Machine Learning
A Probabilistic Approach

David Barber
http://www.idiap.ch/∼barber

Contents

1 Introduction

There are many motivations as to why one might want a machine to "learn" from data. By "learn", one has in mind applications that would be hard to program in a traditional manner, such as the task of face recognition. Formally specifying why you recognise an image as John's face may be extremely difficult. An alternative is to give example images of John's face and examples that are not John's face and let a machine "learn" what it is that differentiates John's face from other faces in the database.

The basic paradigm is to postulate a model for solving the problem, and set any parameters of this model based on the available data. Additionally using prior knowledge about faces to give hints to the machine may make learning easier and our confidence in the learned model greater. As we will see, probability theory enables us to combine our prior beliefs about appropriate models and the data in a principled way. In recent years, "machine learning" has grown rapidly, overlapping strongly with more traditional disciplines such as statistics, computer science and artificial intelligence. Applications abound in computer vision, bioinformatics, speech recognition, robotics, computer gaming, and many more.

Fundamentally, the connection between probability and machine learning stems from the idea that probabilistic models enable us to form a compact description of complex phenomena underlying the generation of the data. Graphical models, upon which much of this book is based, are simply ways of depicting the independence assumptions behind a probabilistic model. They are useful in modelling since they provide an elegant framework to therefore graphically express basic independence assumptions about the processes generating the data. This is useful since the calculus of probability will transfer to graph theoretic operations and algorithms, many of which have deep roots in computer science and related areas.

Probability has not always played a large role in the development of machine learning – areas such as neural networks, and expert systems were typically non-probabilistic. The relatively recent use of probability in machine learning has helped clarify the subject and lead to a large expansion in ideas and opened up many new avenues for the future.

Some may lament that this has inevitably brought statistics and machine learning much closer together and, indeed, there is now a community that comfortably straddles both disciplines. The probabilistic graphical model connection has been particularly beneficial in this sense, enabling also deep connections between models and techniques in machine learning, statistics, physics, game theory and information theory to be much more transparent.

This book is intended as a (non-rigorous) introduction to machine learning, probabilistic graphical models and their applications. Formal proofs of theorems are generally omitted. These notes are formed from the basis of lectures given to both undergraduate and graduate students at Aston University, Edinburgh University, and EPF Lausanne.

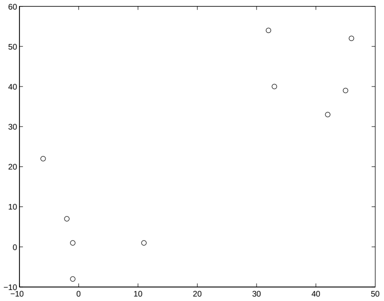


Figure 1.1: In unsupervised learning, we wish to make a compact description of the data. Here, a natural description would be that the data lies in two clusters.

1.1 Machine Learning

Machine learning is traditionally split into two main areas: supervised and unsupervised learning. In supervised learning, one typically wishes to learn models that will make accurate predictions in on novel data. In unsupervised learning, more typically, one desires to make a useful description of the data. More recently, in areas such as semi-supervised learning, the distinction between these areas is becoming more blurred.

1.1.1 Unsupervised Learning

A baby processes a mass of initially confusing sensory data. After a while the baby begins to understand her environment in the sense that novel sensory data from the same environment is familiar or expected. When a strange face presents itself, the baby recognises that this is not familiar and may be upset. The baby has learned a representation of the familiar and can distinguish the expected from the unexpected, *without an explicit supervisor to say what is right or wrong*. Unsupervised learning just addresses how to model an environment. *Clustering* is an example of unsupervised learning, whereby the aim is to partition a set of data into clusters. For example, given a set of questionnaire responses, find clusters whereby in each cluster the responses are ‘similar’. This area is also sometimes called *descriptive modelling*, where we just wish to fit a model which describes succinctly and accurately the data in the database.

Clustering

Descriptive modelling

For example, here are some points:

x_1	-2	-6	-1	11	-1	46	33	42	32	45
x_2	7	22	1	1	-8	52	40	33	54	39

This is an example of *unlabelled* data. We can visualise this data by plotting it in 2 dimensions: By simply eye-balling the data, we can see that there are two apparent clusters here, one centred around (0,0) and the other around (35,35). A reasonable model to describe this data might therefore be to describe it as two clusters, centred at (0,0) and (35,35), each with a variance (spread) of around 1.

1.1.2 Supervised Learning

I’m fond of the following story :

“A father decides to teach his young son what a sports car is. Finding it difficult to explain in words, he decides to try to explain by examples. They stand on a motorway bridge and, as each car passes underneath, the father cries out ‘that’s a sports car!’ when a sports car passes by. After ten minutes, the father asks his son if he’s got it. The son says, ‘sure, it’s easy’. An old red VW Beetle passes by, and the son shouts – ‘that’s a sports car!’. Dejected, the father asks – ‘why do you say that?’. ‘Because all sports cars are red!’, replies the son.”

This story is an example of supervised learning. Here the father is the supervisor, and his son plays the role of the ‘learner’. Based on the labelled examples from the father, the child attempts to build a model of what distinguishes a sports car from non-sports cars. A nice point about this story is that you can’t expect miracles – unless you explicitly give extra information, learning from examples may not always give you what you might hope for. On the other hand, if they had been there the whole week, probably the son would have learned a reasonably good model of a sports car, and helpful hints by the father would be less important. It’s also indicative of the kinds of problems typically encountered in machine learning in that it is not really clear anyway what a sports car is – if we knew that, then we wouldn’t need to go through the process of learning!

Predictive modelling

We typically have a training set of *labelled* data, for example, here are some data

nationality	British	Dutch	Taiwanese	British
height(cm)	175	195	155	165
sex	m	m	f	f

We might have a large database of such entries. A supervised learning problem might be: given a new, previously unseen (nationality,height) pair, predict the sex of the person. For example, given that a person is Taiwanese and 167cm tall, are they going to be male or female? In this case we see the training data as a collection of (input,output) pairs, where the output or label has been given by a ‘supervisor’. Ultimately, we wish to form a mapping from the inputs to the output (possibly more than one output) that accurately describes the label/output given the inputs. Ideally, we would like our model to *generalise* well (predict accurately) novel test data not seen previously during the model building process.

Uncertainty

The above data provides a good example to motivate our later ideas about probability/uncertainty – there is clearly not going to be absolute certainty about our predictions in this case since there are always going to be tall females and shorter males that will make classifying a novel person an inexact science. However, we may be able to infer what is the probability that a novel person is male, given our trained model. In practice, uncertainty often plays a major role in machine learning, and we need to use a framework that can handle this.

Supervised learning problems traditionally come in two flavours, classification and regression.

Classification

Given a set of inputs, predict the class (one of a finite number of discrete labels). Normally, the class is ordinal (there is no intrinsic information in the class label). For example, given an image of a handwritten digit, predict whether it is 0,1,2,3,4,5,6,7,8 or 9. This would be a 10-class classification problem. Many problems involve binary classes (you can always convert a multi-class problem into a set of binary class problems – though this is not always natural or desirable). For binary classes, there is usually no information as to whether we say the data are labelled as class 0 or 1, or alternatively as class 1 or 2. For example, the sports-car classification problem would have been the same if the father said '1' or '0' when the car passing by was a sports car or not. A great deal of problems in the machine learning arena are classification problems. Uncertainty will ultimately play a key role in any real world application. Can we really say that Mr Smith will definitely default on his loan? This may seem a very strong statement if there is little obvious difference between the attributes of Mr Smith and Mr Brown.

Regression Given a set of inputs, predict the output (a continuous value). For example, given historical stock market data, predict the course of the FTSE for tomorrow.

Reinforcement Learning Reinforcement learning is a kind of supervised learning in which the supervisor provides rewards for actions which improve a situation and penalties for deleterious actions.

Semi-Supervised Learning

In machine learning, a common scenario is to have both labelled and unlabelled training data available. For example, it may be that we have access to many images of faces; however, only a small number of them may have been labelled as instances of known faces. In semi-supervised learning, one tries to use the unlabelled data to make a better classifier than that based on the labelled data alone.

Unbalanced Data

In some applications, the amount of labelled training data available for one class far exceeds that of the other. For example, of one wishes to build a cancerous/bengin classifier for mammograms, the vast majority of the labelled examples are benign, with only a small number labelled as cancerous. However, in testing on a novel example, it's clearly important to classify correctly the cancerous cases. Incorrectly classifying a benign mammogram as cancerous may not be catastrophic.

This introduces the concept of *loss functions* such as the matrix $L(\text{Predicted Class}, \text{Correct Class})$ which will quantify how correct or incorrect decisions will be weighted. Such loss functions can also be incorporated into the learning Framework.

1.2 Supervised Learning Approaches

Consider trying to make a system that can distinguish between male and female faces. Each image is represented as a real-valued vector x, formed by concatenating the pixels of the image into a long vector. We have a database of P images, $x^\mu, \mu = 1, \ldots, P$), along with a label $c^\mu \in \{0, 1\}$ stating if the image is a male ($c = 0$) or female ($c = 1$) face. We wish to make a model of the image,sex environment, $p(x, c)$.

Generative Approach

In a generative approach, we define a model for generating data v belonging to a particular class $c \in 1, \ldots, C$ in terms of a distribution $p(v|c)$. Here, v will correspond, say to the image vector. The class c will be say male or female.

