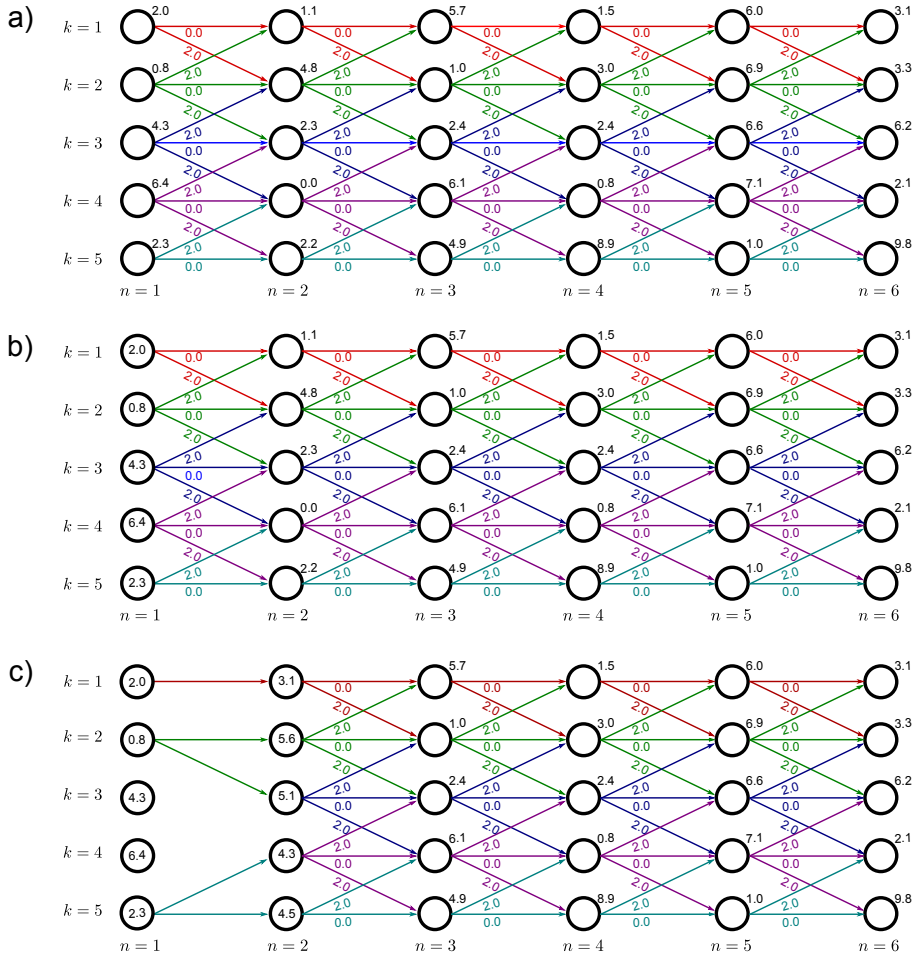


Figure 11.4 Dynamic programming. a) The unary cost $U_n(w_n = k)$ is given by the number above and to the right of each node. The pairwise costs $P_n(w_n, w_{n-1})$ are zero if $w_n = w_{n-1}$ (horizontal), two if $|w_n - w_{n-1}| = 1$ and ∞ otherwise. This favors a solution that is mostly constant but can also vary smoothly. For clarity, we have removed the edges with infinite cost as they cannot become part of the solution. We now work from left to right, computing the minimum cost $S_{n,k}$ for arriving at vertex $V_{n,k}$ by any route. b) For vertices $\{V_{1,k}\}_{k=1}^{K}$, the minimum cost is just the unary cost associated with that vertex. We have stored the values $S_{1,1} \ldots S_{1,5}$ inside the circle representing the respective vertex. c) To compute the minimum cost $S_{2,1}$ at vertex $(n = 2, k = 1)$ we must consider two possible routes. The path could have traveled horizontally from vertex $(1, 1)$ giving a total cost of $2.0 + 0.0 + 1.1 = 3.1$ or it may have come diagonally upward from vertex $(1, 2)$ with cost $0.8 + 2.0 + 1.1 = 3.9$. Since the former route is cheaper, we use this cost, storing $S_{2,1} = 3.1$ at the vertex, and also remembering the path used to get here. Now, we repeat this procedure at vertex $(2, 2)$ where there are three possible routes from vertices $(1, 1)$, $(1, 2)$ and $(1, 3)$. Here it turns out that the best route is from $(1, 2)$ and has total cumulative cost of $S_{2,2} = 5.6$. Example continued in figure 11.5.

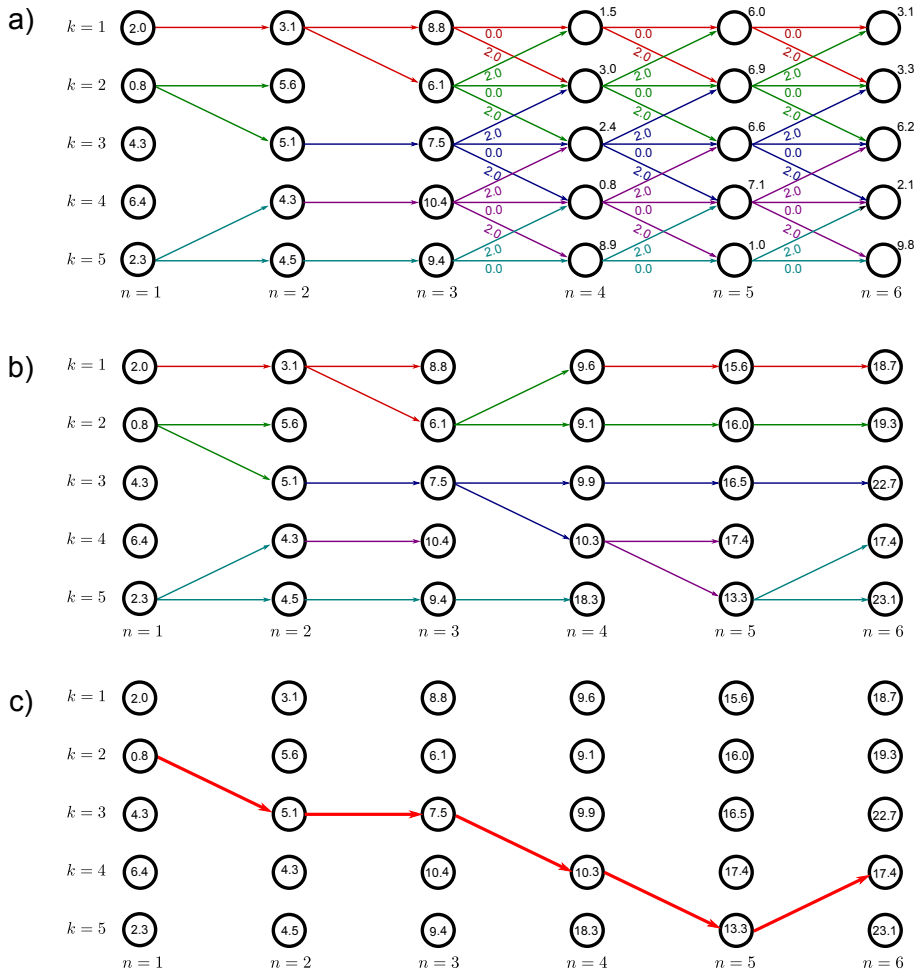


Figure 11.5 Dynamic programming worked example (continued from figure 11.4). a) Having updated the vertices at pixel $n = 2$, we carry out the same procedure at pixel $n = 3$, accumulating at each vertex the minimum total cost to reach this point. b) We continue updating the minimum cumulative costs $S_{n,k}$ to arrive at pixel n in state k until we reach the right hand side. c) We identify the minimum cost from among the right-most vertices. In this case, it is vertex (6,4), which has cost $S_{6,4} = 17.4$. This is the minimum possible cost for traversing the graph. By tracing back the route that we used to arrive here (red arrows), we find the world state at each pixel that was responsible for this cost.

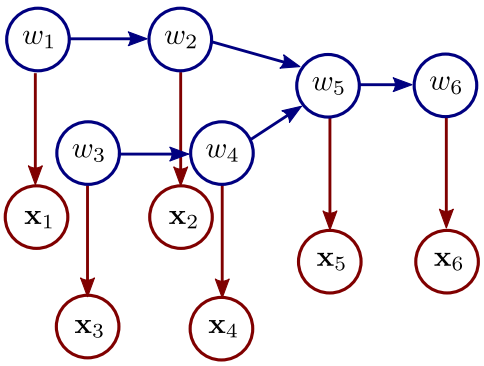


Figure 11.6 Tree-based models. As before, there is one observation $\mathbf{x}_n$ for each world state w_n and these are related by the conditional probability $Pr(\mathbf{x}_n|w_n)$. However, disregarding the directionality of the edges, the world states are now connected as a tree. Vertex w_5 has two incoming connections, which means that there is a 'three-wise' term $Pr(w_5|w_2, w_4)$ in the factorization. The tree structure means it is possible to perform MAP and max-marginals inference efficiently.

$$S_{2,k} = U_2(w_2 = k) + \min_l \left[S_{1,l} + P_2(w_2 = k, w_1 = l)\right]. \tag{11.12}$$

More generally, to calculate the cumulative totals $S_{n,k}$, we use the recursion

$$S_{n,k} = U_n(w_n = k) + \min_l \left[S_{n-1,l} + P_n(w_n = k, w_{n-1} = l)\right], \tag{11.13}$$

and we also cache the route by which this minimum was achieved at each stage. When we reach the right hand side, we find the value of the final variable w_n that minimizes the total cost

$$\hat{w}_N = \operatorname*{argmin}_k \left[S_{N,k}\right], \tag{11.14}$$

and set the remaining labels $\hat{w}_{1\ldots N-1}$ according to the route that we followed to get to this value.

This method exploits the factorization structure of the joint probability between the observations and the states to make vast computational savings. The cost of this procedure is $\mathcal{O}(NK^2)$, as opposed to $\mathcal{O}(K^N)$ for a brute force search through every possible solution.

Problem 11.1
Problem 11.2

11.3 MAP inference for trees

To show how MAP inference works in tree-structured models, consider the model in figure 11.6. For this graph, the prior probability over the states factorizes as

Algorithm 11.2

Problem 11.3
Problem 11.4

$$Pr(w_{1\ldots6}) = Pr(w_1)Pr(w_3)Pr(w_2|w_1)Pr(w_4|w_3)Pr(w_5|w_2, w_4)Pr(w_6|w_5), \tag{11.15}$$

and the world states have the structure of a tree (disregarding the directionality of the edges).

Once more, we can exploit this factorization to compute the MAP solution efficiently. Our goal is to find

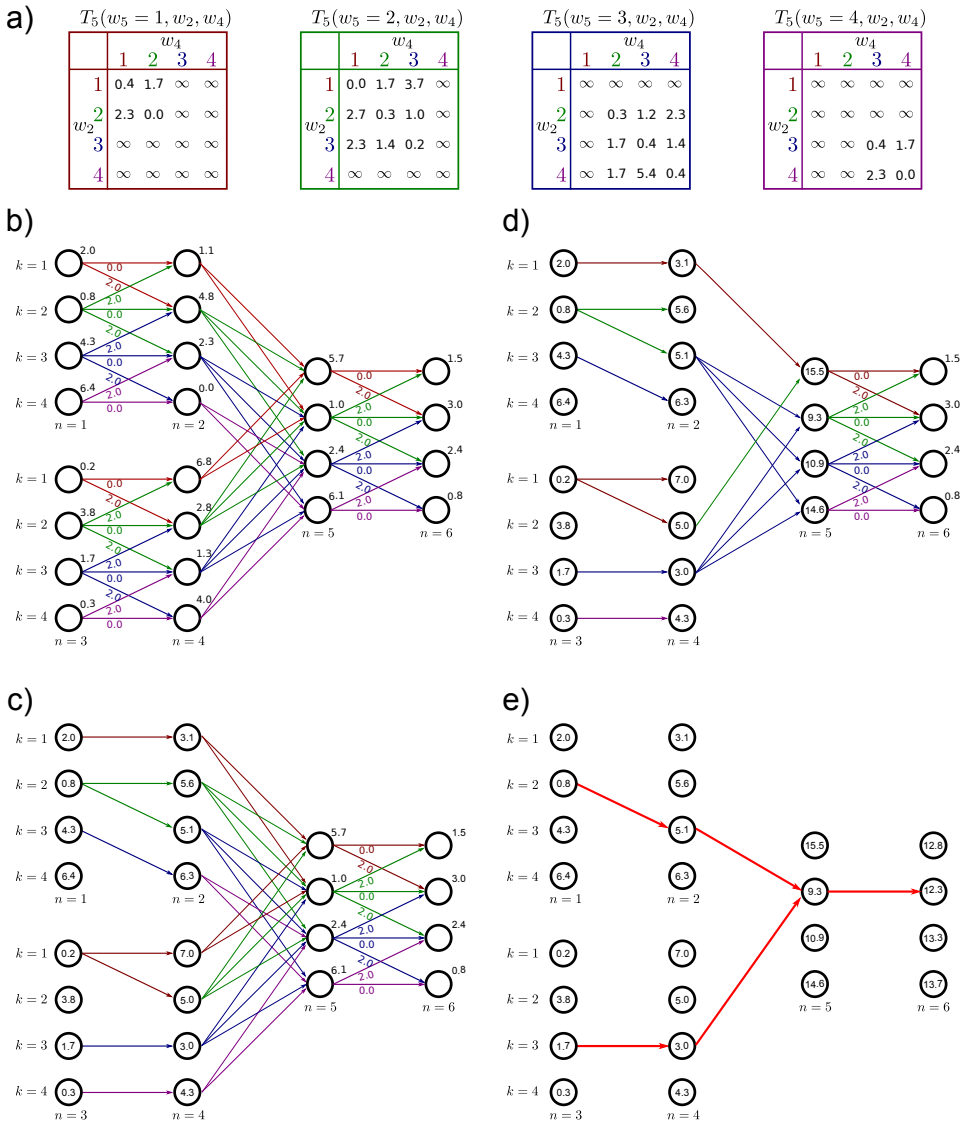


Figure 11.7 Dynamic programming example for tree model in figure 11.6. a) Table of three-wise costs at vertex 5. This is a $K \times K \times K$ table consisting of the costs associated with $Pr(w_5|w_2, w_4)$. Pairwise costs are as for the example in figure 11.4. b) Tree-structured model with unary and pairwise costs attached. c) We work from the leaves, finding the minimal possible cost $S_{n,k}$ to reach vertex n in state k as in the original dynamic programming formulation. d) When we reach the vertex above a branch (here, vertex 5), we find the minimal possible cost, considering every combination of the incoming states. e) We continue until we reach the root. There, we find the minimum overall cost and trace back, making sure to split at the junction according to which pair of states was chosen.

$$\hat{w}_{1\ldots6} = \underset{w_{1\ldots6}}{\operatorname{argmax}} \left[\sum_{n=1}^{6} \log[Pr(\mathbf{x}_n|w_n)] + \log[Pr(w_{1\ldots6})] \right]. \tag{11.16}$$

By a similar process to that in section 11.2, we can rewrite this as a minimization problem with the following cost function:

$$\hat{w}_{1\ldots6} = \underset{w_{1\ldots6}}{\operatorname{argmin}} \left[\sum_{n=1}^{6} U_n(w_n) + P_2(w_2, w_1) + P_4(w_4, w_3) + P_6(w_6, w_5) + T_5(w_5, w_2, w_4) \right]. \tag{11.17}$$

As before, we reformulate this cost function in terms of finding a route through a graph (see figure 11.7). The unary costs U_n are associated with each vertex. The pairwise costs P_m are associated with edges between pairs of adjacent vertices. The three-wise cost T_5 is associated with the combination of states at the point where the tree branches. Our goal now is to find the least cost path from all of the leaves simultaneously to the root.

We work from the leaves to the root of the tree, at each stage computing $S_{n,k}$, the cumulative cost for arriving at this vertex (see worked example in figure 11.7). For the first four vertices, we proceed as in standard dynamic programming:

$$\begin{aligned}
S_{1,k} &= U_1(w_1 = k) \\
S_{2,k} &= U_2(w_2 = k) + \min_l \left[S_{1,l} + P_2(w_2 = k, w_1 = l) \right] \\
S_{3,k} &= U_3(w_3 = k) \\
S_{4,k} &= U_4(w_4 = k) + \min_l \left[S_{3,1} + P_4(w_4 = k, w_3 = l) \right].
\end{aligned} \tag{11.18}$$

When we come to the branch in the tree, we try to find the best combination of routes to reach the nodes for variable 5; we must now minimize over both variables to compute the next term. In other words,

$$S_{5,k} = U_5(w_5 = k) + \min_{l,m} \left[S_{2,l} + S_{4,m} + T_5(w_5 = k, w_2 = l, w_4 = m) \right]. \tag{11.19}$$

Finally, we compute the last terms as normal, so that

$$S_{6,k} = U_6(w_6 = k) + \min_l \left[S_{5,l} + P_6(w_6 = k, w_5 = l) \right]. \tag{11.20}$$

Now we find the world state associated with the minimum of this final sum, and trace back the route that we came by as before, splitting the route appropriately at junctions in the tree.

Dynamic programming in this tree has a greater computational complexity than dynamic programming in a chain with the same number of variables, as we must minimize over two variables at the junction in the tree. The overall complexity is proportional to K^W, where W is the maximum number of variables over which we must minimize.

For directed models, W is equal to the largest number of incoming connections at any vertex. For undirected models, W will be the size of the largest clique. It should be noted that for undirected models, the critical property that allows dynamic programming solutions is that the cliques themselves form a tree (see figure 11.11).

11.4 Marginal posterior inference for chains

In section 11.2, we demonstrated that it is possible to perform MAP inference in chain models efficiently using dynamic programming. In this section, we will consider a different form of inference: we will aim to calculate the marginal distribution $Pr(w_n|\mathbf{x}_{1\ldots N})$ over each state variable w_n separately.