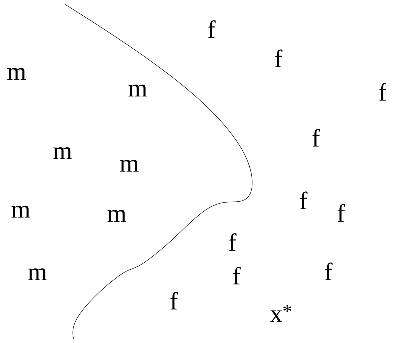


Figure 1.2: Here, each point in this space represents a high dimensional vector x, which has an associated class label, either Male or Female. The point x^* is a new point for which we would like to predict whether this should be male or female. In the generative approach, a Male model would be produced which would ideally generate data which would be similar to the 'm' points. Similarly, another model, the Female model should ideally generate points that are similar to the 'f' points above. We then use Bayes' rule to calculate the probability $p(male|x^*)$ using the two fitted models, as given in the text. In the discriminative case, we directly make a model of $p(male|x^*)$, which cares less about how the points 'm' or 'f' are distributed, but more about whether or not there is a simple description of a boundary which can separate the two classes, as given by the line.

For each class c we train a separate model $p(v|c)$ with associated parameters Θ_c by maximising the likelihood of the observed signals for that class. We then use Bayes' rule to assign a novel point v^* to a certain class c according to

$$p(c|v^*) = \frac{p(v^*|c)p(c)}{p(v^*)}. \tag{1.2.1}$$

That model c with the highest posterior probability $p(c|v^*)$ is designated the predicted class.

Advantages : In general, the potential attraction of a generative approach is that prior information about the structure of the data is often most naturally specified through the generative model $p(v|c)$.

Disadvantages : A potential disadvantage of the generative approach is that it does not directly target the central issue which is to make a good classifier. That is, the goal of generative training is to model the observation data v as accurately as possible, and not to model the class distribution. If the data v is complex, or high-dimensional, it may be that finding a suitable generative data model is a difficult task. Furthermore, since each generative model is separately trained for each class, there is no competition amongst the models to explain the data. In particular, if each class model is quite poor, there may be little confidence in the reliability of the prediction. In other words, training does not focus explicitly on the differences between mental tasks, but rather on accurately modelling the data distributions from each associated class.

Discriminative Approach

In a discriminative probabilistic approach we define a single model $p(c|v)$ common to all classes. The parameters Θ of this model are trained to maximise the probability of the class label c. This is in contrast to the generative approach above which models the data, and not the class. Given novel data v^*, we then directly calculate the probabilities $p(c|v^*)$ for each class c, and assign v^* to the class with the highest probability.

Advantages : A clear potential advantage of this discriminative approach is that it directly addresses the issue we are interested in solving, namely making a classifier. We are here therefore modelling the discrimination boundary, as opposed to the data distribution in the generative approach. Whilst the data from each class may be distributed in a complex way, it could be that the discrimination boundary between then is relatively easy to model.

Disadvantages : A potential drawback of the discriminative approach is that they are usually trained as 'black-box' classifiers, with little prior knowledge of how the signal is formed built into the model.

In principle, one could use a generative description, $p(v|c)$, building in prior information, and use this to form a joint distribution $p(v, c)$, from which a discriminative model $p(c|v)$ may be formed, using Bayes rule. Subsequently the parameters Θ_c, $c = 1, \dots C$ for this model could be found by maximising the discriminative class probability. This approach is rarely taken in the machine learning literature since the resulting functional form of $p(c|v)$ is often complex and training is difficult.

What are the basic assumptions of machine learning?

Arguably all machine learning approaches are based on some notion of smoothness or regularity underlying the mechanism that generated the observed data. Roughly speaking : if two datapoints are close neighbours, they are likely to behave similarly.

The general procedure will be to postulate some model and then adjust it's parameters to best fit the data. For example in a regression problem, we may think that the data $\{(x^\mu, y^\mu), \mu = 1, \dots, P\}$, where x is an input and y an output, is well modelled by the function $y = wx$, and our task is to find an appropriate setting of the parameter w. An obvious way to do this is to see how well the current model predicts the training data that we have, and then to adjust the parameter w to minimise the errors that our model makes on predicting the data. This general procedure will usually involve therefore optimisation methods, usually in high dimensional spaces (although the above is a one-dimensional example).

The traditional non-probabilistic approaches are typically based on errors functions based on squared loss between predictions and desired outputs. In probabilistic approaches the equivalent error functions are derived from criteria such as maximal likelihood.

Noise, overfitting and Generalisation

In the case that there is noise on the data (sometimes, the father might be inconsistent in his labelling of sports cars, or there might be essentially random perturbations on the FTSE index), we don't want to model this noise. That is, we have to be careful to make sure that our models only capture the underlying process that we are truly interested in, and not necessarily the exact details of the training data. If we have an extremely flexible model, it may overfit noisy training data be a very poor predictor of future novel inputs (that is, it will generalise poorly). This is very important topic and central to machine learning. We shall return to this in a later chapter.

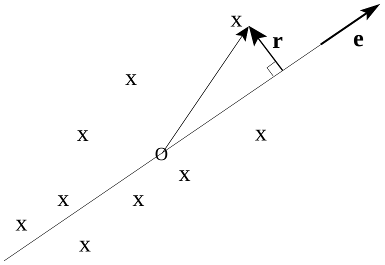


Figure 1.3: Learning the best line fit through a set of data. For zero mean data, we want to find the best unit length vector e such that when data points are projected onto the direction e, the residual r is minimised.

Non-parametric models

Throughout this introduction, we've made the assumption that our models will depend on parameters, and that the task of machine learning is to set these parameters appropriately to fit the data well. In statistics, there are the notions of parameteric and non-parameteric models. A parametric model is one in which the underlying model generating the data has a pre-described parameterised functional form – for example the data y is described by a linear function of x, namely $y = wx$, where w is the parameter to be learned from the data.

In a non-parametric model, there is no pre-described parameterised functional form directly for the model. For example, one might say that the model for the data $(x^\mu, y^\mu), \mu = 1, \ldots P$, is assumed to be modelled by P Gaussians, with a Gaussian centred on (x^μ, y^μ). Predictions for a novel x are then found from the marginal distribution $p(y|x)$. This is non-parametric in the sense that the functional form of the model will depend on the number of datapoints available – it is not fixed a priori. One of the most useful non-parametric models are Gaussian Processes, which we shall encounter later.

Readers may find this categorisation somewhat confusing since all models have some parameters which we will tune according to the data.

1.2.1 Whence optimisation? Fitting a straight line

This section is rather more technical. It is provided to demonstrate that vector algebra, eigenvectors and Lagrange multipliers are central to even the simplest data modelling problems.

Optimisation plays a central role in learning from data. For motivation, consider the following simple problem of trying to explain data by a straight line.

Let $x^\mu, \mu = 1, \ldots P$ be a set of P data points and each x^μ is a two dimensional vector. We wish to "learn" the best straight line approximation to the data.

Each datapoint x can be expressed in terms of component parallel to the desired direction e, and one orthogonal to e:

$$x = (x \cdot e)e + r \tag{1.2.2}$$

where $e \cdot e = 1$. Hence the squared length of the residual vector r is

$$r^2 = (x - (x \cdot e)e)^2 = x \cdot x - 2(x \cdot e)(x \cdot e) + (x \cdot e)(x \cdot e) = x \cdot x - (x \cdot e)^2 \tag{1.2.3}$$

Thus, finding e that minimises the residual is equivalent to finding e that maximises $(x \cdot e)^2$. If we wish to find the best direction for the whole dataset, the problem can be stated as

$$\text{Find } e \text{ to maximise } \sum_{\mu=1}^{P} (x_\mu \cdot e)^2 \text{ such that } e \cdot e = 1 \tag{1.2.4}$$

Thus, this very simple problem of learning a straight line requires us to optimise a quadratic function subject to constraints.

We can restate the problem as

$$\text{Find } e \text{ to maximise } e^{\mathsf{T}} A e \text{ such that } e \cdot e = 1 \tag{1.2.5}$$

where A is the correlation matrix of the data, $A = (1/P)\sum_\mu x_\mu x_\mu^{\mathsf{T}}$. One way to solve this is to use the Lagrangian method:

$$L = e^{\mathsf{T}} A e + \lambda(1 - e^{\mathsf{T}} e) \tag{1.2.6}$$

The gradient ∇_e of L gives the condition $Ae = \lambda e$. That is, e is an eigenvector of the correlation matrix. The eigenvector must be normalised to satisfy the constraint. For a two dimensional problem there will, in general, be two eigenvectors – the one which corresponds to the largest eigenvalue is the solution that we require (since the error $e^{\mathsf{T}} A e = \lambda$ at the optimum).

1.3 Problems

Exercise 1 *Calculate the length of the vector* $(1, -1, 2)$.

Exercise 2 *Show that* $\sum_i w_i x_i = |w||x| \cos\theta$, *where* θ *is the angle between the two vectors* w *and* x.

Exercise 3 *Let* A *and* v *be defined as*

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1 & 3 \\ 1 & 1 & -3 \\ 3 & -3 & -3 \end{pmatrix} \qquad \boldsymbol{v} = \begin{pmatrix} \mathbf{1} \\ \mathbf{-1} \\ \mathbf{-2} \end{pmatrix}$$

Calculate Av. *Is* v *an eigenvector of* A, *and if so what is the corresponding eigenvalue?*

Exercise 4 *Partial derivatives. Find the partial derivatives of the function* $f(x, y, z) = (x + 2y)^2 \sin(xy)$.

Exercise 5 *Probability. Lois knows that on average radio station RANDOM-FM plays 1 out of 4 of her requests. If she makes 3 requests, what is the probability that at least one request is played?*

Exercise 6 *Let* X *be distributed according to a uniform distribution, i.e.* $f(x) = 1$ *for* $0 \le x \le 1$, *and* 0 *otherwise. Show that* X *has mean* $\frac{1}{2}$ *and variance* $\frac{1}{12}$.

Exercise 7 *Find the (unconstrained) minimum of the function*

$$\boldsymbol{f(x) = \frac{1}{2} x^{\mathsf{T}} A x - b^{\mathsf{T}} x}$$

where A is a positive definite symmetric matrix.

Find the minimum of $f(x)$ along the line $x = a + tv$ where t is a real parameter.

Discuss how to find the minimum of $f(x)$ subject to $x^{\mathsf{T}}x = 1$.

Exercise 8 *Calculate explicitly the unconstrained minimum in the case that*

$$\boldsymbol{A} = \begin{pmatrix} 1 & 2 \\ 2 & 5 \end{pmatrix} \qquad\qquad \boldsymbol{b} = \begin{pmatrix} \mathbf{1} \\ \mathbf{-1} \end{pmatrix}$$

Exercise 9 *Let A be the matrix given in question 3, and x be the vector*

$$x = \begin{pmatrix} x \\ y \\ z \end{pmatrix}.$$

Consider the quadratic form $f(x) = \frac{1}{2}x^{\mathsf{T}}Ax$. *Calculate the partial derivatives $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$ and $\frac{\partial f}{\partial z}$, and the matrix of second derivatives*

$$\begin{pmatrix} f_{xx} & f_{xy} & f_{xz} \\ f_{yx} & f_{yy} & f_{yz} \\ f_{zx} & f_{zy} & f_{zz} \end{pmatrix},$$

where

$$f_{xy} = \frac{\partial^2 f}{\partial x \partial y}.$$

Hessian *The matrix of second derivatives is known as the* Hessian *matrix.*

Exercise 10 *Consider the integral*

$$I = \int_{\infty}^{\infty} e^{-\frac{1}{2}x^2} dx$$

By considering

$$I^2 = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-\frac{1}{2}(x^2+y^2)} dxdy$$

and converting to polar coordinates, r, θ, show that

$$I = \sqrt{2\pi}$$

Exercise 11 *Generalise the concept of fitting a straight line in two dimensions to fitting a hyperplane in N dimensions.*

1.4 Solutions

I. Machine Learning : The basics

2 Nearest Neighbour Classification

In classification, we have a training set of data which contains both attributes x and a class label c. For example, the vector x might represent an image of a digit, and c labels which digit it is, $c \in \{0, 1, \dots, 9\}$. A dataset D of P training datapoints is given by $D = \{x^\mu, c^\mu\}$, $\mu = 1, \dots, P$. The aim is, given a novel x, to return the "correct" class $c(x)$. A simple strategy we adopt in this chapter can be very loosely stated as:

'Just say whatever your neighbour says!'

Vectors x which are "similar" should have the same class label

This is a kind of smoothness assumption. In this chapter, we won't explicitly construct a 'model' of the data in the sense that we could generate fake representative data with the model. It is possible, to come up with a model based on the above neighbourhood type idea and we will see how to do this when we learn about density estimation in a later chapter.

What does 'similar' mean?

The key word in the above strategy is 'similar'. Given two vectors x and y representing two different datapoints, how can we measure similarity? Clearly, this would seem to be rather subjective – two datapoints that one person thinks are 'similar' may be to someone else dissimilar.

The dissimilarity function $d(x, y)$

Usually we define a function $d(x, y)$, symmetric in its arguments ($d(x, y) = d(y, x)$) that measures the dissimilarity between the datapoints x and y.

It is common practice to adopt a simple measure of dissimilarity based on the *squared euclidean distance* $d(x, y) = (x - y)^T(x - y)$ (often more conveniently written $(x - y)^2$) between the vector representations of the datapoints. There can be problems with this but, in general, it's not an unreasonable assumption. However, one should bear in mind that more general dissimilarity measures can, and often are used in practice.

Machine Learning's "Dirty Secret"

Machine learning typically hinges on a sense of smoothness in the data. Some might consider this a "dirty secret", but it's essentially the fundamental assumption upon which most machine learning algorithms are based. Machine learners spend much of their time constructing appropriate measures of smoothness for the problem they have at hand. Nearest neighbour methods are a good starting point in many applications, since they are intuitive and easy to program. Simple methods like NN are a good starting point to try to understand the problem.

2.1 Nearest Neighbour

To classify a new vector x, given a set of training data $(x^\mu, c^\mu), \mu = 1, \dots, P$:

1. Calculate the dissimilarity of the test point x to each of the stored points, $d^\mu = d(x, x^\mu)$.

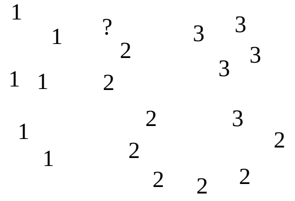


Figure 2.1: In nearest neighbour classification, a new vector with an unknown label, ?, is assigned the label of the vector in the training set which is nearest. In this case, the vector will be classified as a 2.

2. Find the training point x^{μ^*} which is 'closest' to x by finding that μ^* such that $d^{\mu^*} < d^{\mu}$ for all $\mu = 1, \dots, P$.
3. Assign the class label $c(x) = c^{\mu^*}$.

In the case that there are two or more 'equidistant' (or equi-dissimilar) points with different class labels, the most numerous class is chosen. If there is no one single most numerous class, we can use the K-nearest-neighbours case described in the next section.

The decision boundary

Voronoi Tessellation

In general, the decision boundary is the boundary in input space such that our decision as to the class of the input changes as we cross this boundary. In the nearest neighbour algorithm above based on the squared euclidean distance, the decision boundary is determined by the lines which are the perpendicular bisectors of the closet training points with different training labels, see fig(2.2). This is called a Voronoi tessellation.

2.1.1 Problems with Nearest Neighbours

The nearest neighbours algorithm is extremely simple yet rather powerful, and used in many applications. There are, however, some potential drawbacks:

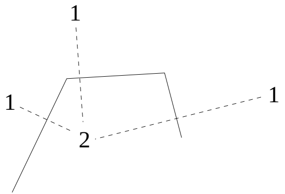


Figure 2.2: The decision boundary for the nearest neighbour classification rule is piecewise linear with each segment corresponding to the perpendicular bisector between two datapoints belonging to different classes.

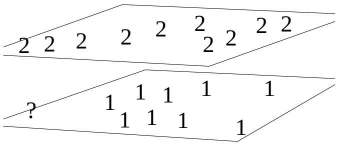


Figure 2.3: Consider data which lie close to (hyper)planes. The Euclidean distance would classify ? as belonging to class 2 – an undesirable effect.

How should we measure the distance between points? Typically one uses the euclidean square distance, as given in the algorithm above. This may not always be appropriate. Consider a situation such as in fig(2.3), in which the euclidean distance leads to an undesirable result. If we use the Euclean distance, $(x-y)^T(x-y)$ then the distance between the orthogonally transformed vectors Mx and My (where M^TM is the identity matrix) remains the same. (This is not true for the Mahalanobis distance). Since classification will be invariant to such transformations, this shows that we do not make a sensible model of how the data is generated – this is solved by density estimation methods – see later chapter.

Invariance to linear transformation

Mahalanobis Distance

The Mahalanobis distance $(x-y)^TA^i(x-y)$ where usually A^i is the inverse covariance matrix of the data from class i can overcome some of these problems. I think it's better to use density estimation methods.

Data Editing

In the simple version of the algorithm as explained above, we need to store the whole dataset in order to make a classification. However, it is clear that, in general, only a subset of the training data will actually determine the decision boundary. This can be addressed by a method called data editing in which datapoints which do not affect (or only very slightly) the decision boundary are removed from the training dataset.

Dimension Reduction

Each distance calculation could be quite expensive if the datapoints are high dimensional. Principal Components Analysis (see chapter on linear dimension reduction) is one way to address this, by first replacing each high dimensional datapoint x^μ with it's low dimensional PCA components vector p^μ. The Euclidean distance of the of two datapoints $\left(x^a-x^b\right)^2$ is then approximately given by $\left(p^a-p^b\right)^2$ – thus we need only to calculate distance among the PCA representations of data. This can often also improve the classification accuracy.

Sensitivity to outliers

An outlier is a “rogue” datapoint which has a strange label – this maybe the result of errors in the database. If every other point that is close to this rogue point has a consistently different label, we wouldn't want a new test point to take the label of the rogue point. K nearest neighbours is a way to more robustly classify datapoints by looking at more than just the nearest neighbour.