Consider computing the marginal distribution over the variable w_N. By Bayes' rule we have

$$Pr(w_N|\mathbf{x}_{1\ldots N}) = \frac{Pr(w_N, \mathbf{x}_{1\ldots N})}{Pr(\mathbf{x}_{1\ldots N})} \propto \Pr(w_N, \mathbf{x}_{1\ldots N}). \tag{11.21}$$

The right hand side of this equation is computed by marginalizing over all of the other state variables except w_N so we have

$$\begin{aligned} Pr(w_N, \mathbf{x}_{1\ldots N}) &\propto \sum_{w_1}\sum_{w_2}\cdots\sum_{w_{N-1}} Pr(w_{1\ldots N}, \mathbf{x}_{1\ldots N}) \qquad (11.22)\\ &\propto \sum_{w_1}\sum_{w_2}\cdots\sum_{w_{N-1}} \left(\prod_{n=1}^{N} Pr(\mathbf{x}_n|w_n)\right) Pr(w_1) \left(\prod_{n=2}^{N} Pr(w_n|w_{n-1})\right). \end{aligned}$$

Unfortunately, in its most basic form, this marginalization involves summing over $N-1$ dimensions of the N dimensional probability distribution. Since this discrete probability distribution contains K^N entries, computing this summation directly is not practical for realistic sized problems. To make progress, we must again exploit the structured factorization of this distribution.

11.4.1 Computing one marginal distribution

We will first discuss how to compute the marginal distribution $Pr(w_N|\mathbf{x}_{1\ldots N})$ for the last variable in the chain w_N. In the following section, we will exploit these ideas to compute all of the marginal distributions $Pr(w_n|\mathbf{x}_{1\ldots N})$ simultaneously.

We observe that not every term in the product in equation 11.22 is relevant to every summation. We can re-arrange the summation terms so that only the variables over which they sum are to the right

$$Pr(w_N|\mathbf{x}_{1\ldots N}) \propto Pr(\mathbf{x}_N|w_N)\sum_{w_{N-1}}\ldots\sum_{w_2}Pr(w_3|w_2)Pr(\mathbf{x}_2|w_2)\sum_{w_1}Pr(w_2|w_1)Pr(\mathbf{x}_1|w_1)Pr(w_1). \tag{11.23}$$

Then, we proceed from right to left, computing each summation in turn. This technique is known as *variable elimination*. Let us denote the rightmost two terms as

$$\mathbf{f}_1[w_1] = Pr(\mathbf{x}_1|w_1)Pr(w_1). \tag{11.24}$$

Then we sum over w_1 to compute the function

$$\mathbf{f}_2[w_2] = Pr(\mathbf{x}_2|w_2)\sum_{w_1}Pr(w_2|w_1)\mathbf{f}_1[w_1]. \tag{11.25}$$

At the n^{th} stage we compute

$$\mathbf{f}_n[w_n] = Pr(\mathbf{x}_n|w_n)\sum_{w_{n-1}}Pr(w_n|w_{n-1})\mathbf{f}_{n-1}[w_{n-1}], \tag{11.26}$$

and we repeat this process until we have computed the full expression. We then normalize the result to find the marginal posterior $Pr(w_N|\mathbf{x}_{1\ldots N})$ (equation 11.21).

Problem 11.5

This solution consists of $N-1$ summations over K values; it is much more efficient to compute than explicitly computing all K^N solutions and marginalizing over $N-1$ dimensions.

11.4.2 Forward-backward algorithm

In the previous section, we showed an algorithm that could compute the marginal posterior distribution $Pr(w_N|\mathbf{x}_{1\ldots N})$ for the last world state w_N. It is easy to adapt this method to compute the marginal posterior $Pr(w_n|\mathbf{x}_{1\ldots N})$ over any other variable w_n. However, we usually want all of the marginal distributions, and it is inefficient to compute each separately as much of the effort is replicated. The goal of this section is to develop a single procedure that computes the marginal posteriors for *all* of the variables simultaneously and efficiently using a technique known as the *forward-backward algorithm*.

Problem 11.6
Problem 11.7

Algorithm 11.3

The principle is to decompose the marginal posterior into two terms

$$\begin{aligned} Pr(w_n|\mathbf{x}_{1\ldots N}) &\propto Pr(w_n, \mathbf{x}_{1\ldots N}) \\ &= Pr(w_n, \mathbf{x}_{1\ldots n})Pr(\mathbf{x}_{n+1\ldots N}|w_n, \mathbf{x}_{1\ldots n}) \\ &= Pr(w_n, \mathbf{x}_{1\ldots n})Pr(\mathbf{x}_{n+1\ldots N}|w_n), \end{aligned} \tag{11.27}$$

where the relation between the second and third line is true because $\mathbf{x}_{1\ldots n}$ and $\mathbf{x}_{n+1\ldots N}$ are conditionally independent given w_n (as can be gleaned from figure 11.2). We will now focus on finding efficient ways to calculate each of these two terms.

Forward recursion

Let us consider the first term $Pr(w_n, \mathbf{x}_{1\ldots n})$. We can exploit the recursion

$$\begin{aligned}
&Pr(w_n, \mathbf{x}_{1\ldots n}) \\
&\quad = \sum_{w_{n-1}} Pr(w_n, w_{n-1}, \mathbf{x}_{1\ldots n}) \\
&\quad = \sum_{w_{n-1}} Pr(w_n, \mathbf{x}_n | w_{n-1}, \mathbf{x}_{1\ldots n-1}) Pr(w_{n-1}, \mathbf{x}_{1\ldots n-1}) \\
&\quad = \sum_{w_{n-1}} Pr(\mathbf{x}_n | w_n, w_{n-1}, \mathbf{x}_{1\ldots n-1}) Pr(w_n | w_{n-1}, \mathbf{x}_{1\ldots n-1}) Pr(w_{n-1}, \mathbf{x}_{1\ldots n-1}) \\
&\quad = \sum_{w_{n-1}} Pr(\mathbf{x}_n | w_n) Pr(w_n | w_{n-1}) Pr(w_{n-1}, \mathbf{x}_{1\ldots n-1}), \qquad (11.28)
\end{aligned}$$

where we have again applied the conditional independence relations implied by the graphical model between the last two lines.

The term $Pr(w_n, \mathbf{x}_{1\ldots n})$ is exactly the intermediate function $\mathbf{f}_n[w_n]$ that we calculated in the solution for the single marginal distribution in the previous section; we have reproduced the recursion

$$\mathbf{f}_n[w_n] = Pr(\mathbf{x}_n | w_n) \sum_{w_{n-1}} Pr(w_n | w_{n-1}) \mathbf{f}_{n-1}[w_{n-1}], \qquad (11.29)$$

but this time, we based the argument on conditional independence rather than the factorization of the probability distribution. Using this recursion, we can efficiently compute the first term of equation 11.27 for all n; in fact, we were already doing this in our solution for the single marginal distribution $Pr(w_N | \mathbf{x}_{1\ldots N})$.

Backward recursion

Now consider the second term $Pr(\mathbf{x}_{n+1\ldots N} | w_n)$ from equation 11.27. Our goal is to develop a recursive relation for this quantity so that we can compute it efficiently for all n. This time the recursion works backwards from the end of the chain to the front, so our goal is to establish an expression for $Pr(\mathbf{x}_{n\ldots N} | w_{n-1})$ in terms of $Pr(\mathbf{x}_{n+1\ldots N} | w_n)$:

$$\begin{aligned}
&Pr(\mathbf{x}_{n\ldots N} | w_{n-1}) \\
&\quad = \sum_{w_n} Pr(\mathbf{x}_{n\ldots N}, w_n | w_{n-1}) \\
&\quad = \sum_{w_n} Pr(\mathbf{x}_{n\ldots N} | w_n, w_{n-1}) Pr(w_n | w_{n-1}) \\
&\quad = \sum_{w_n} Pr(\mathbf{x}_{n+1\ldots N} | \mathbf{x}_n, w_n, w_{n-1}) Pr(\mathbf{x}_n | w_n, w_{n-1}) Pr(w_n | w_{n-1}) \\
&\quad = \sum_{w_n} Pr(\mathbf{x}_{n+1\ldots N} | w_n) Pr(\mathbf{x}_n | w_n) Pr(w_n | w_{n-1}). \qquad (11.30)
\end{aligned}$$

Here we have again applied the conditional independence relations implied by the graphical model between the last two lines. Denoting the probability $Pr(x_{n+1\ldots N}|w_n)$ as $\mathbf{b}_n[w_n]$, we see that we have the recursive relation

$$\mathbf{b}_{n-1}[w_{n-1}] = \sum_{w_n} Pr(\mathbf{x}_n|w_n)Pr(w_n|w_{n-1})\mathbf{b}_n[w_n]. \tag{11.31}$$

We can use this to compute the second term in equation 11.27 efficiently for all n.

Forward-backward algorithm

We can now summarize the forward-backward algorithm to compute the marginal posterior probability distribution for all n. First, we observe (equation 11.27) that the marginal distribution can be computed as

$$Pr(w_n|\mathbf{x}_{1\ldots N}) \propto Pr(w_n, \mathbf{x}_{1\ldots n})Pr(\mathbf{x}_{n+1\ldots N}|w_n) = \mathbf{f}_n[w_n]\mathbf{b}_n[w_n]. \tag{11.32}$$

We recursively compute the forward terms using the relation

$$\mathbf{f}_n[w_n] = Pr(\mathbf{x}_n|w_n)\sum_{w_{n-1}} Pr(w_n|w_{n-1})\mathbf{f}_{n-1}[w_{n-1}], \tag{11.33}$$

where we set $\mathbf{f}_1[w_1] = Pr(\mathbf{x}_1|w_1)Pr(w_1)$. We recursively compute the backward terms using the relation

$$\mathbf{b}_{n-1}[w_{n-1}] = \sum_{w_n} Pr(\mathbf{x}_n|w_n)Pr(w_n|w_{n-1})\mathbf{b}_n[w_n], \tag{11.34}$$

where we set $\mathbf{b}_N[w_N]$ to the constant value $1/K$.

Finally, to compute the n^{th} marginal posterior distribution, we take the product of the associated forward and backward terms and normalize.

11.4.3 Belief propagation

The forward-backward algorithm can be considered a special case of a more general technique called *belief propagation*. Here, the intermediate functions $\mathbf{f}[\bullet]$ and $\mathbf{b}[\bullet]$ are considered as messages that convey information about the variables. In this section, we describe a version of belief propagation known as the *sum-product algorithm*. This does not compute the marginal posteriors any faster than the forward-backward algorithm, but it is much easier to see how to extend it to models based on trees.

The sum-product algorithm operates on a factor graph. A factor graph is a new type of graphical model that makes the factorization of the joint probability more explicit. It is very simple to convert directed and undirected graphical models to factor graphs. As usual, we introduce one node per variable; for example variables w_1, w_2, and w_3 all have a variable node associated with them. We also introduce one *function node* per term in the factorized joint probability distribution; in a directed model this would represent a conditional probability term such as

Problem 11.8

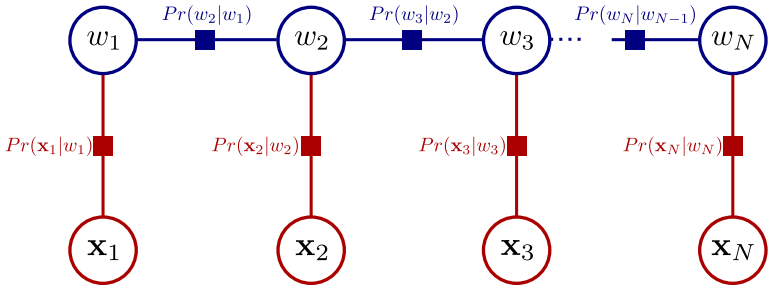


Figure 11.8 Factor graph for chain model. There is one node per variable (circles) and one function node per term in the factorization (squares). Each function node connects to all of the variables associated with this term.

$Pr(w_1|w_2, w_3)$ and in an undirected model it would represent a potential function such as $\phi[w_1, w_2, w_3]$. We then connect each function node to all of the variable nodes relevant to that term with undirected links. So, in a directed model, a term like $Pr(\mathbf{x}_1|w_1, w_2)$ would result in a function node that connects to $\mathbf{x}_1$, w_1 and w_2. In an undirected model, a term like $\phi_{12}(w_1, w_2)$ would result in a function node that connects to w_1 and w_2. Figure 11.8 shows the factor graph for the chain model.

Sum-product algorithm

Algorithm 11.4

The sum product algorithm proceeds in two phases: a forward pass and a backward pass. The forward pass distributes evidence through the graph and the backward pass collates this evidence. Both the distribution and collation of evidence are accomplished by passing messages from node to node in the factor graph. Every edge in the graph is connected to exactly one variable node, and each message is defined over the domain of this variable. There are three types of messages:

1. A message $\mathbf{m}_{\mathbf{z}_p \rightarrow g_q}$ from an unobserved variable $\mathbf{z}_p$ to a function node g_q is given by

$$\mathbf{m}_{\mathbf{z}_p \rightarrow g_q} = \prod_{r \in \text{ne}[p] \setminus q} \mathbf{m}_{g_r \rightarrow \mathbf{z}_p}. \tag{11.35}$$

   where ne[p] returns the set of the neighbors of $\mathbf{z}_p$ in the graph and so the expression ne[p] $\setminus$ q denotes all of the neighbours except q. In other words, the message from a variable to a function node is the pointwise product of all other incoming messages to the variable; it is the combination of other beliefs.

2. A message $\mathbf{m}_{z_p \rightarrow g_q}$ from an observed variable $\mathbf{z}_p = \mathbf{z}_p^*$ to a function node g_q is given by

$$\mathbf{m}_{z_p \rightarrow g_q} = \delta[\mathbf{z}_p^*]. \tag{11.36}$$

In other words, the message from an observed node to a function conveys the certain belief that this node took the observed value.

3. A message $\mathbf{m}_{g_p \rightarrow \mathbf{z}_q}$ from a function node g_p to a recipient variable $\mathbf{z}_q$ is defined as

$$\mathbf{m}_{g_p \rightarrow \mathbf{z}_q} = \sum_{\text{ne}[p] \setminus q} g_p[\text{ne}[p]] \prod_{r \in \text{ne}[p] \setminus q} m_{\mathbf{z}_r \rightarrow g_p}. \tag{11.37}$$

This takes beliefs from all variables connected to the function except the recipient variable and uses the function $g_p[\bullet]$ to convert these to a belief about the recipient variable.

In the forward phase, the message passing can proceed in any order, as long as the outgoing message from any variable or function is not sent until all the other incoming messages have arrived. In the backward pass, the messages are sent in the opposite order to the forward pass.

Finally, the marginal distribution at node $\mathbf{z}_p$ can be computed from a product of all of the incoming messages from both the forward and reverse passes so that

$$Pr(\mathbf{z}_p) \propto \prod_{r \in \text{ne}[p]} \mathbf{m}_{g_r \rightarrow \mathbf{z}_p}. \tag{11.38}$$

A proof that this algorithm is correct is beyond the scope of this book. However, to make this at least partially convincing (and more concrete), we will work through these rules for the case of the chain model (figure 11.8), and we will show that exactly the same computation occurs as for the forward-backward algorithm.

11.4.4 Sum-product algorithm for chain model

The factor graph for the chain solution annotated with messages is shown in figure 11.9. We will now describe the sum-product algorithm for the chain model.

Forward pass

We start by passing a message $\mathbf{m}_{\mathbf{x}_1 \rightarrow g_1}$ from node $\mathbf{x}_1$ to the function node g_1. Using rule 2, this message is a delta function at the observed value $\mathbf{x}_1^*$, so that

$$\mathbf{m}_{\mathbf{x}_1 \rightarrow g_1} = \delta[\mathbf{x}_1^*]. \tag{11.39}$$

Now we pass a message from function g_1 to node w_1. Using rule 3, we have

$$\mathbf{m}_{g_1 \rightarrow w_1} = \int Pr(\mathbf{x}_1|w_1)\delta[\mathbf{x}_1^*]d\mathbf{x}_1 = Pr(\mathbf{x}_1 = \mathbf{x}_1^*|w_1). \tag{11.40}$$

By rule 1, the message from node w_1 to function $g_{1,2}$ is simply the product of the incoming nodes, and since there is only one incoming node, this is just

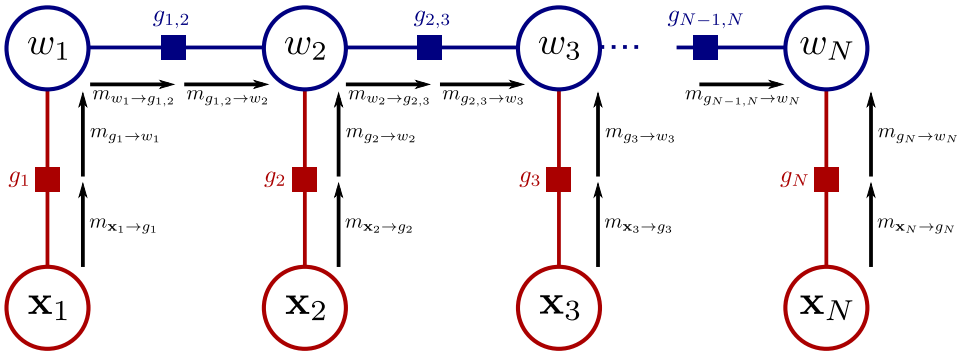


Figure 11.9 Sum product algorithm for chain model (forward pass). The sum-product algorithm has two phases. In the forward phase messages are passed through the graph in an order such that a message cannot be sent until all incoming messages are received at the source node. So, the message $\mathbf{m}_{w_2 \rightarrow g_{23}}$ cannot be sent until the messages $\mathbf{m}_{g_2 \rightarrow w_2}$ and $\mathbf{m}_{g_{1,2} \rightarrow w_2}$ have been received.

$$\mathbf{m}_{w_1 \rightarrow g_{12}} = Pr(\mathbf{x}_1 = \mathbf{x}_1^* | w_1). \tag{11.41}$$

By rule 3, the message from function $g_{1,2}$ to node w_2 is computed as

$$\mathbf{m}_{g_{1,2} \rightarrow w_2} = \sum_{w_1} Pr(w_2|w_1) Pr(\mathbf{x}_1 = \mathbf{x}_1^* | w_1). \tag{11.42}$$

Continuing this process, the messages from $\mathbf{x}_2$ to g_2 and g_2 to w_2 are

$$\begin{aligned} \mathbf{m}_{\mathbf{x}_2 \rightarrow g_2} &= \delta[\mathbf{x}_2^*] \\ \mathbf{m}_{g_2 \rightarrow w_2} &= Pr(\mathbf{x}_2 = \mathbf{x}_2^* | w_2), \end{aligned} \tag{11.43}$$

and the message from w_2 to $g_{2,3}$ is

$$\mathbf{m}_{w_2 \rightarrow g_{2,3}} = Pr(\mathbf{x}_2 = \mathbf{x}_2^* | w_2) \sum_{w_1} Pr(w_2|w_1) Pr(\mathbf{x}_1 = \mathbf{x}_1^* | w_1). \tag{11.44}$$

A clear pattern is emerging; the message from node w_n to function $g_{n,n+1}$ is equal to the forward term from the forward-backward algorithm:

$$\mathbf{m}_{w_n \rightarrow g_{n,n+1}} = \mathbf{f}_n[w_n] = Pr(w_n, \mathbf{x}_{1 \ldots n}). \tag{11.45}$$

In other words, the sum-product algorithm is performing exactly the same computations as the forward pass of the forward-backward algorithm.

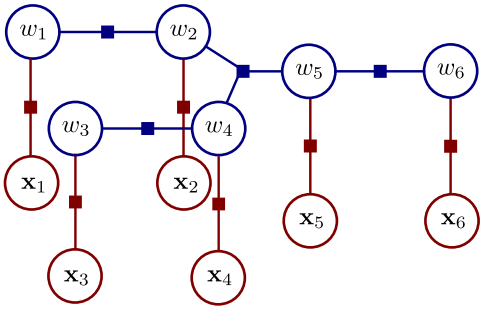


Figure 11.10 Factor graph corresponding to tree model in figure 11.6. There is one function node connecting each world state variable to its associated measurement and these correspond to the terms $Pr(\mathbf{x}_n|w_n)$. There is one function node for each of the three pairwise terms $Pr(w_2|w_1)$, $Pr(w_4|w_3)$ and $Pr(w_6|w_5)$ and this is connected to both contributing variables. The function node corresponding to the three-wise term $Pr(w_5|w_2, w_4)$ has three neighbors, w_5, w_2, and w_4.

Backward pass

When we reach the end of the forward pass of the belief propagation, we initiate the backward pass. There is no need to pass messages toward the observed variables $\mathbf{x}_n$ since we already know their values for certain. Hence, we concentrate on the horizontal connections between the unobserved variables (i.e., along the spine of the model). The message from node w_N to function $g_{N,N-1}$ is given by

$$\mathbf{m}_{w_N \rightarrow g_{N,N-1}} = Pr(\mathbf{x}_N = \mathbf{x}_N^*|w_N), \tag{11.46}$$

and the message from $g_{N,N-1}$ to w_{N-1} is given by

$$\mathbf{m}_{g_{N,N-1} \rightarrow w_{N-1}} = \sum_{w_N} Pr(w_N|w_{N-1}) Pr(\mathbf{x}_N = \mathbf{x}_N^*|w_N). \tag{11.47}$$

In general, we have:

$$\begin{aligned} \mathbf{m}_{g_{n,n-1} \rightarrow w_{n-1}} &= \sum_{w_n} Pr(w_n|w_{n-1}) Pr(\mathbf{x}_n|w_n) \mathbf{m}_{g_{n+1,n} \rightarrow w_n} \\ &= \mathbf{b}_{n-1}[w_{n-1}], \end{aligned} \tag{11.48}$$

which is exactly the backward recursion from the forward-backward algorithm.

Collating evidence

Finally, to compute the marginal probabilities, we use the relation

$$Pr(w_n|\mathbf{x}_{1\ldots N}) \propto \prod_{m \in \text{ne}[n]} \mathbf{m}_{g_m \rightarrow \mathbf{w}_n}, \tag{11.49}$$

and for the general case, this consists of three terms:

$$\begin{aligned} Pr(w_n|\mathbf{x}_{1\ldots N}) &\propto \mathbf{m}_{g_{n-1,n} \rightarrow w_n} \mathbf{m}_{g_n \rightarrow w_n} \mathbf{m}_{g_{n,n+1} \rightarrow w_n} \\ &= \mathbf{m}_{w_n \rightarrow g_{n,n+1}} \mathbf{m}_{g_{n,n+1} \rightarrow w_n} \\ &= \mathbf{f}_n[w_n] \mathbf{b}_n[w_n], \end{aligned} \tag{11.50}$$

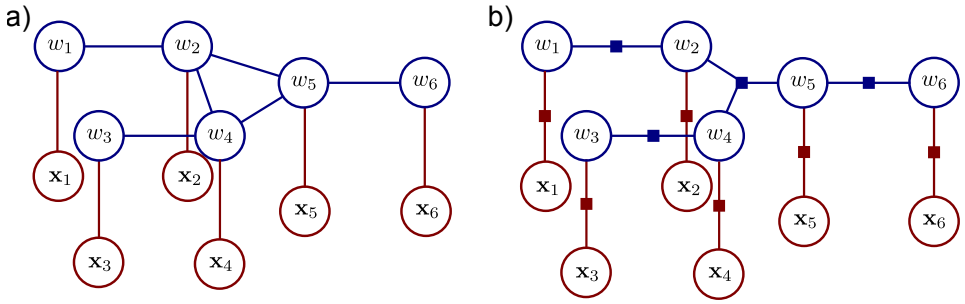


Figure 11.11 Converting an undirected model to a factor graph. a) Undirected model. b) Corresponding factor graph. There is one function node for each maximal clique (each clique which is not a subset of another clique). Although there was clearly a loop in the original graph, there is no loop in the factor graph and so the sum-product algorithm is still applicable.

where in the second line we have used the fact that the outgoing message from a variable node is the product of the incoming messages. We conclude that the sum-product algorithm computes the posterior marginals in exactly the same way as the forward backward algorithm.

11.5 Marginal posterior inference for trees

Problem 11.9

To compute the marginals in tree-structured models we simply apply the sum product algorithm to the new graph structure. The factor graph for the tree in figure 11.6 is shown in figure 11.10. The only slight complication is that we must ensure that the first two incoming messages to the function relating variables 2,4, and 5 have arrived before sending the outgoing message. This is very similar to the order of operations in the dynamic programming algorithm.

For undirected graphs, the key property is that the cliques, not the nodes, form a tree. For example, there is clearly a loop in the undirected model in figure 11.11a, but when we convert this to a factor graph the structure is a tree (figure 11.11b). For models with only pairwise cliques, the cliques always form a tree if there are no loops in the original graphical model.

11.6 Learning in chains and trees

So far, we have only discussed inference for these models. Here, we briefly discuss learning which can be done in a *supervised* or *unsupervised* context. In the supervised case, we are given a training set of I matched sets of states $\{w_{in}\}_{i=1,n=1}^{I,N}$ and

data $\{\mathbf{x}_{in}\}_{i=1,n=1}^{I,N}$. In the unsupervised case, we only observe the data $\{\mathbf{x}_{in}\}_{i=1,n=1}^{I,N}$.

Supervised learning for directed models is relatively simple. We first isolate the part of the model that we want to learn. For example, we might learn the parameters $\boldsymbol{\theta}$ of $Pr(\mathbf{x}_n|w_n,\boldsymbol{\theta})$ from paired examples of $\mathbf{x}_n$ and w_n. We can then learn these parameters in isolation using the ML, MAP, or Bayesian methods.

Unsupervised learning is more challenging; the states w_n are treated as hidden variables and the EM algorithm is applied. In the E-step, we compute the posterior marginals over the states using the forward backward algorithm. In the M-step we use these marginals to update the model parameters. For the hidden Markov model (the chain model), this is known as the *Baum-Welch* algorithm.

As we saw in the previous chapter, learning in undirected models can be challenging; we cannot generally compute the normalization constant Z and this in turn prevents us from computing the derivative with respect to the parameters. However, for the special case of tree and chain models, it is possible to compute Z efficiently and learning is tractable. To see why this is the case, consider the undirected model from figure 11.2b, which we will treat here as representing the conditional distribution

$$Pr(w_{1\ldots N}|\mathbf{x}_{1\ldots N}) = \frac{1}{Z}\left(\prod_{n=1}^{N}\phi[\mathbf{x}_n,w_n]\right)\left(\prod_{n=2}^{N}\zeta[w_n,w_{n-1}]\right), \tag{11.51}$$

since the data nodes $\{\mathbf{x}_n\}_{n=1}^{N}$ are fixed. This model is known as a *1D conditional random field*. The unknown constant Z now has the form

$$Z = \sum_{w_1}\sum_{w_2}\cdots\sum_{w_N}\left(\prod_{n=1}^{N}\phi[\mathbf{x}_n,w_n]\right)\left(\prod_{n=2}^{N}\zeta[w_n,w_{n-1}]\right). \tag{11.52}$$

We have already seen that it is possible to compute this type of sum efficiently, using a recursion (equation 11.26). Hence, Z can be evaluated and maximum likelihood learning can be performed in this model without the need for contrastive divergence.

11.7 Beyond chains and trees

Unfortunately, there are many models in computer vision that do not take the form of a chain or a tree. Of particular importance are models that are structured to have one unknown w_n for each RGB pixel $\mathbf{x}_n$ in the image. These models are naturally structured as *grids* and the world states are each connected to their four pixel neighbors in the graphical model (figure 11.12). Stereo vision, segmentation, de-noising, super-resolution, and many other vision problems can all be framed in this way.

We devote the whole of the next chapter to grid-based problems, but we will briefly take the time to examine why the methods developed in this chapter are not suitable. Consider a simple model based on a 2×2 grid. Figure 11.13 illustrates why

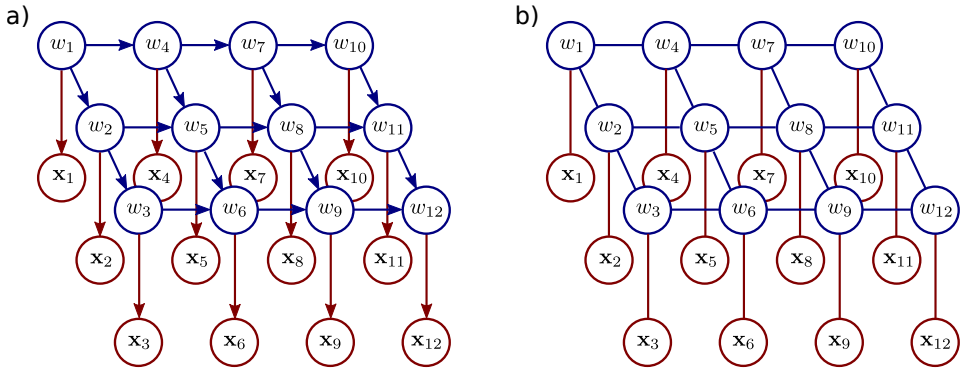
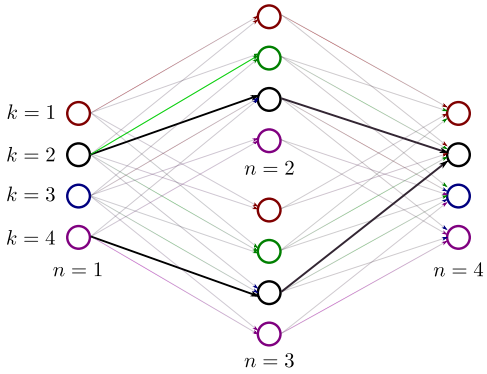


Figure 11.12 Grid-based models. For many vision problems, the natural description is a grid-based model. We observe a grid of pixel values $\{\mathbf{x}_n\}_{n=1}^N$ and wish to infer an unknown world state $\{w_n\}_{n=1}^N$ associated with each site. Each world state is connected to its neighbors. These connections are usually applied to ensure a smooth or piecewise smooth solution. a) Directed grid model. b) Undirected grid model (2D conditional random field).

Figure 11.13 Dynamic programming fails when there are undirected loops. Here, we show a 2×2 image where we have performed a naïive forward pass through the variables. On retracing the route, we see that the two branches disagree over which state the first variable took. For a coherent solution the cumulative minimum costs at node 4, we should have forced the two paths to have common ancestors. With a large number of ancestors this is too computationally expensive to be practical.



we cannot blindly use dynamic programming to compute the MAP solution. To compute the minimum cumulative cost S_n at each node, we might naïvely proceed as normal:

$$\begin{aligned} S_{1,k} &= U_1(w_1 = k) \\ S_{2,k} &= U_2(w_2 = k) + \min_l \left[S_1(w_1 = l) + P_2(w_2 = k, w_1 = l)\right] \\ S_{3,k} &= U_3(w_3 = k) + \min_l \left[S_1(w_1 = l) + P_2(w_3 = k, w_1 = l)\right]. \end{aligned} \quad (11.53)$$

Now consider the fourth term. Unfortunately,

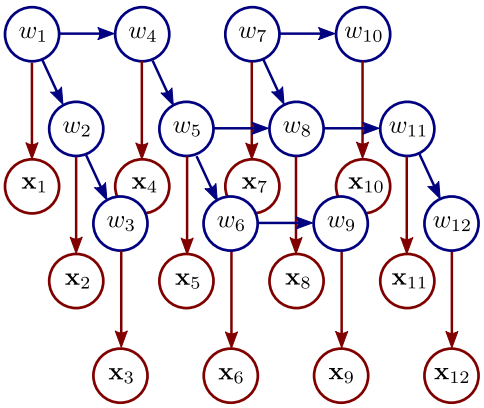


Figure 11.14 Pruning graphs with loops. One approach to dealing with models with loops is simply to prune the connections until the loops are removed. This graphical model is the model from figure 11.12a after such a pruning process. Most of the connections are retained but now the remaining structure is a tree. The usual approach to pruning is to associate a strength with each edge so that weaker edges are more desirable. Then we compute the minimum spanning tree based in these strengths and discard any connections that do not form part of the tree.

$$S_{4,k} \neq U_4(w_k = 4) + \min_{l,m}\left[S_2(w_2 = l) + S_3(w_3 = m) + T(w_4 = k, w_2 = l, w_3 = m)\right]. \tag{11.54}$$

The reason for this is that the partial cumulative sums S_2 and S_3 at the two previous vertices both rely on minimizing over the same variable w_1. However, they did not necessarily choose the same value at w_1. If we were to trace back the paths we took, the two routes back to vertex one might predict a different answer. To properly calculate the minimum cumulative cost at node $S_{4,k}$, we would have to take account of all three ancestors: the recursion is no longer valid, and the problem becomes intractable once more.

Similarly, we cannot perform belief propagation on this graph: the algorithm requires us to send a message from a node only when all other incoming messages have been received. However, the nodes w_1, w_2, w_3, and w_4 all simultaneously require messages from one another and so this is not possible.

11.7.1 Inference in graphs with loops

Although the methods of this chapter are not suitable for models based on graphs with loops, there are a number of ways to proceed:

1. **Prune the graph.** An obvious idea is to prune the graph by removing edges until what is left has a tree structure (figure 11.14). The choice of which edges to prune will depend on the real-world problem.

2. **Combine variables.** A second approach is to combine variables together until what remains has the structure of a chain or tree. For example, in figure 11.15 we combine the variables w_1, w_2, and w_3 to make a new variable w_{123} and the variables w_4, w_5, and w_6 to form w_{456}. Continuing in this way, we form a model that has a chain structure. If each of the original variables had K states, then the compound variables will have K^3 states,

Problem 11.10

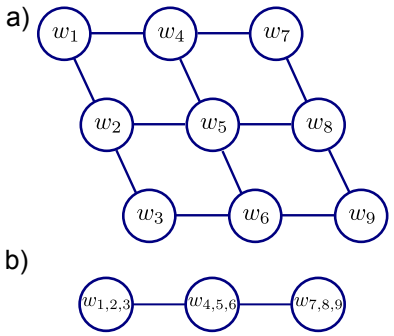


Figure 11.15 Combining variables. a) This graphical model contains loops. b) We form three compound variables, each of which consists of all of the variables in one of the original columns. These are now connected by a chain structure. However, the price we pay is that if there were K states for each original variable, the compound variables now have K^3 states.

and so inference will be more expensive. In general the merging of variables can be automated using the *junction tree* algorithm. Unfortunately, this example illustrates why this approach will not work for large grid models: we must merge together so many variables that the resulting compound variable has too many states to work with.

3. **Loopy belief propagation.** Another idea is to simply apply belief propagation regardless of the loops. All messages are initialized to uniform and then the messages are repeatedly passed in some order according to the normal rules. This algorithm is not guaranteed to converge to the correct solution for the marginals (or indeed to converge at all) but in practice it produces useful results in many situations.

4. **Sampling approaches.** For directed graphical models, it is usually easy to draw samples from the posterior. These can then be aggregated to compute an empirical estimate of the marginal distributions.

5. **Other approaches:** There are several other approaches for exact or approximate inference in graphs, including *tree reweighted message passing* and *graph cuts*. The latter is a particularly important class of algorithm, and we devote most chapter 12 to describing it.

11.8 Applications

The models in this chapter are attractive because they permit exact MAP inference. They have been applied to a number of problems in which there are assumed spatial or temporal connections between parts of a model.

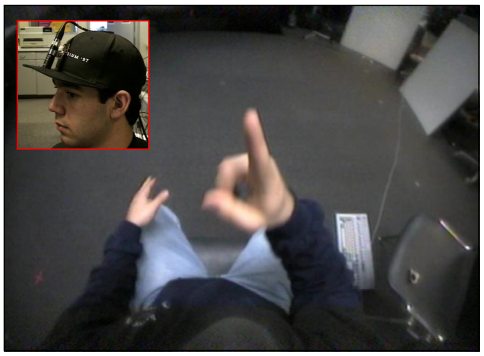


Figure 11.16 Gesture tracking from Starner *et al.* (1998). A camera was mounted on a baseball cap (inset) looking down at the users hands. The camera image (main figure) was used to track the hands in a HMM based system that could accurately classify a 40-word lexicon and worked in real time. Each word was associated with four states in the HMM. The system was based on a compact description of the hand position and orientation within each frame. Adapted from Starner *et al.* (1998). ©1998 Springer.

11.8.1 Gesture tracking

The goal of gesture tracking is to classify the position $\{w_n\}_{n=1}^N$ of the hands within each of the N captured frames from a video sequence into a discrete set of possible gestures, $w_n \in \{1, 2, \ldots, K\}$ based on extracted data $\{\mathbf{x}_n\}_{n=1}^N$ from those frames. Starner *et al.* (1998) presented a wearable system for automatically interpreting sign language gestures. A camera mounted in the user's hat captured a top-down view of their hands (figure 11.16). The positions of the hands were identified by using a per-pixel skin segmentation technique (see section 6.6.1). The state of each hand was characterized in terms of the position and shape of a bounding ellipse around the associated skin region. The final eight-dimensional data vector $\mathbf{x}$ concatenated these measurements from each hand.