2.2 K Nearest Neighbours

As the name suggests, the idea here is to include more than one neighbour in the decision about the class of a novel point x. I will here assume that we are using the Euclidean distance as the simmilarity measure – the generalisation to other dissimilarity measures is obvious. This is achieved by considering a hypersphere centred on the point x with radius r. We increase the radius r until the hypersphere contains exactly K points. The class label $c(x)$ is then given by the most numerous class within the hypersphere. This method is useful since classifications will be robust against “outliers” – datapoints which are somewhat anomalous compared

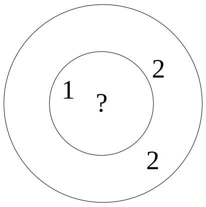


Figure 2.4: In K-nearest neighbours, we centre a hypersphere around the point we wish to classify. The first circle corresponds to the nearest neighbour method, and we would therefore class ? as class 1. However, using the 3 nearest neighbours, we find that there are two 2's and one 1 – and we would therefore class ? as a 2.

with other datapoints from the same class. The influence of such outliers would be outvoted.

How do we choose K?

Clearly if K becomes very large, then the classifications will become all the same – simply classify each x as the most numerous class. We can argue therefore that there is some sense in making $K > 1$, but certainly little sense in making $K = P$ (P is the number of training points). This suggests that there is some "optimal" intermediate setting of K. By optimal we mean that setting of K which gives the best generalisation performance. One way to do this is to leave aside some data that can be used to test the performance of a setting of K, such that the predicted class labels and the correct class labels can be compared. How we define this is the topic of a later chapter.

Generalisation

2.3 Handwritten digit Example

We will apply the nearest neighbour technique to classify handwritten digits. In our first experiement, we will first look at a scenario in which there are only two digit types, zeros, and ones. There are 300 training examples of zeros, and 300 training examples of ones, fig(2.5). We will then use the nearest neighbour method to predict the label of 600 test digits, where the 600 test digits are distinct from the training data and contain 300 zeros and 300 ones (although, of course, the test label is unknown until we assess the performance of our predictions). The nearest neighbour method, applied to this data, predicts correctly the class label of all 600 test points. The reason for the high success rate is that examples of zeros and ones are sufficiently different that they can be easily distinguished using such a simple distance measure.

In a second experiment, a more difficult task is to distinguish between ones and sevens. We repeated the above experiment, now using 300 training examples of ones, and 300 training examples of sevens, fig(2.6). Again, 600 new test examples (containing 300 ones and 300 sevens) were used to assess the performance. This time, 18 errors are found using nearest neighbour classification – a 3% error rate for this two class problem. The 18 test points on which the nearest neighbour method makes errors are plotted in fig(2.7). Certainly this is a more difficult task than distinguishing between zeros and ones. If we use $K = 3$ nearest neighbours, the

State of the art

classification error reduces to 14 – a slight improvement. Real world handwritten digit classification is big business. The best methods classify real world digits (over all 10 classes) to an error of less than 1% – better than human average performance.

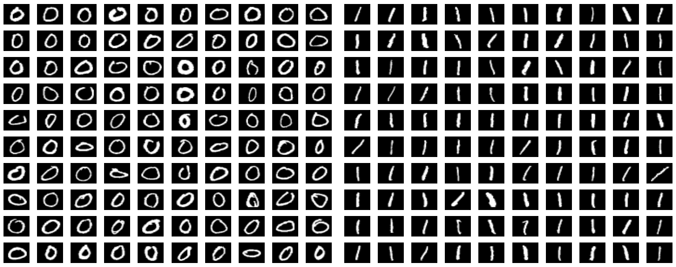


Figure 2.5: (left) Some of the 300 training examples of the digit zero and (right) some of the 300 training examples of the digit one.

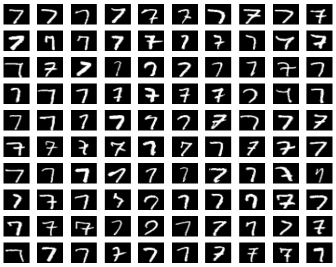


Figure 2.6: Some of the 300 training examples of the digit seven.

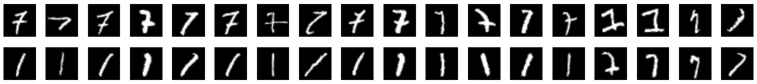


Figure 2.7: The Nearest Neighbour method makes 18 errors out of the 600 test examples. The 18 test examples that are incorrectly classified are plotted (above), along with their nearest neightbour in the training set (below).

2.4 A Probabilistic Interpretation of Nearest Neighbours

The Nearest Neighbour method is powerful, and its use widespread. In the context of probabilistic models, however, it does not at first sight appear to fit well. Here we'll show how NN methods indeed can be seen as limiting cases of probabilistic models. This is useful since this insight opens up the way to generalise upon the simple NN methods to more complex approaches in a natural way.

Consider the situation where we have (for simplicity) data from two classes – class 0 and class 1. We make the following model for data from class 0:

$$p(x|c=0) = \frac{1}{P_0}\frac{1}{(2\pi\sigma^2)^{N/2}}\sum_{\mu\in class0} e^{-(x-x^\mu)^2/(2\sigma^2)}$$

where N is the dimension of a datapoint x and P_0 are the number of training datapoints of class 0, and σ^2 is the variance. This is simple version of a so-called *Parzen* estimator, which simply models the data distribution as a sum of distributions centred on the training points.

Similarly, for data from class 1:

$$p(x|c=1) = \frac{1}{P_1} \frac{1}{(2\pi\sigma^2)^{N/2}} \sum_{\mu \in class1} e^{-(x-x^\mu)^2/(2\sigma^2)}$$

Then, in order to classify a new datapoint x^*, we need to calculate

$$p(c=0|x^*) = \frac{p(x^*|c=0)p(c=0)}{p(x^*|c=0)p(c=0) + p(x^*|c=1)p(c=1)}$$

which follows from using Bayes' rule. One can show (exercise) that the maximum likelihood setting of $p(c=0)$ is $P_0/(P_0+P_1)$, and $p(c=1) = P_1/(P_0+P_1)$. A similar expression holds for $p(c=1|x^*)$. Hence

$$\frac{p(c=0|x^*)}{p(c=1|x^*)} = \frac{p(x^*|c=0)p(c=0)}{p(x^*|c=1)p(c=1)} \tag{2.4.1}$$

If σ is very small, the numerator, which is a sum of exponential terms, will be dominated by that term for which x^{μ_0} in class 0 is closest to the point x^*. Similarly, the denominator will be dominated by that point x^{μ_1} in class 1 which is closest to x^*. Hence

$$\frac{p(c=0|x^*)}{p(c=1|x^*)} \approx \frac{e^{-(x^*-x^{\mu_0})^2/(2\sigma^2)}p(c=0)/P_0}{e^{-(x^*-x^{\mu_1})^2/(2\sigma^2)}p(c=1)/P_1} = \frac{e^{-(x^*-x^{\mu_0})^2/(2\sigma^2)}}{e^{-(x^*-x^{\mu_1})^2/(2\sigma^2)}}$$

Taking the limit $\sigma^2 \to 0$, we will with certainty therefore classify x^* as class 0 if x^* has a point in the class 0 data which is closer than the closest point in the class 1 data, and vice versa. This, of course, is simply the nearest neighbour method.

The motivation of using K nearest neighbours is produce a result that is more robust to outliers (mislabelled training data points). To ensure a similar kind of robustness in the probabilistic interpretation, we can simply use a larger value of σ^2. This will smooth the extreme probabilities of classification and mean that more points (not just the nearest) will have an effective contribution to the numerator and denominator of equation (2.4.1).

The extension to case of more than two classes is straightforward.

Using this interpretation we can see how NN can be seen as a limiting case of a *generative model* for data, from which Bayes rule is used to infer the class. Indeed, the extension to something akin to KNN is natural. To go further forward, it would therefore be natural to relax the assumption about using a Parzen estimator, and use something more complex. We will examine such cases in some detail in later chapters.

3 Linear Dimension Reduction

Often in machine learning, the data is very high dimensional. In the case of the hand-written digits from chapter(2), the data is 784 dimensional. Images are a good example of high dimensional data for which some of the basic motivations and assumptions about machine learning may come to light.

For simplicity, consider the case of the handwritten digits in which each pixel is binary – either 1 or 0. In this case, the total possible number of images that could ever exist is $2^{784} \approx 10^{236}$ – this is an extremely large number. However, it is clear that only perhaps at most a hundred or so examples of a digit 7 would be sufficient (for a human) to understand how to recognise a 7. Indeed, the world of digits must therefore lie in a highly constrained subspace of the 784 dimensions. It is certainly not true that each dimension is independent of the other in the case of digits. In other words, certain directions in the space will be more important than others for describing digits. This is exactly the hope, in general, for machine learning – that only a relatively small number of directions are relevant for describing the true process underlying the data generating mechanism. That is, any model of the data will have a relatively low number of effective degrees of freedom. These lower dimensional independent representations are often called 'feature' representations, since it is these quintessential features which succinctly describe the data.