To describe the time sequences of these measurements, Starner *et al.* (1998) developed a hidden Markov model-based system in which the states w_n each represented a part of a sign language word. Each of these words was represented by a progression through four values of the state variable w, representing the various stages in the associated gesture for that word. The progression through these states might last any number of time steps (each state can be followed by itself so it can cycle indefinitely) but must come in the required order. They trained the system using 400 training sentences. They used a dynamic programming method to estimate the most likely states w and achieved recognition accuracy of 97.8% using a 40 word lexicon with a test set of 100 sentences. They found that performance was further improved if they imposed knowledge about the fixed grammar of each phrase (pronoun, verb, noun, adjective, pronoun). Remarkably, the system worked at a rate of 10 frames a second.

11.8.2 Stereo vision

In dense stereo vision, we are given two images of the same scene taken from slightly different positions. For our purposes we will assume that they have been preprocessed so that for each pixel in image 1, the corresponding pixel is on the same scanline in image 2 (a process known as *rectification*, see chapter 16). The

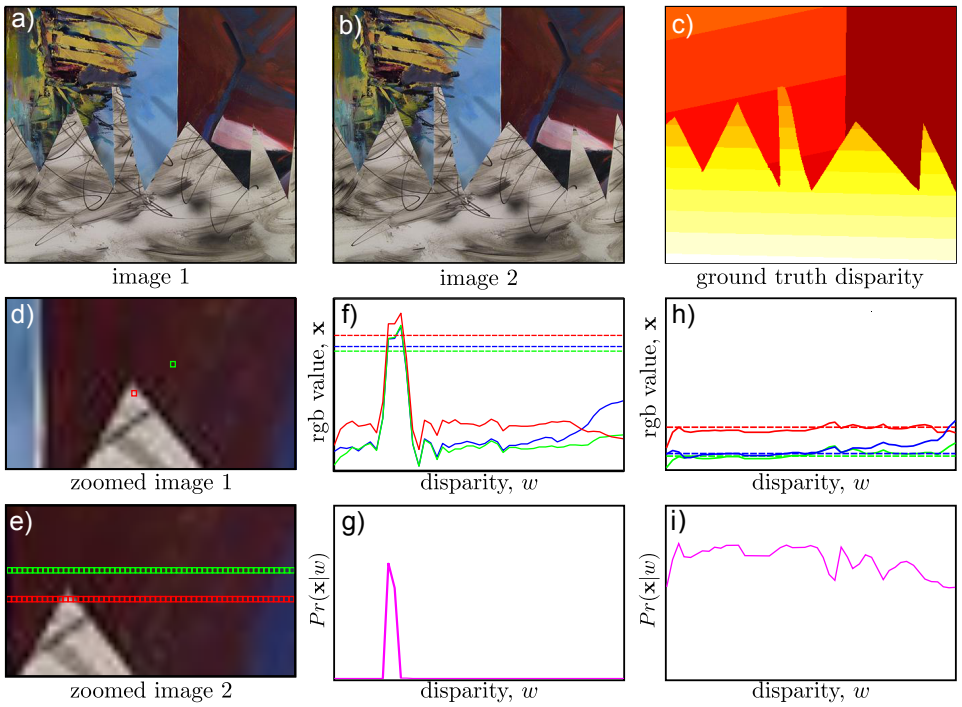


Figure 11.17 Dense stereo vision. a)-b) Two images taken from slightly different positions. The corresponding point for every pixel in the first image is somewhere on the same scanline in the second image. The horizontal offset is known as the disparity and is inversely related to depth. c) Ground truth disparity map for this image. d) Close up of part of first image with two pixels highlighted. e) Close up of second image with potential corresponding pixels highlighted. f) RGB values for red pixel (dashed lines) in first image and as a function of the position in second image (solid lines). At the correct disparity, there is very little difference between the RGB values in the two images and so g) the likelihood that this disparity is correct is large. h-i) For the green pixel (which is in a smoothly changing region of the image) there are many positions where the RGB values in the second image are similar and hence many disparities have high likelihoods; the solution is ambiguous.

horizontal offset or *disparity* between corresponding points depends on the depth. Our goal is to find the discrete disparity field $\mathbf{w}$ given the observed images $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$, from which the depth of each pixel can be recovered (figure 11.17).

We assume that the pixel in image 1 should closely resemble the pixel at the appropriate offset (disparity) in image 2, and any remaining small differences are treated as noise so that:

$$Pr(\mathbf{x}_{m,n}^{(1)}|w_{m,n}=k)=\text{Norm}_{\mathbf{x}_{m,n}^{(1)}}\left[\mathbf{x}_{m,n+k}^{(2)},\sigma^2\mathbf{I}\right], \tag{11.55}$$

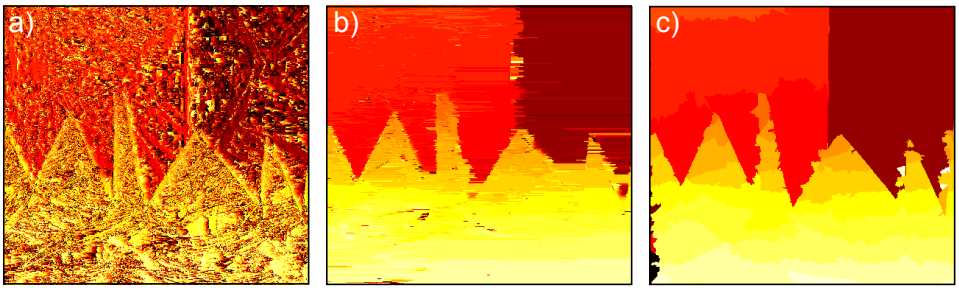


Figure 11.18 Dense stereo results. Recovered disparity maps for a) independent pixels model b) independent scanlines model, and c) tree-based model of Veksler (2005).

where $w_{m,n}$ is the disparity at pixel (m,n) of image 1, $\mathbf{x}_{m,n}^{(1)}$ is the RGB vector from pixel (m,n) of image 1 and $\mathbf{x}_{m,n}^{(2)}$ is the RGB vector from pixel (m,n) of image 2.

Unfortunately, if we compute the maximum likelihood disparities $w_{m,n}$ at each pixel separately, the result is extremely noisy (figure 11.18a). As figure 11.17 illustrates, the choice of disparity is ambiguous in regions of the image where there are few visual changes. In layman's terms, if the nearby pixels are all similar, it is difficult to establish with certainty which corresponds to a given position in the other image. To resolve this ambiguity, we introduce a prior $Pr(\mathbf{w})$ that encourages piecewise smoothness in the disparity map; we are exploiting the fact that we know that the scene mainly consists of smooth surfaces, with occasional jumps in disparity at the edge of objects.

One possible approach (attributed originally to Ohta & Kanade 1985) to recovering the disparity is to use an independent prior for each scanline so that

$$Pr(\mathbf{w}) = \prod_{m=1}^{M} Pr(\mathbf{w}_m), \tag{11.56}$$

where each scanline was organized into a chain model (figure 11.2) so that

$$Pr(\mathbf{w}_m) = Pr(w_{m,1}) \prod_{n=2}^{N} Pr(w_{m,n}|w_{m,n-1}). \tag{11.57}$$

The distributions $Pr(w_{m,n}|w_{m,n-1})$ are chosen so that they allot a high probability when adjacent disparities are the same, an intermediate probability when adjacent disparities change by a single value, and a low probability if they take values that are more widely separated. Hence, we encourage piecewise smoothness.

MAP inference can be performed within each scanline separately, using the dynamic programming approach, and the results combined to form the full disparity field $\mathbf{w}$. Although this definitely improves the fidelity of the solution, it results in a characteristic 'streaky' result (figure 11.18b). These artifacts are due to the (erroneous) assumption that the scanlines are independent. To get an improved

Problem 11.11

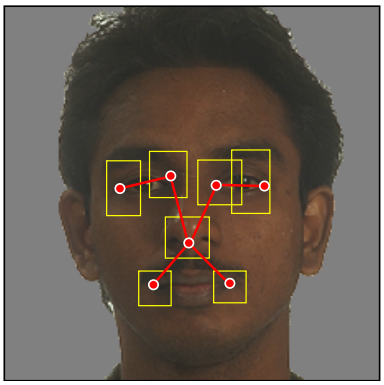


Figure 11.19 Pictorial structure. This face model consists of seven parts (red dots) which are connected together in a tree-like structure (red lines). The possible positions of each part are indicated by the yellow boxes. Although each part can take several hundred pixel positions, the MAP positions can be inferred efficiently by exploiting the tree-structure of the graph using a dynamic programming approach. Localizing facial features is a common element of many face recognition pipelines.

solution, we should smooth in the vertical direction as well, but the resulting grid-based model will contain loops, making MAP inference problematic.

Veksler (2005) addressed this problem by pruning the full grid-based model until it formed a tree. Each edge was characterized by a cost that increased if the associated pixels were close to large changes in the image; at these positions, either there is texture in the image (and so the disparity is relatively well defined) or there is an edge between two objects in the scene. In either case, there is no need to apply a smoothing prior here. Hence, the minimum spanning tree tends to retain edges in regions where they are most needed. The minimum spanning tree can be computed using a standard method such as Prim's algorithm (see Cormen *et al.* 2001).

The results of MAP inference using this model are shown in figure 11.18c. The solution is piecewise smooth in both directions and is clearly superior to either the independent pixels model or the independent scanline approach. However, even this model is an unnecessary approximation; we would ideally like the variables to be fully connected in a grid structure but this would obviously contain loops. In chapter 12, we consider models of this sort and re-visit stereo vision.

11.8.3 Pictorial Structures

Pictorial structures are models for object classes that consist of a number of individual parts that are connected together by spring-like connections. A typical example would be a face model (figure 11.19) which might consist of a nose, eyes and mouth. The spring-like connections encourage the relative positions of these features to take sensible values. For example, the mouth is strongly encouraged to be below the nose. Pictorial structures have a long history in computer vision but were revived in a modern form by Felzenszwalb & Huttenlocher (2005) who identified that if the connections between parts take the form of an acyclic graph (a tree), then they can be fit to images in polynomial time.

The goal of matching a pictorial structure to an image is to identify the positions $\{w_n\}_{n=1}^N$ of the N parts based on data $\{\mathbf{x}_n\}$ associated with each. For example, a simple system might assign a likelihood $Pr(\mathbf{x}|w_n = k)$ that is a normal distribution

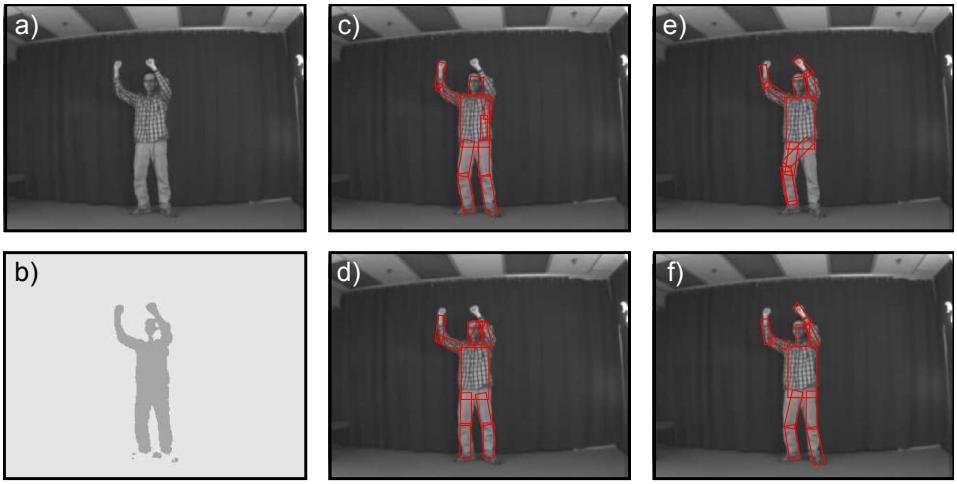


Figure 11.20 Pictorial structure for human body. a) Original image. b) After background subtraction. c-f) Four samples from the posterior distribution over part positions. Each part position is represented by a rectangle of fixed aspect ratio and characterized by its position, size, and angle. Adapted from Felzenszwalb & Huttenlocher (2005). ©2005 Springer.

over a patch of the image at position k. The relative positions of the parts are encoded using distributions of the form $Pr(w_n|w_{pa[n]})$. MAP inference in this system can be achieved using a dynamic programming technique.

Figure 11.19 shows a pictorial structure for a face. This model is something of a compromise in that it would be preferable if the features had more dense connections: for example the left eye provides information about the position of the right eye as well as the nose. Nonetheless, this type of model can reliably find features on frontal faces.

A second application is for fitting articulated models such as human bodies (figure 11.20). These naturally have the form of a tree and so the structure is determined by the problem itself. Felzenszwalb & Huttenlocher (2005) developed a system of this sort, in which each state w_n represented a joint in the model and could take K possible values, each of which represented a different position and shape of an associated rectangle.

The image was pre-classified into foreground and background using a background subtraction technique. The likelihood $Pr(\mathbf{x}_n|w=k)$ for a particular part position was evaluated using this binary image. In particular, the likelihood was chosen so that it increased if the area within the rectangle was considered foreground and the area surrounding it was considered background.

Unfortunately, MAP inference in this model is somewhat unreliable: a common failure mode is for more than one part of the body to become associated with the same part of the binary image. This is technically possible as the limbs may occlude each other, but it can also happen erroneously if one limb dominates and

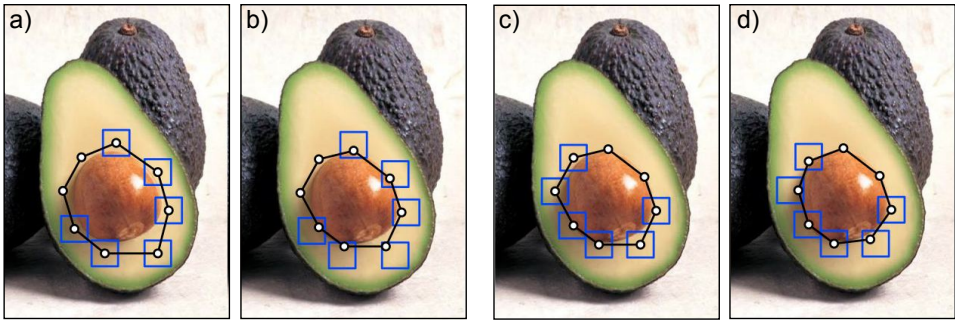


Figure 11.21 Segmentation using snakes. a) Two points are fixed, but the remaining points can take any position within their respective boxes. The posterior distribution favors positions that are on image contours (due to the likelihood term) and positions that are close to other points (due to the pairwise connections). b) Results of inference. c) Two other points are considered fixed. d) Result of inference. In this way, a closed contour in the image is identified. Adapted from Felzenszwalb & Zabih (2011). ©2011 IEEE.

supports the rectangle model significantly more than the others. Felzenszwalb & Huttenlocher (2005) dealt with this problem by drawing samples from the posterior distribution $Pr(w_{1\ldots N}|\mathbf{x}_{1\ldots N})$ over the positions of the parts of the model, and using a more complex criterion to choose the most promising sample.

11.8.4 Segmentation

Problem 11.12

In section 7.9.3 we considered segmentation as the problem of labeling pixels according to the object to which they belong. A different approach to segmentation is to infer the position of a closed contour that delineates two objects. In inference, the goal is usually to infer the positions of a set of points $\{w_n\}$ on the boundary of this contour based on the image data $\mathbf{x}$. As we update these points during an attempt to find the MAP solution, the contour moves across the image, and for this reason this type of model is referred to as an active contour or snake model.

Figure 11.21 shows an example of fitting this type of model. At each iteration, the positions $\{w_n\}$ of all of the points except two are updated, and can take any position within small region surrounding their previous position. The likelihood of taking a particular value $w_n = k$ is high at positions in the image where the intensity changes rapidly (i.e., the edges) and low in constant regions. In addition, neighboring points are connected and have an attractive force: they are more likely to be close to one another. As usual, inference can be carried out using the dynamic programming method. During inference, the points tend to become closer together (due to their mutual attraction) but get stuck on the edge of an object.

In the full system, this process is repeated, but with a different pair of adjacent

points chosen to be fixed at each step. Hence, the dynamic programming is a component step of a larger inference problem. As the inference procedure continues, the contour moves across the image and eventually fixes onto the boundary of an object. For this reason, these models are known as *snakes* or *active contour models*. They are considered in more detail in chapter 17.

Discussion

In this chapter we have considered models based on acyclic graphs (chains and trees). In the chapter 12 we will consider grid based models which contain many loops. We will see that MAP inference is only tractable in a few special cases. In contrast to this chapter, we will also see a large difference between directed and undirected models.

Notes

Dynamic programming: Dynamic programming is used in many vision algorithms, including those where there is not necessarily a clear probabilistic interpretation. It is an attractive approach when it is applicable because of its speed, and some efforts have been made to improve this further (Raphael 2001). Interesting examples include image retargeting (Avidan & Shamir 2007), contour completion (Sha'ashua & Ullman 1988), fitting of deformable templates (Amit & Kong 1996; Coughlan *et al.* 2000), shape matching (Basri *et al.* 1998), the computation of superpixels (Moore *et al.* 2008) and semantic labeling of scenes with tiered structure (Felzenszwalb & Veksler 2010) as well as the applications described in this chapter. Felzenszwalb & Zabih (2011) provide a recent review of dynamic programming and other graph algorithms in computer vision.

Stereo vision: Dynamic programming was variously applied to stereo vision by Baker & Binford (1981), Ohta & Kanade (1985) (who use a model based on edges) and Geiger *et al.* (1992) (who used a model based on intensities). Birchfield & Tomasi (1998) improved the speed by removing unlikely search nodes from the dynamic programming solution and introduced a mechanism that made depth discontinuities more likely where there was intensity variation. Torr & Criminisi (2004) developed a system that integrated dynamic programming with known constraints such as matched keypoints. Gong & Yang (2005) developed a dynamic programming algorithm that ran on a graphics processing unit (GPU). Kim *et al.* (2005) introduced a method for identifying disparity candidates at each pixel using spatial filters and a two-pass method that performed optimization both along and across the scanlines. Veksler (2005) used dynamic programming in a tree to solve for the whole image at once, and this idea has subsequently been used in a method based on line segments (Deng & Lin 2006). A recent quantitative comparison of dynamic programming algorithms in computer vision can be found in Salmen *et al.* (2009). Alternative approaches to stereo vision which are not based on dynamic programming are considered in chapter 12.