A hook for machine learning

Features

Linear Dimension Reduction

In general, it seems clear that the way dimensions depend on each other is, for a general machine learning problem (and certainly the digits data) very complex – certain dimensions being 'on' means that others are likely to be 'off'. This suggests that non-linear effects will, in general, be important for the efficient description of data. However, finding non-linear representations of data is numerically difficult. Here, we concentrate on linear dimension reduction in which a high dimensional datapoint x is represented by $y = Fx$ where the non-square matrix F has dimensions $dim(y) \times dim(x)$, $dim(y) < dim(x)$. The matrix F represents a linear projection from the higher dimensional x space to the lower dimensional y space. The form of this matrix determines what kind of linear projection is performed and, classically, there are several popular choices. The two most popular correspond to Principal Components Analysis (PCA) and Linear Discriminants. The first is an unsupervised and the latter a supervised projection. We concentrate in this chapter on the more generic PCA, leaving linear discriminants to a later chapter. Note that, again, these methods do not describe any model from which we could generate data and, are also non-probabilistic. However, probabilistic data generating versions do exist which are model based but are beyond the scope of this course.

3.1 Principal Components Analysis

If data lies in a high dimensional space, we might hope that it lies close to a hyperplane, as in fig(20.1). We then can approximate each data point by using the vectors that span the hyperplane alone. I will sometimes refer to this small set of vectors as the "basis" set. Strictly speaking this is not a basis for the

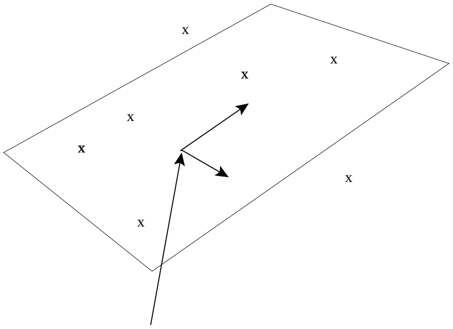


Figure 3.1: In linear dimension reduction we hope that data that lies in a high dimensional space lies close to a hyperplane that can be spanned by a smaller number of vectors.

whole space, rather is is a 'basis' which approximately spans the space where the data is concentrated. Effectively, we are trying to choose a more appropriate low dimensional co-ordinate system that will approximately represent the data. Mathematically, we write

$$x \approx c + \sum_{i=1}^{M} w_i b^i \tag{3.1.1}$$

The vectors $b^i, i \in 1, \dots M$ are chosen to be orthonormal. That is $(b^i)^\mathsf{T} b^j = 0$ for $i \neq j$, and $(b^i)^\mathsf{T} b^i = 1$. There is no loss of generality in this, since any non-orthonormal basis would simply correspond to a transformation of the coefficients w_i^μ.

If the dimension of the data space, $dim(x) = N$, our hope is that we can describe the data using only a small number M of vectors. If we can do so, we can reduce greatly the information needed to accurately describe the data. For example, if the data lies in a 784 dimensional space, we might hope that we can describe the data accurately using the above linear prescription with a much smaller dimensional representation.

One can show (see end of chapter) that the optimal lower dimensional representation (optimal in the sense of minimal squared reconstruction error) is given by projecting the data onto the eigenvectors of covariance matrix with the largest M eigenvalues. Algorithmically, this is :

1. Find the mean and covariance matrix of the data:

$$m = \frac{1}{P}\sum_{\mu=1}^{P} x^\mu, \qquad S = \frac{1}{P-1}\sum_{\mu=1}^{P}(x^\mu - m)(x^\mu - m)^\mathsf{T} \tag{3.1.2}$$

2. Find the eigenvectors $e^1, \dots, e^M$ of the covariance matrix S which have the largest eigenvalues. Form the matrix $E = [e^1, \dots, e^M]$ which has the largest eigenvectors as its columns.

3. The lower dimensional represention of each data point x^μ is given by $y^\mu = E^\mathsf{T}(x^\mu - m)$.

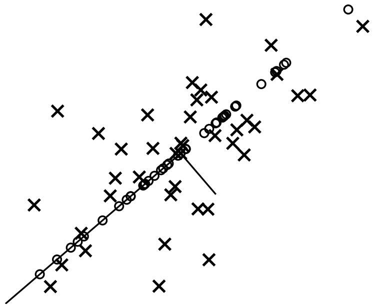


Figure 3.2: Projection of two dimensional data using one dimensional PCA. Plotted are the original datapoints (crosses) and their reconstructions using 1 dimensional PCA (circles). The two lines represent the eigenvectors and their lengths their corresponding eigenvalues.

4. The approximate reconstruction of the original datapoint x^μ is

$$x^\mu \approx m + Ey^\mu \tag{3.1.3}$$

5. The total squared error over all the training data made by the approximation is $(P-1)\sum_{j=M+1}^{N}\lambda_j$ where λ_j, $j = M+1\dots N$ are the eigenvalues discarded in the projection.

One can view the PCA reconstructions (though there is usually little use for these except to check that they give an adequate representation of the original data) as orthogonal projections of the data onto the subspace spanned by the M largest eigenvectors of the covariance matrix, see fig(3.2).

Interpreting the Eigenvectors

Do the eigenvectors themselves explicitly have any meaning? No! They only act together to define the linear subspace onto which we project the data – in themselves they have no meaning. We can see this since, in principle, any basis which spans the same subspace as the eigenvectors of the covariance matrix is equally valid as a representation of the data. For example, any rotation of the basis vectors within the subspace spanned by the first M eigenvectors would also have the same reconstruction error. The only case when the subspace is uniquely defined is when we only use one basis vector – that is, the principal component of the correlation matrix alone.

The "intrinsic" dimension of data

How many dimensions should the linear subspace have? As derived (at the end of the chapter), the reconstruction error is dominated by the largest eigenvalues of the covariance matrix. If we plot the eigenvalue spectrum (the set of eigenvalues ordered by decreasing value), we might hope to see a few large values and many small values. Indeed, if the data did lie very close to say a M dimensional linear manifold (hyperplane), we would expect to see M large eigenvalues, and the rest to be very small. This would give an indication of the number of degrees of freedom in the data, or the intrinsic dimensionality. The directions corresponding to the small eigenvalues are then interpreted as "noise".

Warning!

It might well be that a small reconstruction error can be made by using a small number of dimensions. However, it could be that precisely the information required to perform a classification task lies in the "noise" dimensions thrown away by the above procedure (though this will hopefully be rather rare). The purpose of linear discriminants is to try to deal with this problem.

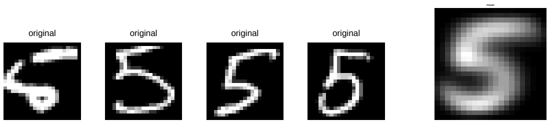


Figure 3.3: (left) Four of the 892 images. (right) The mean of the 892 images

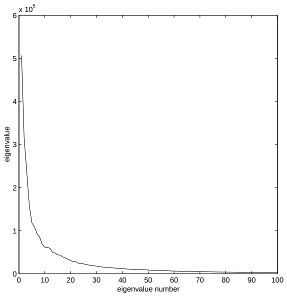


Figure 3.4: The 100 largest eigevalues

Non-linear Dimension Reduction

Whilst it is straightforward to perform the above linear dimension reduction, bear in mind that we are presupposing that the data lies close to a hyperplane. Is this really realistic? More generally, we would expect data to lie on low dimensional curved manifolds. Also, data is often clustered – examples of handwritten '4's look similar to each other and form a cluster, separate from the '8's cluster. Nevertheless, since linear dimension reduction is so straightforward, this is one of the most powerful and ubiquitous techniques used in dimensionality reduction.

3.1.1 Example : Reducing the dimension of digits

We have 892 examples of handwritten 5's. Each is a 21*23 pixel image – that is, each data point is a 483 dimensional vector. We plot 4 of these images in fig(3.3). The mean of the data is also plotted and is, in a sense, an archetypal 5. The covariance matrix has eigenvalue spectrum as plotted in fig(3.4), where we plot only the 100 largest eigenvalues. The reconstructions using different numbers of eigenvectors (10, 50 and 100) are plotted in fig(3.5). Note how using only a small number of eigenvectors, the reconstruction more closely resembles the mean image.

3.1.2 PCA and Nearest Neighbours

In the chapter on nearest neighbour methods, we needed to calculate (many times) distances between vectors. This can be computationally demanding, and it is often a good idea (when using the euclidean distance) to project the data onto a lower dimension first. For example, in the case where we wanted to make a classifier to distinguish between the digit 1 and the digit 7 we first use PCA by ignoring the classlabel (to make a dataset of 1200 training points). Each of the training points

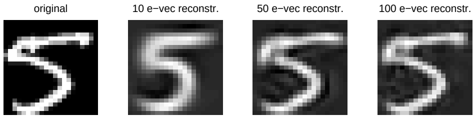


Figure 3.5: The reconstruction using different linear subspace dimensions

x^μ is then projected onto its 50 dimensional PCA representation y^μ. Subsequently, any distance calculations $(x^a - x^b)^2$ are replaced by $(y^a - y^b)^2$.

To see that this is reasonable, consider

$$\begin{aligned}(x^a - x^b)^{\mathsf{T}}(x^a - x^b) &\approx (Ey^a - m - Ey^b + m)^{\mathsf{T}}(Ey^a - m - Ey^b + m) \\ &= (y^a - y^b)^{\mathsf{T}} E^{\mathsf{T}} E (y^a - y^b) \\ &= (y^a - y^b)^{\mathsf{T}}(y^a - y^b) \qquad (3.1.4)\end{aligned}$$

where the last equality is due to the orthogonality of eigenvectors : $E^{\mathsf{T}} E = I$.