Pictorial structures: Pictorial structures were originally introduced by Fischler & Erschlager (1973) but recent interest was stimulated by the work of Felzenszwalb & Huttenlocher (2005) who introduced efficient methods of inference based on dynamic programming. There have been a number of attempts to improve the appearance (likelihood) term of the model (Kumar *et al.* 2004; Eichner & Ferrari 2009; Andriluka *et al.* 2009; Felzenszwalb *et al.* 2010). Models that do not conform to a tree structure have also been introduced (Kumar *et al.* 2004; Sigal & Black 2006; Ren *et al.* 2005; Jiang & Martin 2008) and here alternative methods such as loopy propagation must be used for inference. These more general structures are particularly important for addressing problems associated with occlusions in human body models. Other authors have based their model on a mixture of trees (Everingham *et al.* 2006; Felzenszwalb *et al.* 2010). In terms of applications, Ramanan *et al.* (2008) have developed a notable system for tracking humans in video sequences based on pictorial structures, Everingham *et al.* (2006) have developed a widely used system for locating facial features and Felzenszwalb *et al.* (2010) have presented a system that is used for detecting more general objects.

Hidden Markov models: Hidden Markov models are essentially chain based models that are applied to quantities evolving in time. Good tutorials on the subject including details of how to learn them in the unsupervised case can be found in Rabiner (1989) and Ghahramani (2001). Their most common application in vision is for gesture recognition (Starner *et al.* 1998; Rigoll *et al.* 1998 and see Moni & Ali 2009 for a recent review), but they have also been used in other contexts such as modeling interactions of pedestrians (Oliver *et al.* 2000). Some recent work (e.g., Bor Wang *et al.* 2006) uses a related dis-

criminative model for tracking objects in time known as a *conditional random field* (see chapter 12).

Snakes: The idea of a contour evolving over the surface of an image is due to Kass *et al.* (1987). Both Amini *et al.* (1990) and Geiger *et al.* (1995) describe dynamic programming approaches to this problem. These models are considered further in chapter 17.

Belief propagation: The sum-product algorithm (Kschischang *et al.* 2001) is a development of earlier work on belief propagation by Pearl (1988). The factor graph representation is due to Frey *et al.* (1997). The use of belief propagation for finding marginal posteriors and MAP solutions in graphs with loops has been investigated by Murphy *et al.* (1999) and Weiss & Freeman (2001), respectively. Notable applications of loopy belief propagation in vision include stereo vision (Sun *et al.* 2003) and super-resolving images (Freeman *et al.* 2000). More information about belief propagation can be found in machine learning textbooks such as Bishop (2006), Barber (2012), and Koller & Friedman (2009).

Problems

Problem 11.1 Compute by hand the lowest possible cost for traversing the graph in figure 11.22 using the dynamic programming method.

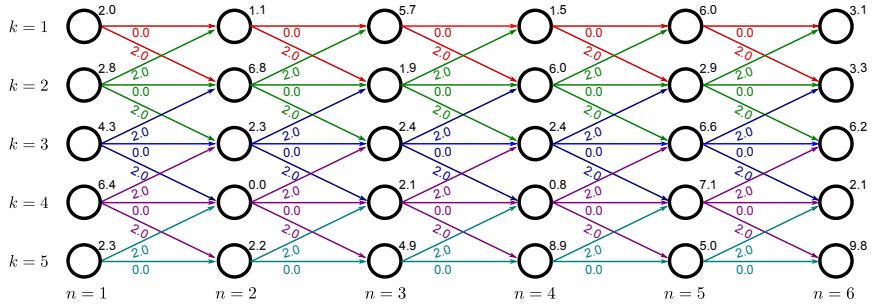


Figure 11.22 Dynamic programming example for problem 11.1.

Problem 11.2 MAP inference in chain models can also be performed by running Djikstra's algorithm on the graph in figure 11.23, starting from the node on the left hand side and terminating when we first reach the node on the right hand side. If there are N variables, each of which takes K values, what is the best and worst case complexity of the algorithm? Describe a situation where Djikstra's algorithm outperforms dynamic programming.

Problem 11.3 Consider the graphical model in figure 11.24a. Write out the cost function for MAP estimation in the form of equation 11.17. Discuss the difference between your answer and equation 11.17.

Problem 11.4 Compute the solution (minimum cost path) to the dynamic programming problem on the tree in figure 11.25 (which corresponds to the graphical model from figure 11.6).

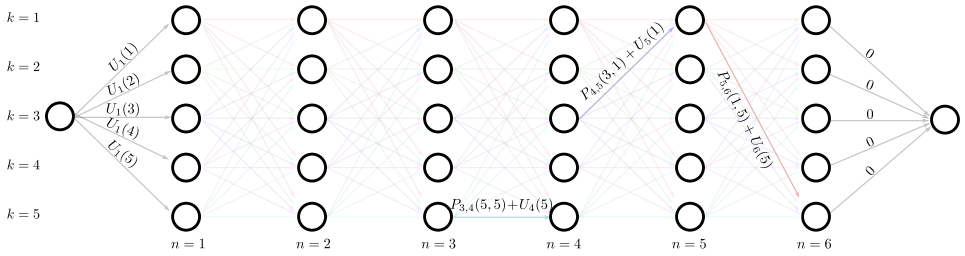


Figure 11.23 Graph construction for problem 11.2. This is the same as the dynamic programming graph (figure 11.3) except that: (i) there are two extra nodes at the start and the end of the graph. (ii) There are no vertex costs. (iii) The costs associated with the left-most edges are $U_1(k)$ and the costs associated with the right-most edges are 0. The general edge cost for passing from label a and node n to label b at node $n+1$ is given by $P_{n,n+1}(a,b) + U_{n+1}(b)$.

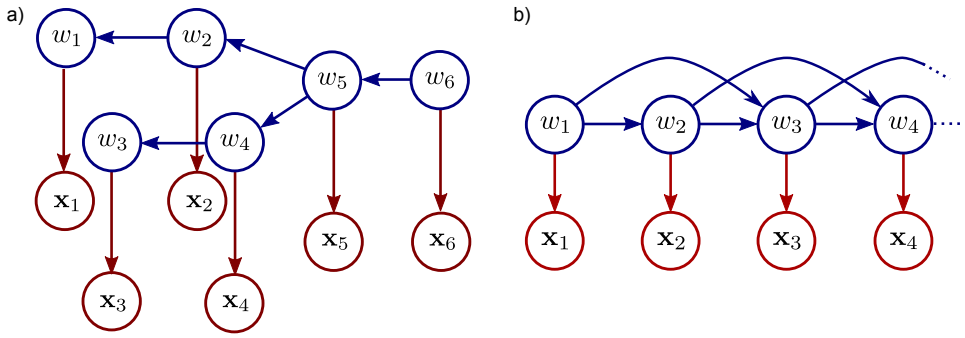


Figure 11.24 a) Graphical model for problem 11.3. b) Graphical model for problem 11.10. The unknown variables $w_3, w_4 \ldots$ in this model receive connections from the two preceding variables and so the graph contains loops.

Problem 11.5 MAP inference for the chain model can be expressed as

$$\hat{w}_N = \underset{w_N}{\text{argmax}}\left[\max_{w_1}\left[\max_{w_2}\left[\ldots \max_{w_{N-1}}\left[\sum_{n=1}^{N}\log[Pr(\mathbf{x}_n|w_n)] + \sum_{n=2}^{N}\log[Pr(w_n|w_{n-1})]\right]\ldots\right]\right]\right]$$

Show that it is possible to compute this expression piecewise by moving the maximization terms through the summation sequence in a manner similar to that described in section 11.4.1.

Problem 11.6 Develop an algorithm that can compute the marginal distribution for an arbitrary variable w_n in a chain model.

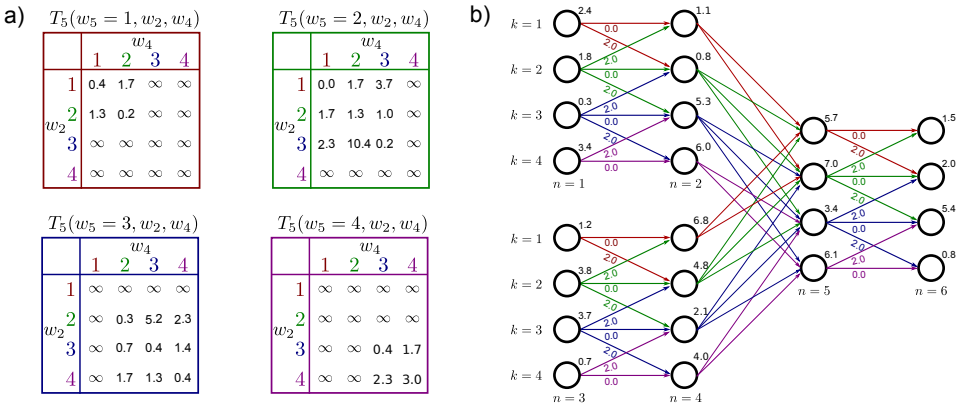


Figure 11.25 Dynamic programming example for problem 11.4.

Problem 11.7 Develop an algorithm that computes the joint marginal distribution of any two variables w_m and w_n in a chain model.

Problem 11.8 Consider the following two distributions over three variables x_1, x_2, and x_3:

$$\begin{aligned} Pr(x_1, x_2, x_3) &= \frac{1}{Z_1}\phi_{12}[x_1, x_2]\phi_{23}[x_2, x_3]\phi_{31}[x_3, x_1] \\ Pr(x_1, x_2, x_3) &= \frac{1}{Z_2}\phi_{123}[x_1, x_2, x_3]. \end{aligned}$$

Draw (i) an undirected model and (ii) a factor graph for each distribution. What do you conclude?

Problem 11.9 Convert each of the graphical models in figure 11.26 into the form of a factor graph. Which of the resulting factor graphs take the form of a chain?

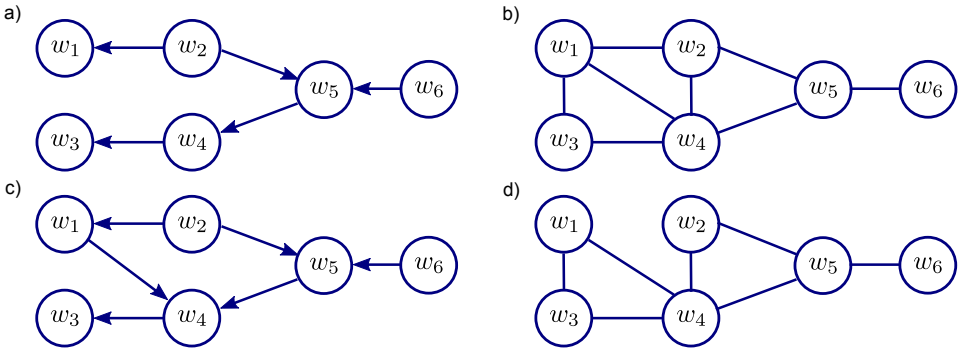


Figure 11.26 Graphical models for problem 11.9.

Problem 11.10 Figure 11.24b shows a chain model in which each unknown variable w depends on its two predecessors. Describe a dynamic programming approach to finding

the MAP solution. (hint: you need to combine variables.) If there are N variables in the chain and each takes K values, what is the overall complexity of your algorithm?

Problem 11.11 In the stereo vision problem, the solution was very poor when the pixels are treated independently (figure 11.18a). Suggest some improvements to this method (while keeping the pixels independent).

Problem 11.12 Consider a variant on the segmentation application (figure 11.21) in which we update all of the contour positions at once. The graphical model for this problem is a loop (i.e., a chain where there is also a edge between w_N and w_1). Devise an approach to finding the exact MAP solution in this model. If there are N variables each of which can take K values, what is the complexity of your algorithm?

Chapter 12

Models for grids

In chapter 11, we discussed models that were structured as chains or trees. In this chapter, we consider models that associate a label with each pixel of an image. Since the unknown quantities are defined on the pixel lattice, models defined on a grid structure are appropriate. In particular, we will consider graphical models in which each label has a direct probabilistic connection to each of its four neighbours. Critically, this means that there are loops in the underlying graphical model and so the dynamic programming and belief propagation approaches of the previous chapter are no longer applicable.

These grid models are predicated on the idea that the pixel provides only very ambiguous information about the associated label. However, certain spatial configurations of labels are known to be more common than others, and we aim to exploit this knowledge to resolve the ambiguity. In this chapter, we describe the relative preference for different configurations of labels with a pairwise *Markov random field* or MRF. As we shall see, maximum a posteriori inference for pairwise MRFs is tractable in some circumstances using a family of approaches known collectively as *graph cuts*.

To motivate the grid models, we introduce a representative application. In *image denoising* we observe a corrupted image in which the intensities at a certain proportion of pixels have been randomly changed to another value according to a uniform distribution (figure 12.1). Our goal is to recover the original clean image. We note two important aspects of the problem.

1. Most of the pixels are uncorrupted,so the data usually tell us which intensity value to pick.
2. The uncorrupted image is mainly smooth, with few changes between intensity levels.

Consequently, our strategy will be to construct a generative model where the MAP solution is an image that is mostly the same as the noisy version, but is smoother. As part of this solution, we need to define a probability distribution over images that favors smoothness. In this chapter, we will use a discrete formulation of a Markov random field to fulfil this role.

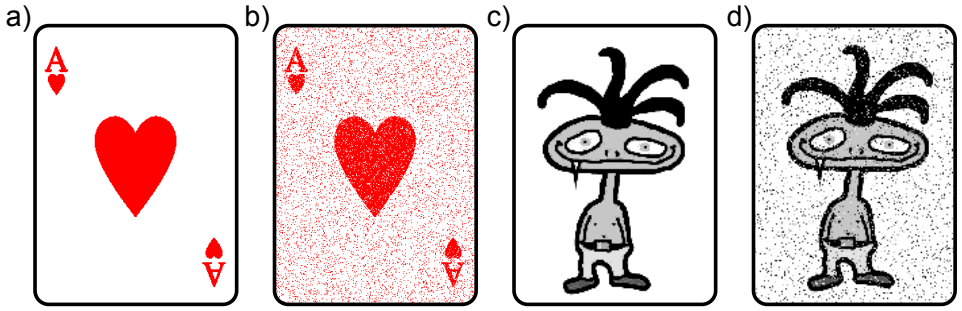


Figure 12.1 Image denoising. a) Original binary image. b) Observed image created by randomly flipping the polarity of a fixed proportion of pixels. Our goal is to recover the original image from the corrupted one. c) Original grayscale image. d) Observed corrupted image is created by setting a certain proportion of the pixels to values drawn from a uniform distribution. Once more, we aim to recover the original image.

12.1 Markov random fields

A *Markov random field* is formally determined by:

- A set of sites $\mathcal{S} = \{1 \ldots N\}$. These will correspond to the N pixel locations.
- A set of random variables $\{w_n\}_{n=1}^N$ associated with each of the sites.
- A set of neighbors $\{\mathcal{N}_n\}_{n=1}^N$ at each of the N sites.

To be a Markov random field, the model must obey the Markov property:

$$Pr(w_n|w_{\mathcal{S}\setminus n}) = Pr(w_n|w_{\mathcal{N}_n}). \tag{12.1}$$

In other words, the model should be conditionally independent of all of the other variables given its neighbors. This property should sound familiar: this is exactly how conditional independence works in an undirected graphical model.

Consequently, we can consider a Markov random field (MRF) as an undirected model (section 10.3) that describes the joint probability of the variables as a product of potential functions so that

$$Pr(\mathbf{w}) = \frac{1}{Z}\prod_{j=1}^{J}\phi_j[\mathbf{w}_{\mathcal{C}_j}], \tag{12.2}$$

where $\phi_j[\bullet]$ is the j^{th} potential function and always returns a non-negative value. This value depends on the state of the subset of variables $\mathcal{C}_j \subset \{1, \ldots, N\}$. In this context, this subset is known as a *clique*. The term Z is called the partition function and is a normalizing constant that ensures that the result is a valid probability distribution.

Alternatively, we can rewrite the model as a Gibbs distribution:

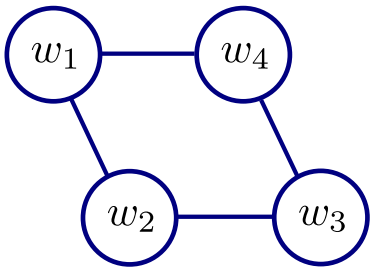


Figure 12.2 Graphical model for worked MRF example. The variables form a 2×2 grid. This is an undirected model where each link represents a potential function defined over the two variables that it connects. Each potential returns a positive number that indicates the tendency of the two variables to take these particular values.

$$Pr(\mathbf{w}) = \frac{1}{Z}\exp\left[-\sum_{j=1}^{J}\psi_j[\mathbf{w}_{\mathcal{C}_j}]\right], \tag{12.3}$$

where $\psi[\bullet] = -\log[\phi[\bullet]]$ is known as a cost function and returns either positive or negative values.

12.1.1 Grid example

In a Markov random field, each potential function $\phi[\bullet]$ (or cost function $\psi[\bullet]$) addresses only a small subset of the variables. In this chapter, we will mainly be concerned with *pairwise Markov random fields* in which the cliques (subsets) consist of only neighboring pairs in a regular grid structure.

To see how the pairwise MRF can be used to encourage smoothness in an image, consider the graphical model for a 2×2 image (figure 12.2). Here, we have defined the probability $Pr(w_{1\ldots4})$ over the associated discrete states as a normalized product of pairwise terms:

$$Pr(\mathbf{w}) = \frac{1}{Z}\phi_{12}(w_1, w_2)\phi_{23}(w_2, w_3)\phi_{34}(w_3, w_4)\phi_{41}(w_4, w_1), \tag{12.4}$$

where $\phi_{mn}(w_m, w_n)$ is a potential function that takes the two states w_m and w_n and returns a positive number.