Using 50 principal components, the error rate using the nearest neighbour rule to classify ones and sevens gave an error of 13 in 600 examples – better than without using PCA! How can this be? A possible explanation for this type of phenomenon is that the new PCA representation of the data is more relevant (effectively, common, irrelevant directions in the data are ignored), so that distances between these more relevant representations of the data can produce better results. This is not always to be expected, however.

3.1.3 Mega Dimensional Data

You might be wondering how it is possible to perform PCA on extremely high dimensional data. For example, if we have 500 images each of $1000 \times 1000 = 10^6$ pixels, the covariance matrix will be $10^6 \times 10^6$ dimensional – well beyond the storage capacities of many computers.

One approach around this difficulty is to perform the calculations in a lower dimensional space. Note that there can only be at most P non-zero eigenvalues.

Using X to denote the (zero mean) data and E the matrix of eigenvectors – this is non-square since there will be fewer eigenvalues than dimensions. We write the eigenvalues as a diagonal matrix Λ. The eigenvalue requirement is

$$XX^{\mathsf{T}}E = E\Lambda \qquad (3.1.5)$$

$$X^{\mathsf{T}}XX^{\mathsf{T}}E = X^{\mathsf{T}}E\Lambda \qquad (3.1.6)$$

$$X^{\mathsf{T}}X\tilde{E} = \tilde{E}\Lambda \qquad (3.1.7)$$

where we defined $\tilde{E} = X^{\mathsf{T}}E$. The last line above represents the eigenvector equation for $X^{\mathsf{T}}X$. This is a matrix of dimensions $P \times P$ – in the above example, a 500×500 matrix as opposed to a $10^6 \times 10^6$ matrix previously. We then can calculate the eigenvectors $\tilde{E}$ and eigenvalues Λ of this matrix more easily. Once found, we then use

$$E = X\tilde{E}\Lambda^{-1} \qquad (3.1.8)$$

3.1.4 PCA is good because it is a poor compressor!

A moments thought throws up the following condundrum: It seems that we wish to compress high dimensional data to a lower dimensional representation. However, clearly, the optimal compressed representation retains no structure since, if it did, further compression would still be possible. The goal of feature extraction is not consistent with optimal compression, since we wish to remove some redundancy, yet retain enough structure in the lower dimensional representation such that any further use of the data – making a machine which can generalise from the lower dimensional representations for example – has a chance. Hence, perhaps somewhat perversely, PCA is a reasonable feature extraction method because it is such a poor compressor!

3.2 Deriving the Optimal Linear Reconstruction

To find the best basis b and corresponding expansion coefficients w, we need to minimize the error term

$$E = \sum_{\mu,i} \left(x_i^\mu - \sum_j w_j^\mu b_i^j \right)^2 \tag{3.2.1}$$

In matrix notation, we could write this similarly as

$$E = \text{trace}\left((X - WB) (X - WB)^\mathsf{T} \right)$$

for suitably defined X, W and B. Consider a transformation of the basis B so that $\tilde{B} \equiv QB$ is an orthogonal matrix. Since Q is invertible, we may absorb this into W so that WB may be written as $\tilde{W}\tilde{B}$, which is of then same form as WB, albeit with an orthogonality constraint on $\tilde{B}$. Hence, without loss of generality, we may consider equation (3.2.1) with the b vectors assumed orthonormal. By differentiating equation (3.2.1) with respect to w_k^μ we obtain (using the orthonormality constraint)

$$w_k^\mu = \sum_i b_i^k x_i^\mu$$

Defining $[U]_{i,j} = b_i^j$, the term

$$\sum_j w_j^\mu b_i^j = \sum_{k,j} b_i^j b_k^j x_k^\mu = \sum_{j,k} U_{i,j} U_{k,j} x_k^\mu = [UU^\mathsf{T} x^\mu]_i$$

and the orthogonality constraint is $U^\mathsf{T} U = I$. The objective E (neglecting the orthonormality constraint) becomes

$$E = \sum_\mu \left((I - UU^\mathsf{T}) x^\mu \right)^2$$

For simplicity, define the symmetric matrix

$$\tilde{B} = I - UU^\mathsf{T}$$

One can show easily that $\tilde{B}\tilde{B} = \tilde{B}$. Then

$$E = \sum_\mu \text{trace}\left((x^\mu)^\mathsf{T} \tilde{B} x^\mu \right)$$

using the permutability of trace, this is

$$E = \text{trace}\left(\sum_\mu (x^\mu) (x^\mu)^\mathsf{T} \tilde{B} \right)$$

Hence the objective becomes (neglecting constants)

$$E = -\mathrm{trace}\left(SUU^{\mathsf{T}}\right)$$

where S is the correlation matrix of the data. The constraint can be written (using a set of Lagrange multipliers)

$$-\mathrm{trace}\left(SUU^{\mathsf{T}}\right) + \mathrm{trace}\left(M\left(U^{\mathsf{T}}U - I\right)\right)$$

Since the constraint is symmetric, we can assume that M is also symmetric. Differentiating with respect to U, we get

$$SU = UM$$

Clearly, one solution is given by taking M to be diagonal and then U to be the matrix whose columns are the corresponding eigenvectors of S. In this case, $\mathrm{trace}\left(SU^{\mathsf{T}}U\right) = \mathrm{trace}\left(M\right)$, which is the sum of the eigenvalues corresponding to the eigenvectors forming U. Since we wish to minimise E, we should then take the largest eigenvalues to minimize E.

Remember that, whilst the solution to this eigen-problem is unique, this only serves to define the solution subspace since, from the previous argument, we may transform the basis as we wish. This means that the eigenvectors should only be interpreted through defining the space, and any other vector set which defines the same subspace would be equally appropriate.

3.3 Problems

Exercise 12 *Consider $AA^{-1} = I$. By applying a differential operator ∂, show that*

$$\partial A^{-1} = A^{-1}\partial A A^{-1}$$

By using the identity

$$e^{\log A} = A$$

show that

$$\partial \log(A) = A^{-1}\partial A$$

Show that

$$\log\det A = \mathrm{trace}\left(\log A\right)$$

Exercise 13 *Consider a dataset in two dimensions where the data lies on the circumference of a circle of unit radius. What would be the effect of using PCA on this dataset, in which we attempt to reduce the dimensionality to 1? Suggest an alternative one dimensional representation of the data.*

Exercise 14 *Consider two vectors x^a and x^b and their corresponding PCA approximations $c+\sum_{i=1}^{M} a_i e^i$ and $c+\sum_{i=1}^{M} b_i e^i$, where the eigenvectors $e^i, i = 1, \dots M$ are mutually orthogonal and have unit length. The eigenvector e^i has corresponding eigenvalue λ^i.*

Approximate $(x^a - x^b)^2$ by using the PCA representations of the data, and show that this is equal to $(a - b)^2$.

Exercise 15 *The Gaussian distribution in one dimension is defined as*

$$p(x|c) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$$

We decide to fit a Gaussian to each class from a dataset of one-dimensional data. Show that the Maximum Likelihood estimator of μ_1 *is* $\hat{\mu}_1 = \frac{1}{n_1}\sum_{x \in class1}^{n_1} x$ *and that the ML estimate of* σ_1^2 *is* $\hat{\sigma}_1^2 = \frac{1}{n_1}\sum_{x \in class1}^{n_1}(x - \hat{\mu}_1)^2$

Exercise 16 *Let* S *be the covariance matrix of the data. The Mahalanobis distance between* x^a *and* x^b *is defined as*

$$\left(x^a - x^b\right)^{\mathsf{T}} S^{-1} \left(x^a - x^b\right).$$

Explain how to approximate this distance using the M*-dimensional PCA approximations, as described above.*

Exercise 17 *You are hired as the researcher in a startup company specializing in image processing. You are hired to provide the theory, algorithms and eventually to build a hand-written digit classifier to be used for Postcode recognition by the Post Office, giving a value from* 0 *to* 9 *for the class of a novel testpoint. The training data is of the form* $x^\mu, \mu = 1, \ldots, P$*, where each image* x^μ *has a corresponding class label,* $c^\mu \in \{0, 1, \ldots, 9\}$*.*

Your line manager is very worried. She argues that for the case of binary valued images, with $100 \times 100 = 10000$ *pixels, there are in total* 2^{10000} *such binary images. She argues that this number is astronomically larger than the number of atoms in the universe and that therefore the measly* 25000 *training examples they have will be woefully inadequate for training. Explain how you might persuade your line manager that the situation may not be so hopeless, giving compelling reasons for your optimism.*

Your line manager is so impressed with your argument that she decides to discard all the training data currently in the database, and busies herself with making 1000 training examples of her own fair hand. Is this a reasonable thing to do? Justify your answer.

As a first step, you decide to use the K *nearest neighbour method (KNN) to classify a novel test point* x^**.*

Describe fully how the KNN algorithm works, including how to determine the optimal number of neighbours K *to use.*

Your line manager is pleased with your algorithm but is concerned that it performs as well as you say it does. How can you persuade her that it really performs according to your claims?

One morning your line manager is delighted to tell you that she now has more training data than ever before, indeed the training data consists of $P = 100,000$ *real valued images. You estimate that your current KNN method is going to be too slow to do real time classification of digits and you decide to use PCA to increase classification speed.*

Describe fully and mathematically how to use PCA to replace an N *dimensional vector* x *with an* M *dimensional vector* y *where* $M < N$*.*

Derive a formula for approximating the distance $(x^a - x^b)^2$ between two vectors x^a and x^b using their corresponding PCA representations y^a and y^b.

Your line manager is pleased with your faster algorithm, which you claim provides 95% classification accuracy. She tells you that it is important to make sure that 99% are classified correctly in the end, even if this means that 10% of test images need to be classified by a human. She asks you to adapt your algorithm accordingly. Suggest an amendment to your algorithm and explain how you would decide whether or not to leave a novel test point to be classified by a human.

Exercise 18 *In a recent radio lecture, the following phrase was uttered by a famous Professor of Experimental Psychology:*

"In a recent data survey, 90% of people claim to have above average intelligence, which is clearly nonsense!" [Audience Laughs]. Discuss the truth or falsity of this statement, and justify your answer.

Exercise 19 (PCA with external inputs) *In some applications, one may suspect that certain variables have a strong influence on how the data x is distributed. For example, we could have a set of variables v_k^μ,$k = 1, \ldots K$ for each observation x^μ, $\mu = 1, \ldots P$, which we think will heavily influence each x^μ. It may therefore be sensible to include these variables, and assume an approximation*

$$x^\mu \approx \sum_j w_j^\mu b^j + \sum_k v_k^\mu c^k \tag{3.3.1}$$

where the coefficients w_i^μ, $i = 1, \ldots N$, $\mu = 1, \ldots P$ and basis vectors b^j, $j = 1, \ldots J$ and c^k, $k = 1, \ldots K$ are to be determined.

The sum squared error loss between the x^μ and their linear reconstruction equation (3.3.1) is

$$E = \sum_{\mu,i} \left(x_i^\mu - \sum_j w_j^\mu b_i^j - \sum_k v_k^\mu c_i^k \right)^2 \tag{3.3.2}$$

Find the parameters that minimise E.

Exercise 20 *Consider the following 3-dimensional datapoints:*

$(1.3, 1.6, 2.8)(4.3, -1.4, 5.8)(-0.6, 3.7, 0.7)(-0.4, 3.2, 5.8)(3.3, -0.4, 4.3)(-0.4, 3.1, 0.9)$

. Perform Principal Components Analysis by:

- *Calculating the mean, c of the data.*
- *Calculating the covariance matrix $\boldsymbol{S = \frac{1}{6} \sum_{\mu=1,\ldots,6} x^\mu (x^\mu)^\mathsf{T} - c c^\mathsf{T}}$ of the data.*
- *Finding the eigenvalues and eigenvectors e^i of the covariance matrix. (This is easy to do in MATLAB using the `eig` command.)*

You should find that only two eigenvalues are large, and therefore that the data can be well represented using two components only. Let e^1 and e^2 be the two eigenvectors with largest eigenvalues.

- *Calculate the two dimensional representation of each datapoint* $(\boldsymbol{e^1} \cdot (\boldsymbol{x^\mu} - \boldsymbol{c}), \boldsymbol{e^2} \cdot (\boldsymbol{x^\mu} - \boldsymbol{c})), \boldsymbol{\mu = 1, \ldots, 6}$.
- *Calculate the reconstruction of each datapoint* $\boldsymbol{c} + (\boldsymbol{e^1} \cdot (\boldsymbol{x^\mu} - \boldsymbol{c}))\boldsymbol{e^1} + (\boldsymbol{e^2} \cdot (\boldsymbol{x^\mu} - \boldsymbol{c}))\boldsymbol{e^2}$, $\mu = 1, \ldots, 6$.

Exercise 21 *Consider a set of* N*-dimensional data* $x^\mu, \mu = 1, \ldots, P$. *A PCA analysis is performed on this dataset, and the eigenvectors* $e^1, \ldots, e^M$ *are used to represent the data, together with the mean* c. *That is, each original* N *dimensional datapoint* x^μ *is represented in the form*

$$x^\mu \approx c + \sum_{i=1}^{M} a_i e^i \tag{3.3.3}$$

- *What are the optimal coefficients* a_i *if we are to minimise the square length of the residual vector?*
- *Consider two vectors* x^a *and* x^b *and their corresponding PCA approximations* $c + \sum_{i=1}^M a_i e^i$ *and* $c + \sum_{i=1}^M b_i e^i$. *Approximate* $(x^a - x^b)^2$ *by using the PCA representations of the data, and show that this is equal to* $(a - b)^2$.
- *Explain how the above result enables us to rapidly compare distances between PCA representations of datapoints, and how therefore in the Eigenfaces experiment, we can perform nearest neighbour classification rapidly, once the 20 largest eigenvectors have been calculated.*

3.4 Solutions

16 Using the approximations, we have

$$(x^a - x^b)^{\mathsf{T}} S^{-1} (x^a - x^b) \approx (\sum_i a_i e^i - \sum_i b_i e^i)^{\mathsf{T}} S^{-1} (\sum_j a_j e^j - \sum_i b_j e^j)$$

Due to the orthonormality of the eigenvectors, this is $\sum_i a_i^2/\lambda_i - 2a_i b_i/\lambda_i + b_i^2/\lambda_i = (a - b)^{\mathsf{T}} D^{-1} (a - b)$ where D is a diagonal matrix containing the eigenvalues.

17 Even though 25000 is a very small number compared to 2^{10000}, the point is that digits are not simply random point in a 10000 dimensional space. There is a great deal of regularity and constraint on the form that each digit can take, so that digits will occupy only a very small fraction of the space of all possible images. Indeed, humans are capable of learning digits based on only a small number of training examples, and there is therefore every reason to be optimistic that a machine could do the same.

If we wish to make a classifier that works well on a wide variety of peoples handwriting, we need training data that is representative of a wide variety of styles. Otherwise, the trained classifier may be appropriate for recognizing the handwriting of the line manager, but not necessarily anyone else.

The classification of the KNN method is based on finding the K nearest neighbours. If none of the neighbours is very close, this will result is potentially inaccurate classification. A simple method is therefore to use an independent testset, and set a threshold value. Measure the distance to the nearest neighbour for each testpoint to be classified, and discard this point if it is greater than the threshold. For the remaining undiscarded points, determine the classification. If this is not 99%, increase the threshold and repeat the procedure until a just sufficient value of the threshold has been found.

18 Clearly false. A canny student will be able to give an example to demonstrate this, which is surely the result of a highly non-symmetric distribution with many (slightly) above average values and a few extremely low values. A simple demonstration is to assume that the average IQ is 100, and the minimum 0. If there are only two possible scores, the above average score, and the below average score, then one can easily show that 90 percent of people can indeed have an above average IQ if the above average IQ score is less than 111.111.

19 To optimise equation (3.3.2), it is straightforward to show that we should first transform the data to be zero mean : $\sum_\mu x^\mu = 0$ and $\sum_\mu v_k^\mu = 0$, $k = 1, \ldots, K$. We may assume, without loss of generality, that the b^j are orthonormal (since we could rescale the w_j^μ if not). However, we cannot assume that the c^k, $k = 1, \ldots, K$ are orthonormal, since we cannot rescale the v. Similarly, we assume nothing, a priori, regarding the relationship between the vectors b^j and c^k. Differentiating equation (3.3.2) with respect to w^μ gives (using the orthonormality constraint on the b^i)

$$w^\mu = \sum_i (b^i)^\mathsf{T} \left(x^\mu - \sum_l v_l^\mu c^l \right)$$

The residual vector (difference between x^μ and the linear reconstruction) is then

$$r^\mu = x^\mu - \sum_i (b^i)^\mathsf{T} \left(x^\mu - \sum_l v_l^\mu c^l \right) b^i - \sum_j v_j^\mu c^j$$

By defining $\tilde{B} \equiv I - \sum_i b^i (b^i)^\mathsf{T} \equiv I - UU^\mathsf{T}$, (using the notation of the previous section), the residual is

$$r^\mu = \tilde{B} \left(x^\mu - \sum_j v_j^\mu c^j \right)$$

Differentiating $E = \sum_\mu (r^\mu)^\mathsf{T} r^\mu$ with respect to c^i, we get

$$\sum_\mu v_i^\mu \tilde{B} x^\mu = \sum_j \sum_\mu v_j^\mu v_i^\mu \tilde{B} c^j$$

Define

$$[\tilde{V}]_{ij} = \sum_\mu v_i^\mu v_j^\mu, \qquad [\tilde{X}]_{ij} = \sum_\mu v_i^\mu x_j^\mu, \qquad C = [c^1, \ldots, c^K]$$

Then the above has solution

$$C = \tilde{X} \tilde{V}^{-1}$$

Hence, the objective involves the matrix

$$\tilde{S} = \sum_\mu (x^\mu - d^\mu)(x^\mu - d^\mu)^\mathsf{T}$$

where

$$d^{\mu} = \sum_{j} v_{j}^{\mu} c^{j}$$

Hence, the optimal solution is given by taking the principal e-vecs of $\tilde{S}$, with C set as above.

I believe this is a special case of Constrained Principal Components Analysis (CPCA)[1].

4 Generalisation

In the previous chapters, we have discussed ways to practically implement a classifier. In this chapter, we deal with some of the underlying conceptual issues as to how to gain confidence that the method will generalise way to new examples not seen in the training set.