Let us consider the situation where the world state w_n at each pixel is binary and so takes a value of 0 or 1. The function ϕ_{mn} will now return four possible values depending on which of the four configurations $\{00, 01, 10, 11\}$ of w_m and w_n is present. For simplicity, we will assume that the functions $\phi_{12}, \phi_{23}, \phi_{34}$ and ϕ_{41} are identical and that for each:

$$\begin{aligned} \phi_{mn}(0,0) &= 1.0 \qquad & \phi_{mn}(0,1) &= 0.1 \\ \phi_{mn}(1,0) &= 0.1 \qquad & \phi_{mn}(1,1) &= 1.0. \end{aligned} \tag{12.5}$$

Since there are only four binary states, we can calculate the constant Z explicitly by computing the un-normalized probabilities for each of the 16 possible combinations, and taking the sum. The resulting probabilities for each of the 16 possible states are:

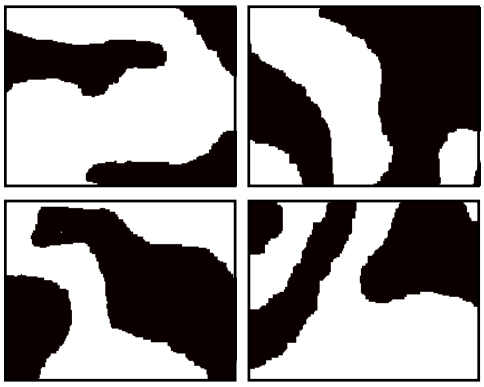


Figure 12.3 Samples from Markov random field prior. Four samples from the MRF prior which were generated using a Gibbs sampling procedure (see section 10.7.2). Each sample is a binary image that is smooth almost everywhere; there are only very occasional changes from black to white and vice-versa. This prior encourages smooth solutions (like the original image in the denoising problems) and discourages isolated changes in label (as are present in the noise).

$w_{1\ldots4}$	$Pr(w_{1\ldots4})$	$w_{1\ldots4}$	$Pr(w_{1\ldots4})$	$w_{1\ldots4}$	$Pr(w_{1\ldots4})$	$w_{1\ldots4}$	$Pr(w_{1\ldots4})$
0000	0.47176	0100	0.00471	1000	0.00471	1100	0.00471
0001	0.00471	0101	0.00005	1001	0.00471	1101	0.00471
0010	0.00471	0110	0.00471	1010	0.00005	1110	0.00471
0011	0.00471	0111	0.00471	1011	0.00471	1111	0.47176

Problem 12.1

The potential functions in equation 12.5 encourage smoothness: the functions ϕ_{mn} return higher values when the neighbors take the same state and lower values when they differ, and this is reflected in the resulting probabilities.

We can visualize this by scaling this model up to a larger image-sized grid where there is one node per pixel, and drawing samples from the resulting probability distribution (figure 12.3). The resulting binary images are mostly smooth, with only occasional changes between the two values.

It should be noted that for this more realistically-sized model, we cannot compute the normalizing constant Z by brute force as for the 2×2 case. For example, with 10,000 pixels each taking a binary value, the normalizing constant is the sum of $2^{10,000}$ terms. In general we will have to cope with only knowing the probabilities up to an unknown scaling factor.

12.1.2 Image denoising with discrete pairwise MRFs

Now we will apply the pairwise Markov random field model to the denoising task. Our goal is to recover the original image pixel values from the observed noisy image.

More precisely, the observed image $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$ is assumed to consist of discrete variables where the different possible values (labels) represent different intensities. Our goal is to recover the original uncorrupted image $\mathbf{w} = \{w_1, w_2, \ldots, w_N\}$, which also consists of discrete variables representing the intensity. We will initially restrict our discussion to generative models, and compute the posterior probability over the unknown world state $\mathbf{w}$ using Bayes' rule

$$Pr(w_{1\ldots N}|x_{1\ldots N}) = \frac{\prod_{n=1}^{N} Pr(x_n|w_n) Pr(w_{1\ldots N})}{Pr(x_{1\ldots N})}, \tag{12.6}$$

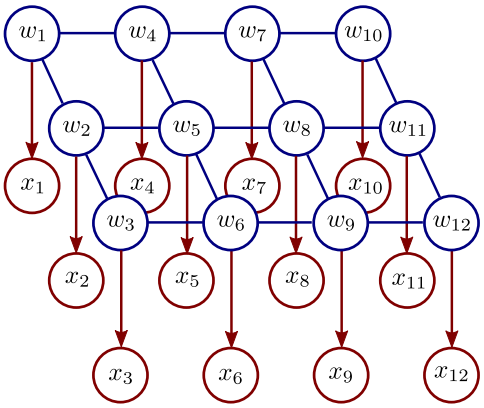


Figure 12.4 Denoising model. The observed data $\mathbf{x}_n$ at pixel n is conditionally dependent on the associated world state w_n (red directed edges). Each world state w_n has undirected edges to its four-connected neighbors (blue undirected edges). This is hence a mixed model: it contains both directed and undirected elements. Together the world states are connected in a Markov random field with cliques that consist of neighboring pairs of variables. For example variable w_5 contributes to cliques $\mathcal{C}_{25}, \mathcal{C}_{45}, \mathcal{C}_{65}, \mathcal{C}_{85}$.

where we have assumed that the conditional probability $Pr(x_{1\ldots N}|w_{1\ldots N})$ factorizes into a product of individual terms associated with each pixel. We will first consider denoising binary images in which the noise process flips the pixel polarity with probability ρ so that

$$\begin{aligned} Pr(x_n|w_n = 0) &= \text{Bern}_{x_n}[\rho] \\ Pr(x_n|w_n = 1) &= \text{Bern}_{x_n}[1-\rho]. \end{aligned} \tag{12.7}$$

We subsequently consider gray level denoising where the observed pixel is replaced with probability ρ by a draw from a uniform distribution.

We now define a prior that encourages the labels w_n to be smooth: we want them to mostly agree with the observed image, but to discourage configurations with isolated changes in label. To this end, we model the prior as a pairwise MRF. Each pair of four-connected neighboring pixels contributes one clique so that

$$Pr(w_{1\ldots N}) = \frac{1}{Z}\exp\left[-\sum_{(m,n)\in\mathcal{C}}\psi[w_m, w_n, \boldsymbol{\theta}]\right], \tag{12.8}$$

where we have assumed that the clique costs $\psi[\bullet]$ are the same for every (w_m, w_n). The parameters $\boldsymbol{\theta}$ define the costs $\psi[\bullet]$ for each combination of neighboring pairwise values,

$$\psi[w_m = j, w_n = k, \boldsymbol{\theta}] = \theta_{jk}, \tag{12.9}$$

so when the first variable w_m in the clique takes label j and the second variable w_n takes label k ,we pay a price of θ_{jk}. As before, we will choose these values so that there is a small cost when neighboring labels are the same (so θ_{00} and θ_{11} are small) and a larger one when the neighboring labels differ (so θ_{01} and θ_{10} are large). This has the effect of encouraging solutions that are mostly smooth.

The associated graphical model is illustrated in figure 12.4. It is a mixed model, containing both directed and undirected links. The likelihood terms (equation 12.7) contribute the red directed links between the observed data and the denoised image

at each pixel, and the MRF prior (equation 12.8) contributes the blue grid that connects the pixels together.

12.2 MAP inference for binary pairwise MRFs

To denoise the image, we estimate the variables $\{w_n\}_{n=1}^N$ using MAP inference; we aim to find the set of world states $\{w_n\}_{n=1}^N$ that maximizes the posterior probability $Pr(w_{1\ldots N}|\mathbf{x}_{1\ldots N})$ so that

$$\begin{aligned}\hat{w}_{1\ldots N} &= \underset{w_{1\ldots N}}{\operatorname{argmax}}\left[Pr(w_{1\ldots N}|\mathbf{x}_{1\ldots N})\right] \\ &= \underset{w_{1\ldots N}}{\operatorname{argmax}}\left[\prod_{n=1}^{N} Pr(x_n|w_n)Pr(w_{1\ldots N})\right] \\ &= \underset{w_{1\ldots N}}{\operatorname{argmax}}\left[\sum_{n=1}^{N} \log[Pr(x_n|w_n)] + \log[Pr(w_{1\ldots N})]\right], \end{aligned} \tag{12.10}$$

where we have applied Bayes' rule and transformed to the log domain. Because the prior is an MRF with pairwise connections, we can express this as

$$\begin{aligned}\hat{w}_{1\ldots N} &= \underset{w_{1\ldots N}}{\operatorname{argmax}}\left[\sum_{n=1}^{N} \log[Pr(x_n|w_n)] - \sum_{(m,n)\in\mathcal{C}} \psi[w_m, w_n, \boldsymbol{\theta}]\right] \\ &= \underset{w_{1\ldots N}}{\operatorname{argmin}}\left[\sum_{n=1}^{N} -\log[Pr(x_n|w_n)] + \sum_{(m,n)\in\mathcal{C}} \psi[w_m, w_n, \boldsymbol{\theta}]\right] \\ &= \underset{w_{1\ldots N}}{\operatorname{argmin}}\left[\sum_{n=1}^{N} U_n(w_n) + \sum_{(m,n)\in\mathcal{C}} P_{mn}(w_m, w_n)\right], \end{aligned} \tag{12.11}$$

where $U_n(w_n)$ denotes the *unary* term at pixel n. This is a cost for observing the data at pixel n given state w_n, and is the negative log likelihood term. Similarly, $P_{mn}(w_m, w_n)$ denotes the *pairwise* term. This is a cost for placing labels w_m and w_n at neighboring locations m and n, and is due to the clique costs $\psi[w_m, w_n, \boldsymbol{\theta}]$ from the MRF prior. Note that we have omitted the term $-\log[Z]$ from the MRF definition as it is constant with respect to the states $\{w_n\}_{n=1}^N$ and hence does not affect the optimal solution.

The cost function in equation 12.11 can be optimized using a set of techniques known collectively as *graph cuts*. We will consider three cases:

- binary MRFs (i.e., $w_i \in \{0, 1\}$) where the costs for different combinations of adjacent labels are "submodular" (we will explain what this means later in the chapter). Exact MAP inference is tractable here.

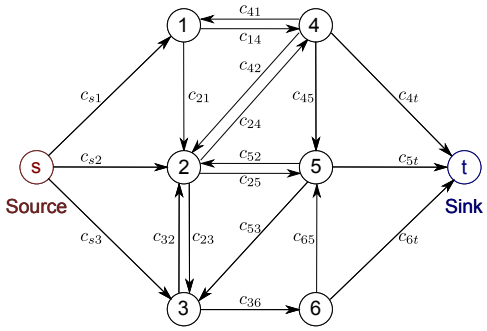


Figure 12.5 Max flow problem: we are given a network of vertices connected by directed edges, each of which has a non-negative capacity c_{mn}. There are two special vertices s and t termed the source and sink, respectively. In the max-flow problem, we seek to push as much 'flow' from source to sink while respecting the capacities of the edges.

- multi-label MRFs (i.e., $w_i \in \{1, 2, \ldots, K\}$) where the costs are "submodular." Once more, exact MAP inference is possible.
- multi-label MRFs where the costs are more general. Exact MAP inference is intractable, but good approximate solutions can be found in some cases.

To solve these MAP inference tasks, we will translate them into the form of *maximum flow* (or *max-flow*) problems. Max-flow problems are well-studied, and exact polynomial time algorithms exist. In the following section we describe the max-flow problem and its solution. In subsequent parts of the chapter, we describe how to translate MAP inference in Markov random fields into a max-flow problem.

12.2.1 Max-flow / Min-cut

Consider a graph $\mathbf{G} = (\mathcal{V}, \mathcal{E})$ with vertices $\mathcal{V}$ and directed edges $\mathcal{E}$ connecting them (figure 12.5). Each edge has a non-negative *capacity* so that the edge between vertices m and n has capacity c_{mn}. Two of the vertices are treated as special and are termed the *source* and the *sink*.

Consider transferring some quantity ('flow') through the network from the source to the sink. The goal of the max-flow algorithm is to compute the maximum amount of flow that can be transferred across the network without exceeding any of the edge capacities.

When the maximum possible flow is being transferred – the so-called *max-flow* solution – every path from source to sink must include a saturated edge (one where the capacity is reached). If not, then we could push more flow down this path, and so by definition this is not the maximum flow solution.

It follows that an alternate way to think about the problem is to consider the edges that saturate. We define a *cut* on the graph to be a minimal set of edges that separate the source from the sink. In other words, when these edges are removed, there is no path from the source to the sink. More precisely, a cut partitions the vertices into two groups: vertices that can be reached by some path from the source, but cannot reach the sink, and vertices that cannot be reached from the source, but can reach the sink via some path. For short, we will refer to a cut as 'separating' the source from the sink. Every cut is given an associated cost, which is the sum of the capacities of the excised edges.

Since the saturated edges in the max-flow solution separate the source from the sink, they form a cut. In fact, this particular choice of cut has the minimum possible cost and is referred to as the *min-cut* solution. Hence, the maximum flow and minimum cut problems can be considered interchangeably.

Augmenting paths algorithm for maximum flow

There are many algorithms to compute the maximum flow, and to describe them properly is beyond the scope of this volume. However, for completeness, we present a sketch of the *augmenting paths* algorithm (figure 12.6).

Consider choosing any path from the source to the sink and pushing the maximum possible amount of flow along it. This flow will be limited by the edge on that path that has the smallest capacity, which will duly saturate. We remove this amount of flow from the capacities of all of the edges along the path, causing the saturated edge to have a new capacity of zero. We repeat this procedure, finding a second path from source to sink, pushing as much flow as possible along it, and updating the capacities. We continue this process until there is no path from source to sink without at least one saturated edge. The total flow that we have transferred is the maximum flow, and the saturated edges form the minimum cut.

In the full algorithm, there are some extra complications: for example, if there is already some flow along edge $i-j$, it may be that there is a remaining path from source to sink that includes the edge $j-i$. In this situation, we reduce the flow in $i-j$ before adding flow to $j-i$. The reader should consult a specialized text on graph-based algorithms for more details.

If we choose the path with the greatest remaining capacity at each step, the algorithm is guaranteed to converge and has complexity $O(|\mathcal{E}|^2|\mathcal{V}|)$ where $|\mathcal{E}|$ is the number of edges and $|\mathcal{V}|$ the number of vertices in the graph. From now on, we will assume that the max-flow/min-cut problem can be solved, and concentrate on how to convert MAP estimation problems with MRFs into this form.

12.2.2 MAP inference: binary variables

Algorithm 12.1

Recall that to find the MAP solution we must find

$$\hat{w}_{1\ldots N} = \underset{w_{1\ldots N}}{\operatorname{argmin}} \left[\sum_{n=1}^{N} U_n(w_n) + \sum_{(m,n)\in\mathcal{C}} P_{mn}(w_m, w_n) \right], \tag{12.12}$$

where $U_n(w_n)$ denotes the *unary* term and $P_{mn}(w_m, w_n)$ denotes the *pairwise* term.

For pedagogical reasons, we will first consider cases where the unary terms are positive and the pairwise terms have the following zero-diagonal form

$$\begin{aligned} P_{mn}(0,0) &= 0 & P_{mn}(1,0) &= \theta_{10} \\ P_{mn}(0,1) &= \theta_{01} & P_{mn}(1,1) &= 0, \end{aligned}$$

where $\theta_{01}, \theta_{10} > 0$. We discuss the more general case later in this section.

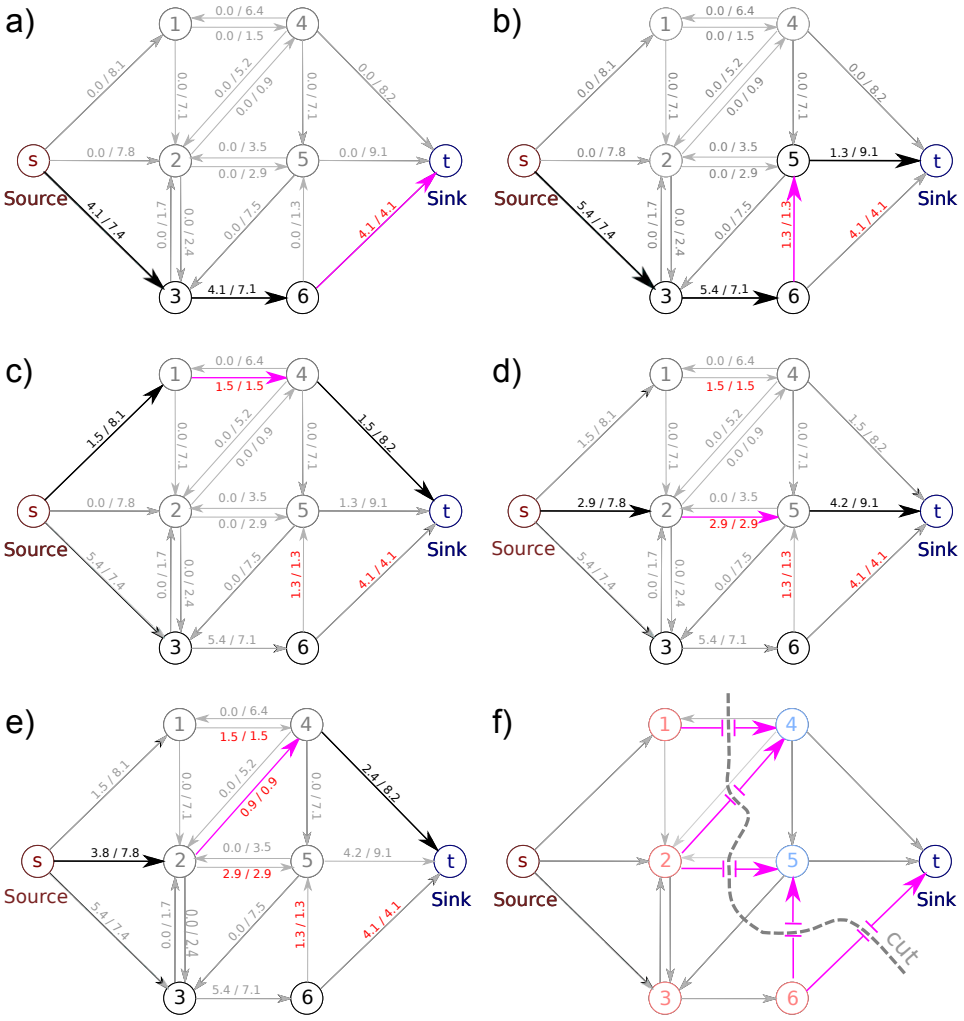


Figure 12.6 Augmenting paths algorithm for max-flow. The numbers attached to the edges correspond to current flow / capacity. a) We choose any path from source to sink with spare capacity, and push as much flow as possible along this path. The edge with the smallest capacity (here edge 6-t) saturates. b) We then choose another path where there is still spare capacity and push as much flow as possible. Now edge 6-5 saturates. c-e) We repeat this until there is no path from source to sink that does not contain a saturated edge. The total flow pushed is the maximum flow. f) In the min-cut problem, we seek a set of edges that separate the source from the sink and have minimal total capacity. The min-cut (dashed line) consists of the saturated edges in the max-flow problem. In this example, the paths were chosen arbitrarily, but to ensure that this algorithm converges in the general case, we should choose the remaining path with the greatest capacity at each step.

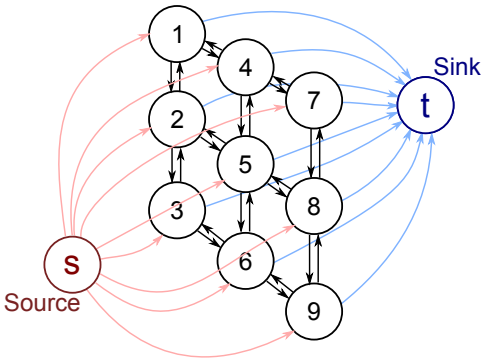


Figure 12.7 Graph structure for finding MAP solution for a MRF with binary labels and pairwise connections in a 3×3 image. There is one vertex per pixel and neighbors in the pixel grid are connected by reciprocal pairs of directed edges. Each pixel vertex receives a connection from the source and sends a connection to the sink. To separate source from sink, the cut must include one of these two edges for each vertex. The choice of which edge is cut will determine which of two labels is assigned to the pixel.

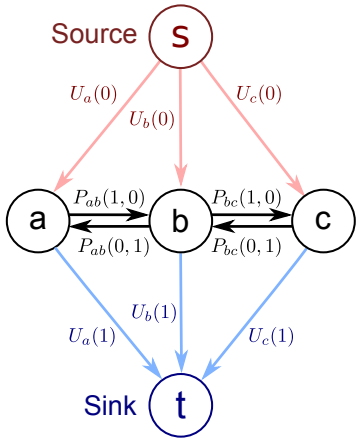


Figure 12.8 Graph construction for binary MRF with diagonal pairwise terms using simple 1D example. After the cut, vertices attached to the source are given label 1 and vertices attached to the sink are given label 0. We hence attach the appropriate unary costs to the links between the sink/source and the pixel vertices. The pairwise costs are attached to the horizontal links between pixels as shown. This arrangement ensures that the correct cost is paid for each of the eight possible solutions (see figure 12.9).

The key idea will be to set up a directed graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ and attach weights to the edges, so that the minimum cut on this graph corresponds to the maximum a posteriori solution. In particular, we construct a graph with one vertex per pixel, and a pair of directed edges between adjacent vertices in the pixel grid. In addition, there is a directed edge from the source to every vertex and a directed edge from every vertex to the sink (figure 12.7).

Now consider a cut on the graph. In any cut we must either remove the edge that connects the source to a pixel vertex, or the edge that connects the pixel vertex to the sink, or both. If we do not do this, then there will still be a path from source to sink and it is not a valid cut. For the minimum cut, we will never cut both (assuming the general case where the two edges have different capacities) – this is unnecessary and will inevitably incur a greater cost than cutting one or the other. We will label pixels where the edge to the source was cut as $w_n = 0$, and pixels where the edge to the sink was cut as having label $w_n = 1$. So each plausible minimum cut is associated with a pixel labeling.

Our goal is now to assign capacities to the edges, so the cost of each cut matches the cost of the associated labeling as prescribed in equation 12.12. For simplicity, we illustrate this with a 1D image with three pixels (figure 12.8), but we stress that all the ideas are also valid for 2D images and higher dimensional constructions.

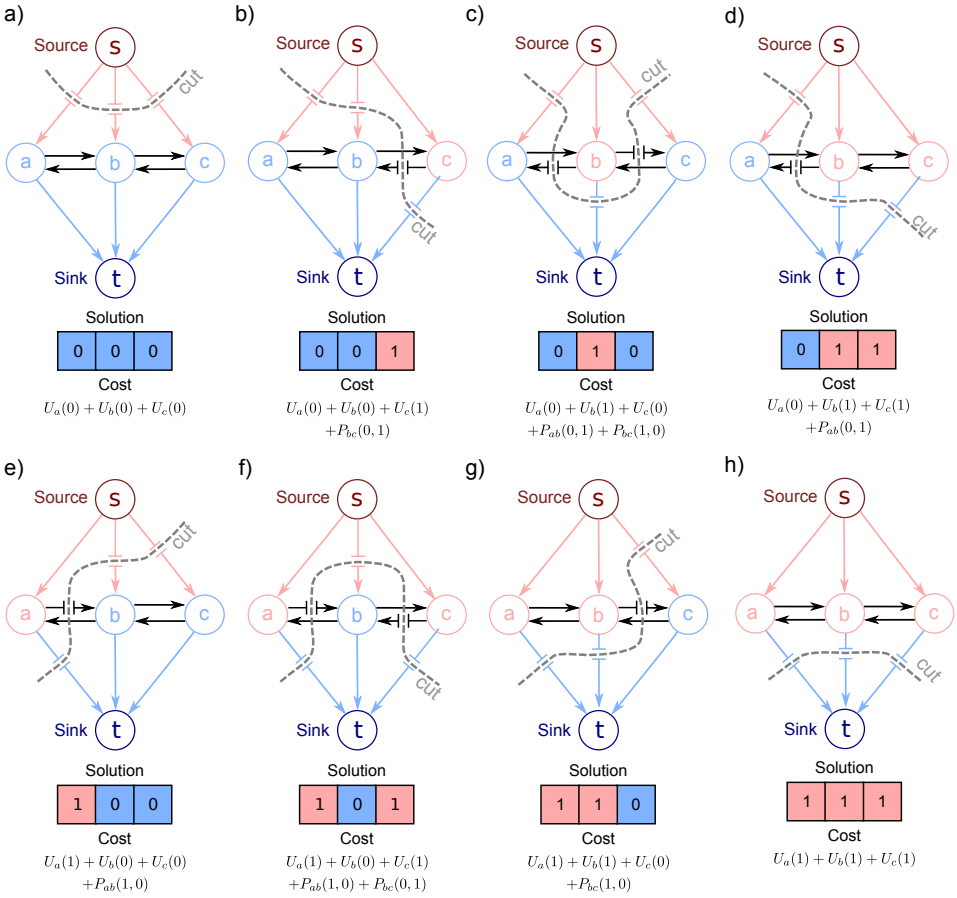


Figure 12.9 Eight possible solutions for three pixel example. When we set the costs as in figure 12.8, each solution has the appropriate cost. a) For example, the solution $(a = 0, b = 0, c = 0)$ requires us to cut edges $s{-}a, s{-}b, s{-}c$ and pay the cost $U_a(0) + U_b(0) + U_c(0)$. b) For the solution $(a = 0, b = 0, c = 1)$ we must cut edges $s{-}a, s{-}b, c{-}t$, and $c{-}b$ (to prevent flow through the path $s{-}c{-}b{-}t$). This incurs a total cost of $U_a(0) + U_b(0) + U_c(1) + P_{bc}(0, 1)$. c) Similarly, in this example with $(a = 0, b = 1, c = 0)$, we pay the appropriate cost $U_a(0) + U_b(1) + U_c(0) + P_{ab}(0, 1) + P_{bc}(1, 0)$. d-h) The other five possible configurations.

We attach the unary costs $U_n(0)$ and $U_n(1)$ to the edges from the pixel to the source and sink, respectively. If we cut the edge from the source to a given pixel (and hence assign $w_n = 0$), we pay the cost $U_n(0)$. Conversely, if we cut the edge from the pixel to the sink (and hence assign $w_n = 1$), we pay the cost $U_n(1)$.

We attach the pairwise costs $P_{mn}(1, 0)$ and $P_{mn}(0, 1)$ to the pairs of edges between adjacent pixels. Now if one pixel is attached to the source and the other to the sink, we pay either $P_{mn}(0, 1) = \theta_{01}$ or $P_{mn}(1, 0) = \theta_{10}$ as appropriate to sepa-

Problem 12.2

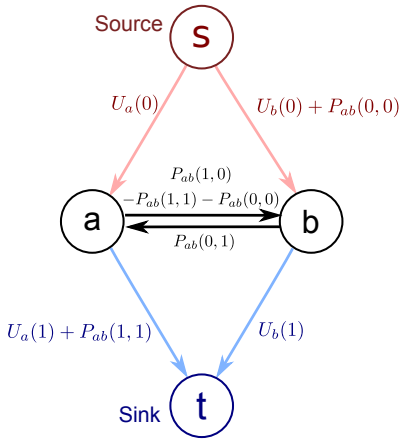


Figure 12.10 Graph structure for general (i.e., non-diagonal) pairwise costs. Consider the solution $(a = 0, b = 0)$. We must break the edges $s-a$ and $s-b$ giving a total cost of $U_a(0) + U_b(0) + P_{ab}(0,0)$. For the solution $(a = 1, b = 0)$ we must break the edges $a-t, a-b$ and $s-b$ giving a total cost of $U_a(1)+U_b(0)+P_{ab}(1,0)$. Similarly, the cuts corresponding to the solutions $(a = 0, b = 1)$ and $(a = 1, b = 1)$ on this graph have pairwise costs $P_{ab}(0,1)$ and $P_{ab}(1,1)$, respectively.

rate source from sink. The cuts corresponding to all eight possible configurations of the three pixel model and their costs are illustrated in figure 12.9.

Any cut on the graph in which each pixel is either separated from the source or the sink now has the appropriate cost from equation 12.12. It follows that the minimum cut on this graph will have the minimum cost and the associated labeling $\hat{w}_{1\ldots N}$ will correspond to the maximum a posteriori solution.

General pairwise costs

Now let us consider how to use the more general pairwise costs,

$$\begin{aligned} P_{mn}(0,0) &= \theta_{00} \qquad & P_{mn}(1,0) &= \theta_{10} \\ P_{mn}(0,1) &= \theta_{01} \qquad & P_{mn}(1,1) &= \theta_{11}. \end{aligned} \tag{12.13}$$

Problem 12.3

To illustrate this, we use an even simpler graph with only two pixels (figure 12.10). Notice that we have added the pairwise cost $P_{ab}(0,0)$ to the edge $s-b$. We will have to pay this cost appropriately in the configuration where $w_a = 0$ and $w_b = 0$. Unfortunately, we would also pay it in the case where $w_a = 1$ and $w_b = 0$. Hence, we subtract the same cost from the edge $a-b$, which must also be cut in this solution. By a similar logic, we add $P_{ab}(1,1)$ to the edge $a-t$ and subtract it from edge $a-b$. In this way, we associate the correct costs with each labeling.

Reparameterization

Algorithm 12.2

The preceding discussion assumed that the edge costs are all non-negative and can be valid capacities in the max-flow problem. If they are not, then it is not possible to compute the MAP solution. Unfortunately, it is often the case that they are negative; even if the original unary and pairwise terms were positive, the edge $a-b$ in figure 12.10 with cost $P_{ab}(1,0) - P_{ab}(1,1) - P_{ab}(0,0)$ could be negative. The solution to this problem is *reparameterization*.

The goal of reparameterization is to modify the costs associated with the edges in the graph in such a way that the MAP solution is not changed. In particular,

we will adjust the edge capacities so that every possible solution has a constant cost added to it. This does not change which solution has the minimum cost, and so the MAP labeling will be unchanged.

We consider two reparameterizations (figure 12.11). First, consider adding a constant cost α to the edge from a given pixel to the source, and the edge from the same pixel to the sink. Since any solution cuts exactly one of these edges, the overall cost of every solution increases by α. We can use this to ensure that none of the edges connecting the pixels to the source and sink have negative costs: we simply add a sufficiently large positive value α to make them all non-negative.

A more subtle type of reparameterization is illustrated in figure 12.11c. By changing the costs in this way, we increase the total cost of each possible solution by β. For example, in the assignment $(w_a = 0, w_b = 1)$ we must cut the links $s-a$, $b-a$ and $b-t$ giving a total cost of $U_a(0) + U_b(1) + P_{ab}(0,1) + \beta$.

Problem 12.4

Applying the reparameterization in figure 12.11c to the general construction in figure 12.10, we must ensure that the capacities on edges between pixel nodes are non-negative so that

$$\begin{aligned} \theta_{10} - \theta_{11} - \theta_{00} - \beta &\geq 0 \qquad (12.14) \\ \theta_{01} + \beta &\geq 0. \qquad (12.15) \end{aligned}$$

Adding these equations together, we can eliminate β to get a single inequality

$$\theta_{01} + \theta_{10} - \theta_{11} - \theta_{00} \geq 0. \qquad (12.16)$$

If this condition holds, the problem is termed *submodular* , and the graph can be reparameterized to have only non-negative costs. It can then be solved in polynomial time using the max-flow algorithm. If the condition does not hold, then this approach cannot be used and in general the problem is NP hard. Fortunately, the former case is common for vision problems; we generally favor smooth solutions where neighboring labels are the same and hence the costs θ_{01}, θ_{10} for labels differing are naturally greater than the costs θ_{00}, θ_{11} for the labels agreeing.

Figure 12.12 shows the MAP solutions to the binary denoising problem with an MRF prior, as we increase the strength of the cost for having adjacent labels that differ. Here we have assumed that the costs for adjacent labels being different are the same ($\theta_{01} = \theta_{10}$) and that there is no cost when neighboring labels are the same ($\theta_{00}, \theta_{11} = 0$); we are in the 'zero-diagonal' regimen. When the MRF costs are small, the solution is dominated by the unary terms and the MAP solution looks like the noisy image. As the costs increase, the solution ceases to tolerate isolated regions and most of the noise is removed. When the costs become larger, details such as the center of the '0' in '10' are lost, and eventually nearby regions are connected together. With very high pairwise costs, the MAP solution is a uniform field of labels: the overall cost is dominated by the pairwise terms from the MRF and the unary terms merely determine the polarity.

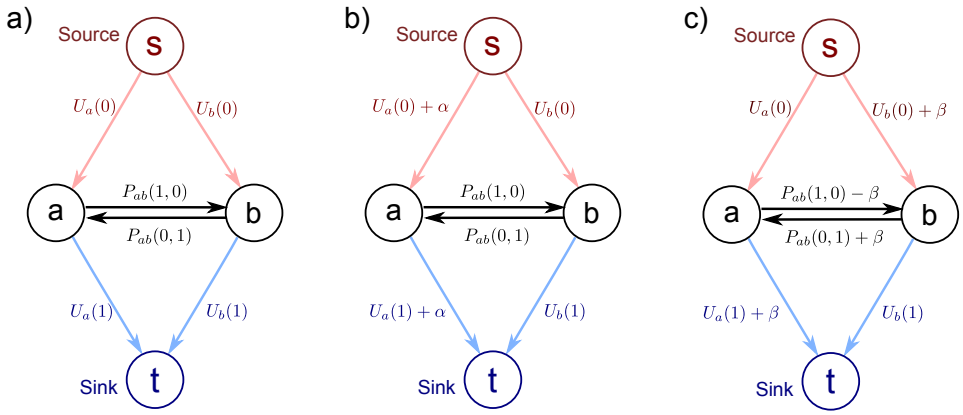


Figure 12.11 Reparameterization. a) Original graph construction. b) Reparameterization 1. Adding a constant cost α to the connections from a pixel vertex to both the source and sink results in a problem with the same MAP solution. Since we must cut either, but not both of these edges, every solution increases in cost by α, and the minimum cost solution remains the same. c) Reparameterization 2. Manipulating the edge capacities in this way results in a constant β being added to every solution and so the choice of minimum cost solution is unaffected.

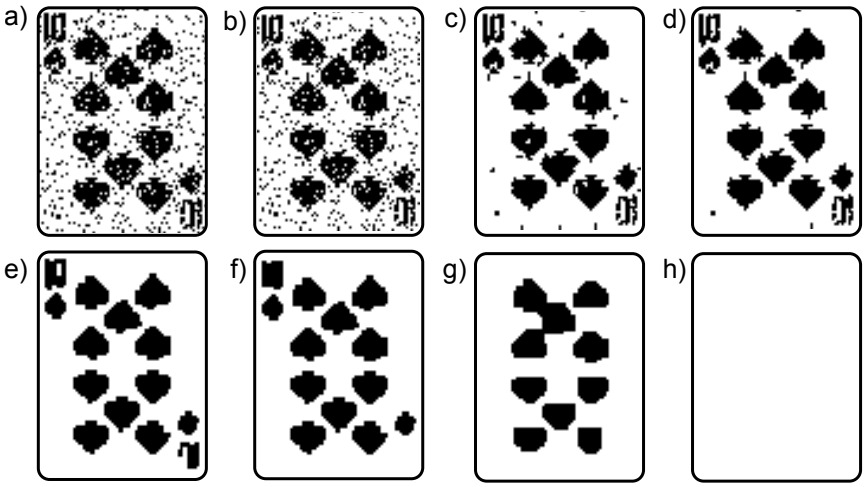


Figure 12.12 Denoising results. a) Observed noisy image. b-h) Maximum a posteriori solution as we increase zero-diagonal pairwise costs. When the pairwise costs are low, the unary terms dominate and the MAP solution is the same as the observed image. As the pairwise costs increase the image gets more and more smooth until eventually it becomes uniform.

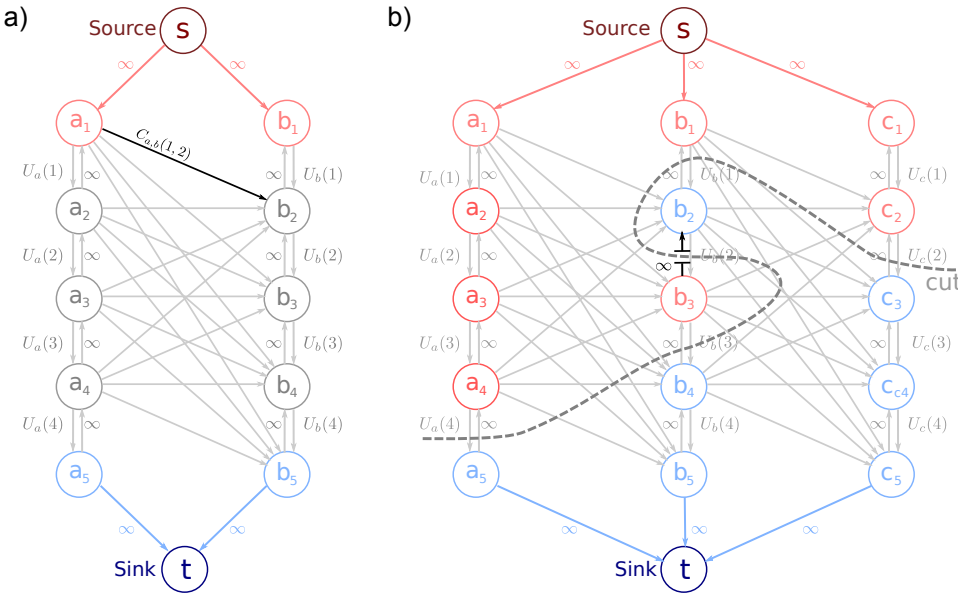


Figure 12.13 a) Graph setup for multi-label case for two pixels (a, b) and four labels $(1, 2, 3, 4)$. There is a chain of five vertices associated with each pixel. The four vertical edges between these vertices are assigned the unary costs for the four labels. The minimum cut must break this chain to separate source from sink, and the label is assigned according to where the chain is broken. Vertical constraint edges of infinite capacity run between the four vertices in the opposite direction. There are also diagonal edges between the i^{th} vertex of pixel a and the j^{th} vertex of pixel b with assigned costs $C_{ab}(i, j)$ (see text). b) The vertical constraint edges prevent solutions like this example with three pixels. Here, the chain of vertices associated with the central pixel is cut in more than one place and so the labeling has no clear interpretation. However, for this to happen, a constraint link must be cut, and hence this solution has an infinite cost.

12.3 MAP inference for multi-label pairwise MRFs

We now investigate MAP inference using MRF priors with pairwise connections when the world state w_n at each pixel can take multiple labels $\{1, 2, \ldots, K\}$. To solve the multi-label problem, we change the graph construction (figure 12.13a). With K labels and N pixels, we introduce $(K+1)N$ vertices into the graph.

Algorithm 12.3

For each pixel, the $K + 1$ associated vertices are stacked. The top and bottom of the stack are connected to the source and sink by edges with infinite capacity. Between the $K + 1$ vertices in the stack are K edges forming a path from source to sink. These edges are associated with the K unary costs $U_n(1) \ldots U_n(K)$. To separate the source from the sink, we must cut at least one of the K edges in this chain. We will interpret a cut at the k^{th} edge in this chain as indicating that the pixel takes label k and this incurs the appropriate cost of $U_n(k)$.

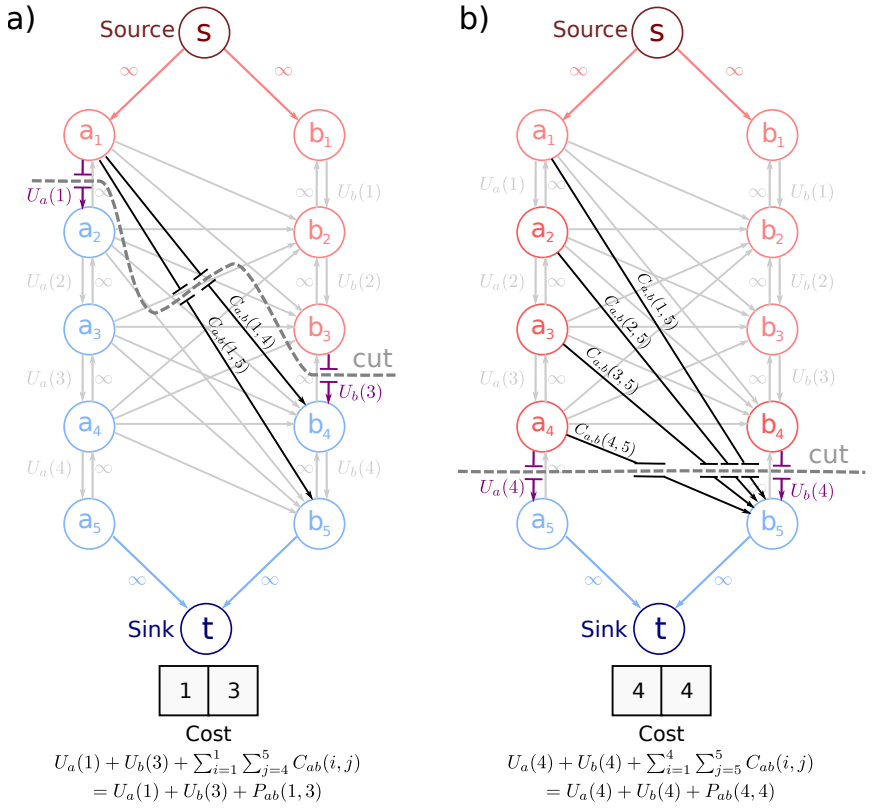


Figure 12.14 Example cuts for multi-label case. To separate the source and sink, we must cut all of the links that pass from above the chosen label for pixel a to below the chosen label for pixel b. a) Pixel a is set to label 1 and pixel b is set to label 3 meaning we must cut the links from vertex a_1 to nodes b_4 and b_5. b) Pixel a takes label 4 and pixel b takes label 4.

Problem 12.5
Problem 12.6
Problem **??**

To ensure that only a single edge from the chain is part of the minimum cut (and hence that each cut corresponds to one valid labeling), we add *constraint edges*. These are edges of infinite capacity that are strategically placed to prevent certain cuts occurring. In this case, the constraint edges connect the vertices backwards along each chain. Any cut that crosses the chain more than once must cut one of these edges and will never be the minimum cut solution (figure 12.13b).

In figure 12.13a, there are also diagonal inter-pixel edges from the vertices associated with pixel a to those associated with pixel b. These are assigned costs $C_{ab}(i,j)$, where i indexes the vertex associated with pixel a and j indexes the vertex associated with pixel b. We choose the edge costs to be

$$C_{ab}(i,j) = P_{ab}(i,j-1) + P_{ab}(i-1,j) - P_{ab}(i,j) - P_{ab}(i-1,j-1), \quad (12.17)$$

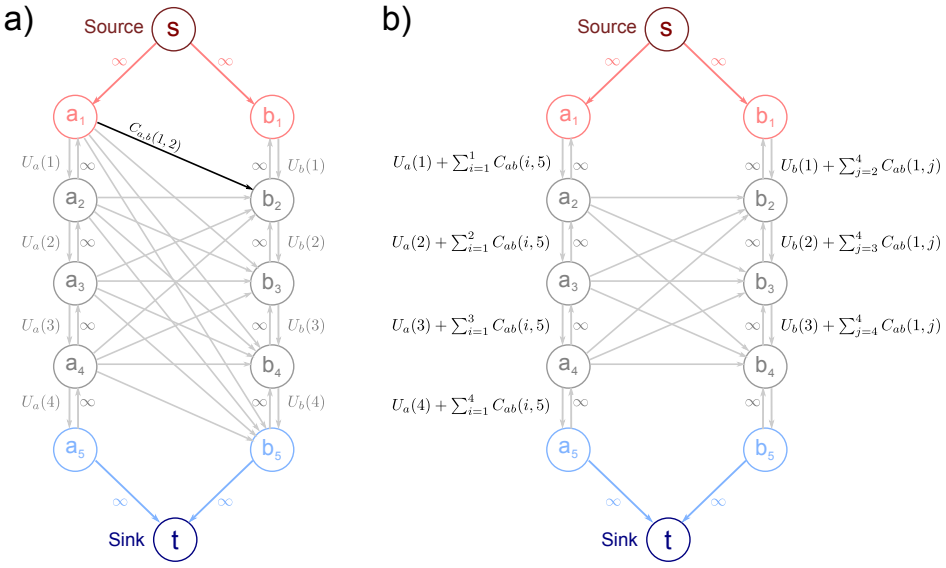


Figure 12.15 Reparameterization for multi-label graph cuts. The original construction (a) is equivalent to construction (b). The label at pixel b determines which edges that leave node a_1 are cut. Hence, we can remove these edges and add the extra costs to the vertical links associated with pixel b. Similarly, the costs of the edges passing into node b_5 can be added to the vertical edges associated with pixel a. If any of the resulting vertical edges associated with a pixel are negative, we can add a constant α to each: since exactly one is broken, the total cost increases by α, but the MAP solution remains the same.

where we define any superfluous pairwise costs associated with the non-existent labels 0 or $K+1$ to be zero, so that

$$\begin{aligned} P_{ab}(i,0) &= 0 \qquad P_{ab}(i,K+1) = 0 \qquad \forall\ i \in \{0\ldots K+1\} \\ P_{ab}(0,j) &= 0 \qquad P_{ab}(K+1,j) = 0 \qquad \forall\ j \in \{0\ldots K+1\}. \end{aligned} \tag{12.18}$$

When label I is assigned to pixel a and label J to pixel b, we must cut all of the links from vertices $a_1 \ldots a_I$ to the vertices $b_{J+1} \ldots b_{K+1}$ to separate the source from the sink (figure 12.14). So, the total cost due to the inter-pixel edges for assigning label I to pixel a and label J to pixel b is

Problem 12.7

$$\begin{aligned} \sum_{i=1}^{I}\sum_{j=J+1}^{K+1} C_{ab}(i,j) &= \sum_{i=1}^{I}\sum_{j=J+1}^{K+1} P_{ab}(i,j-1)+P_{ab}(i-1,j)-P_{ab}(i,j)-P_{ab}(i-1,j-1) \\ &= P_{ab}(I,J)+P_{ab}(0,J)-P_{ab}(I,K+1)-P_{ab}(0,K+1) \\ &= P_{ab}(I,J). \end{aligned} \tag{12.19}$$

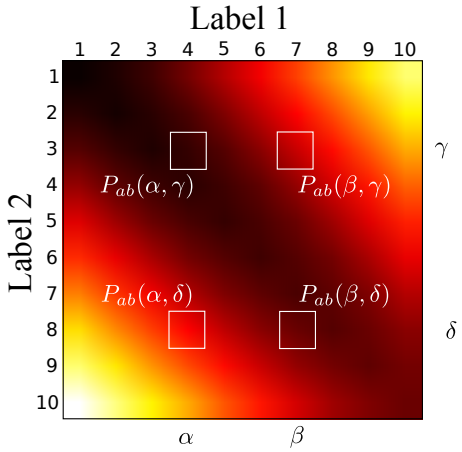


Figure 12.16 Submodularity constraint for multi-label case. Color at position (m,n) indicates pairwise costs $P_{ab}(m,n)$. For all edges in the graph to be positive, we require that the pairwise terms obey $P_{ab}(\beta,\gamma) + P_{ab}(\alpha,\delta) - P_{ab}(\beta,\delta) - P_{ab}(\alpha,\gamma) \geq 0$ for all $\alpha,\beta,\gamma,\delta$ such that $\beta > \alpha$ and $\delta > \gamma$. In other words, for any four positions arranged in a square configuration as in the figure, the sum of the two costs on the diagonal from top-left to bottom-right must be less than the sum on the off diagonal. If this condition holds, the problem can be solved in polynomial time.

Adding the unary terms, the total cost is $U_a(I) + U_b(J) + P_{ab}(I,J)$ as required.

Once more, we have implicitly made the assumption that the costs associated with edges are non-negative. If the vertical (intra-pixel) edge terms have negative costs, it is possible to reparameterize the graph by adding a constant α to all of the unary terms. Since the final cost includes exactly one unary term per pixel, every possible solution increases by α, and the MAP solution is unaffected.

The diagonal inter-pixel edges are more problematic. It is possible to remove the edges that leave node a_1 and the edges that arrive at b_{K+1} by adding terms to the intra-pixel edges associated with the unary terms (figure 12.15). These intra-pixel edges can then be reparameterized as described above if necessary. Unfortunately, we can neither remove nor reparameterize the remaining inter-pixel edges, so we require that

$$C_{ab}(i,j) = P_{ab}(i,j-1) + P_{ab}(i-1,j) - P_{ab}(i,j) - P_{ab}(i-1,j-1) \geq 0. \quad (12.20)$$

By mathematical induction, we get the more general result (figure 12.16),

$$P_{ab}(\beta,\gamma) + P_{ab}(\alpha,\delta) - P_{ab}(\beta,\delta) - P_{ab}(\alpha,\gamma) \geq 0, \quad (12.21)$$

where $\alpha,\beta,\gamma,\delta$ are any four values of the state y such that $\beta > \alpha$ and $\delta > \gamma$. This is the multi-label generalization of the submodularity condition (equation 12.16). An important class of pairwise costs that are submodular are those that are convex in the absolute difference $|w_i - w_j|$ between the labels at adjacent pixels (figure 12.17a). Here, smoothness is encouraged; the penalty becomes increasingly stringent as the jumps between labels increase.

12.4 Multi-label MRFs with non-convex potentials

Unfortunately, convex potentials are not always appropriate. For example, in the denoising task we might expect the image to be piecewise smooth: there are smooth

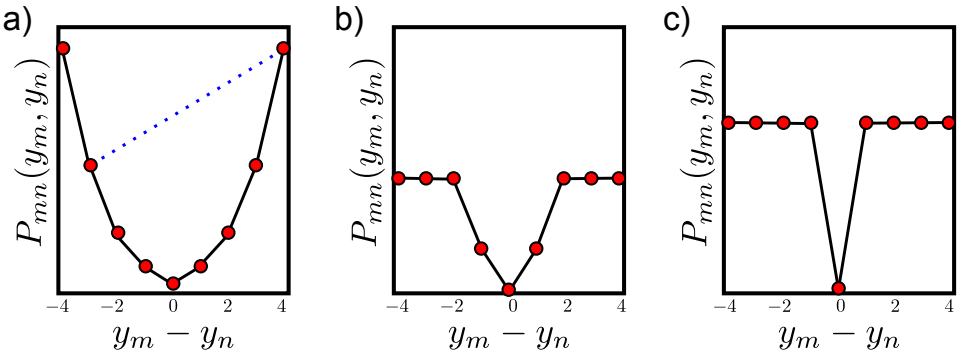


Figure 12.17 Convex vs. non-convex potentials. The method for MAP inference for multi-valued variables depends on whether the costs are a convex or non-convex function of the difference in labels. a) Quadratic function (convex), $P_{mn}(w_m, w_n) = \kappa(w_m - w_n)^2$. For convex functions, it is possible to draw a chord between any two points on the function without intersecting the function elsewhere (e.g., dotted blue line). b) Truncated quadratic function (non-convex), $P_{mn}(w_m, w_n) = \min(\kappa_1, \kappa_2(w_m - w_n)^2)$. c) Potts model (non-convex), $P_{mn}(w_m, w_n) = \kappa(1 - \delta(w_m - w_n))$.

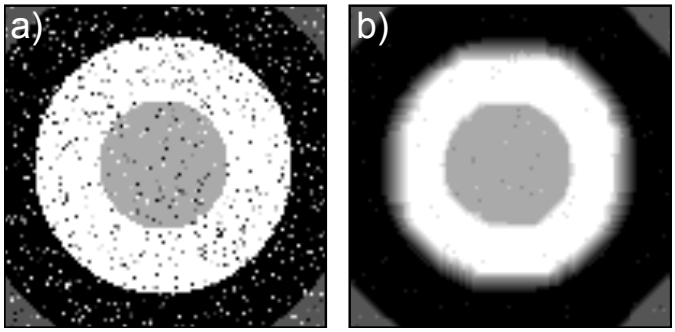


Figure 12.18 Denoising results with convex (quadratic) pairwise costs. a) Noisy observed image. b) Denoised image has artiWfacts where there are large intensity changes in the original image. Convex costs imply that there is a lower cost for a number of small changes rather than a single large one.

regions (corresponding to objects) followed by abrupt jumps (corresponding to the boundaries between objects). A convex potential function cannot describe this situation because it penalizes large jumps much more than smaller ones. The result is that the MAP solution smooths over the sharp edges, changing the label by several smaller amounts rather than one large jump (figure 12.18).

To solve this problem, we need to work with interactions that are non-convex in the absolute label difference, such as the truncated quadratic function or the

Problem 12.8

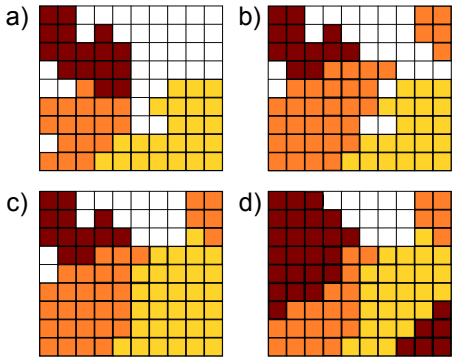


Figure 12.19 The alpha-expansion algorithm breaks the problem down into a series of binary sub-problems. At each step, we choose a label α and we *expand*: for each pixel we either leave the label as it is, or replace it with α. This sub-problem is solved in such a way that it is guaranteed to decrease the multilabel cost function. a) Initial labeling. b) Orange label is expanded: each label stays the same or becomes orange. c) Yellow label is expanded. d) Red label is expanded.

Potts model (figures 12.17b-c). These favor small changes in the label and penalize large changes equally or nearly equally. This reflects the fact that the exact size of an abrupt jump in label is relatively unimportant. Unfortunately, these pairwise costs do not satisfy the submodularity constraint (equation 12.21). Here, the MAP solution cannot in general be found exactly with the method described previously, and the problem is NP-hard. Fortunately, there are good approximate methods for optimizing such problems, one of which is the alpha-expansion algorithm.

12.4.1 Inference: alpha-expansion

Algorithm 12.4

The *alpha-expansion* algorithm works by breaking the solution down into a series of binary problems, each of which can be solved exactly. At each iteration, we choose one label value α, and for each pixel, we consider either retaining the current label, or switching it to α. The name alpha-expansion derives from the fact that the space occupied by label α in the solution expands at each iteration (figure 12.19). The process is iterated until no choice of α causes any change. Each expansion move is guaranteed to lower the overall objective function, although the final result is not guaranteed to be the global minimum.

For the alpha-expansion algorithm to work, we require that the edge costs form a metric. In other words, we require that

$$\begin{aligned} P(\alpha,\beta) &= 0 \quad \Leftrightarrow \alpha = \beta \\ P(\alpha,\beta) &= P(\beta,\alpha) \geq 0 \\ P(\alpha,\beta) &\leq P(\alpha,\gamma) + P(\gamma,\beta). \end{aligned} \tag{12.22}$$

These assumptions are reasonable for many applications in vision, and allow us to model non-convex priors.

In the alpha-expansion graph construction (figure 12.20), there is one vertex associated with each pixel. Each of these vertices is connected to the source (representing keeping the original label or $\overline{\alpha}$) and the sink (representing the label α). To separate source from sink, we must cut one of these two edges at each pixel. The choice of edge will determine whether we keep the original label or set it to