4.1 Introduction

The central goal in supervised learning is, on the basis of labelled training data, to encapsulate the underlying mechanism which generated the data, thereby learning a model with *predictive power* . That is, given a novel unlabelled instance, to make an accurate prediction.

4.1.1 Supervised Learning

Formally, supervised learning consists of the following scenario: A given set of training data, $D_{train} = \{(x^\mu, t^\mu), \mu = 1, \ldots, P\}$ where each x^μ is a vector of (in general real valued) attributes, and t^μ is the associated *target* for the μ^{th} pattern. (In the binary classification tasks we have been considering, $t^\mu \in \{0, 1\}$). If t^μ can take on one of only a finite set of values, we call this a *classification* problem. If t^μ can be any value, we call this a *regression* problem.

Our basic paradigm is the following : There is some underlying process which generates "clean" data. The data is then possibly corrupted by noise to give the actual data that we observe. The aim in supervised learning is to try to recover the underlying clean data generating mechanism.

Prediction without assumptions is meaningless

Consider the following problem. What is the next number in the sequence 1,2,3,4,5, ?[1] Well, there is no "correct" answer. I could predict anything, and who's to say that I'm incorrect? It may well be perfectly reasonable to a Martian to say the next number is 78. In fact, the "answer" that I was looking for was 63. This is the number of the bus that follows busses 1,2,3,4 and 5 in my home town. "Not fair!", you might say, "we didn't know that you were talking about busses in your home town". Well, that's what learning from data is about – you have to try to collate as much information about the data domain as you can, and hope that your assumptions are reasonable. Whatever you do, your predictions are only as good as your assumptions.

Consider the training data in fig(4.1). This is an example of a regression problem in which the targets t are real values. In this case, the inputs x are also real values. Our aim is to fit a function $f(x|\boldsymbol{\theta})$ to this data. What kinds of functions might we fit? For example, a straight line fit $f(x|\boldsymbol{\theta}) = \theta_0 + \theta_1 x$ may look reasonable.

Or is it a 10^{th} order polynomial, $f(x|\boldsymbol{\theta}) = \sum_{i=0}^{10} x^i \theta_i$, see fig(4.2)? In fact, the data was generated using the rule $t = \sin(2.5x) + \eta$, where η is zero mean Gaussian noise of variance 0.2^2.

[1] This is a supervised problem since a sequence has a temporal ordering and can be written as (x, t) pairs : $(1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, ?)$.

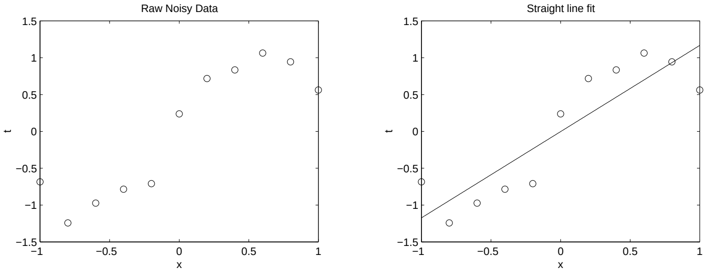


Figure 4.1: Left: Training Data for a regression problem. We wish to fit a function $f(x|\boldsymbol{\theta})$ to this data. Right: A straight line fit might look reasonable.

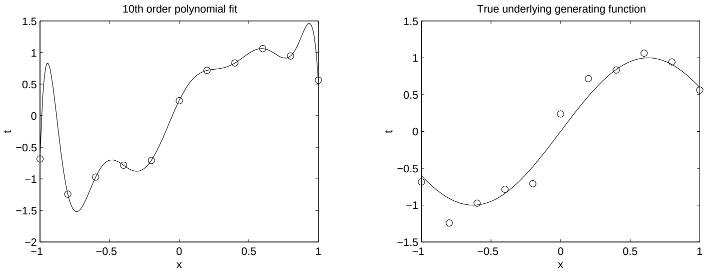


Figure 4.2: Left: What about a 10^th order polynomial. This has zero training error. Right: The “correct” clean underlying function which generated the data.

To find a “good” curve, we need appropriate beliefs/assumptions about the smoothness of the “clean” underlying function *and* the level of noise. If our assumptions are not correct, our predictions will be poor. n

Classes of Predictor

Our general approach to supervised learning is to make some function $f(x|\boldsymbol{\theta})$ so that, given a novel point x our prediction $f(x|\boldsymbol{\theta})$ will be accurate. What do we mean by accurate? If we had some extra data, D_{test}, different from the training data and generated in the same manner, then we would like that the error made by our predictions is roughly the same as the error that would be made even if we knew exactly what the clean underlying data generating process were. Of course, this is in some sense, an impossible task. However, we can devise procedures that can help give us some confidence in our predictions.

4.1.2 Training Error

The typical way that we train/learn the adjustable parameters $\boldsymbol{\theta}$ of our model is to optimise some objective function. For example, if our current model outputs $f(x^\mu|\boldsymbol{\theta})$ on an input x^μ and the training data output for that x^μ is t^μ, we would like to adjust the parameters $\boldsymbol{\theta}$ such that $f(x^\mu|\boldsymbol{\theta})$ and t^μ are close. We can measure how close these values are by using a function $d(x,y)$ which measures the discrepancy between two outputs x and y. To find the best parameter settings for the whole training set, we use

$$E_{train}(\boldsymbol{\theta}) = \sum_{(x^\mu,t^\mu)\in D_{train}} d(f(x^\mu|\boldsymbol{\theta}), t^\mu) \tag{4.1.1}$$

If we adjust the parameters $\boldsymbol{\theta}$ to minimise the training error, what does this tell us about the prediction performance on a novel point x? In principle, nothing! However, in practice, since the mechanisms which generate data are in some sense smooth, we hope that our predictions will be accurate.

4.1.3 Test Error

Imagine that we have gone through a procedure to minimise training error. How can we assess if this will have a good predictive performance – i.e., will *generalise* well? If we have an independent set of data D_{test}, the *test error*

$$E_{test}(\boldsymbol{\theta}) = \sum_{(x^\mu,t^\mu)\in D_{test}} d(f(x^\mu|\boldsymbol{\theta}), t^\mu) \qquad (4.1.2)$$

is an unbiased estimate of the prediction error of our model $f(x|\boldsymbol{\theta})$.

4.1.4 Validation Data

Consider two competing prediction model classes, $f_1(x|\boldsymbol{\theta}_1)$ and $f_2(x|\boldsymbol{\theta}_2)$. We train each of these by minimising the training error to end up with training error "optimal" parameter settings $\boldsymbol{\theta}_1^*$ and $\boldsymbol{\theta}_2^*$. Which is the better model? Is it the one with the lower training error? No. We can say that model with setting $\boldsymbol{\theta}_1^*$ is better than a model with setting $\boldsymbol{\theta}_2^*$ by comparing the *test* errors, $E_{test}(\boldsymbol{\theta}_1) < E_{test}(\boldsymbol{\theta}_2)$. Using test data in this way enables us to validate which is the better model.

The standard procedure is to split any training data into three sets. The first is the training data, D_{train}, used to train any model. The second $D_{validate}$ is used to assess which model has a better test performance. Once we have chosen our optimal model on the basis of using validation data, we can get an unbiased estimate of the expected performance of this model by using a third set of independent data D_{test}. This data should not have been used in any way during the training procedure if we wish to obtain an unbiased estimate of the expected test performance of the model.

4.1.5 Dodgy Joe and Lucky Jim

Perhaps the following parody will make the above arguments clearer:

Let me introduce two characters, "Lucky Jim" and "Dodgy Joe". Lucky Jim invents some new procedure, and initially, finds that it works quite well. With further experimentation, he finds that it doesn't always work, and that perhaps it requires some rather fine tuning to each problem. Undeterred, this charismatic scientist attracts both funds and attention enough to stimulate a world wide examination of his method. Working independently of each other, surely enough research groups from around the world begin to report that they manage to achieve zero test error on each problem encountered. Eventually, some research group reports that they have found a procedure, based on Lucky Jim's method that is able to give *zero* test error on *every* problem that has ever been known to exist. After so many years of hard work, Lucky Jim happily announces his universal predictor (perhaps a billion hidden unit neural network with fixed parameters), with the (indeed true) claim that it gives zero test error on every known problem that ever existed. He markets this product and hopes to claim a fortune.

Contrast the dogged determination of Lucky Jim now with the downright unscrupulous behaviour of Dodgy Joe. Quite frankly, he doesn't have the patience of Lucky Jim, and he simply assembles all the known problems that ever existed, and their corresponding test sets. He then constructs his method such that, when asked to perform the prediction on problem A with corresponding test set B, he simply makes the output of his method the output for the test set B (which he of course knows). That is, his algorithm is nothing more than a lookup table - if the user says, “this is the test set B” then Dodgy Joe’s algorithm simply reads off the predictions for test set B which, by definition, will give zero error. He then also markets his universal predictor package as giving zero test performance on every known problem (which is indeed true) and also hopes to make a fortune.

If we look at this objectively, both Lucky Jim and Dodgy Joe’s programs are doing the same thing, even though they arrived at the actual code for each method in a different way. They are both nothing more than lookup tables. The point is that we have *no* confidence whatsoever that either Lucky Jim’s or Dodgy Joe’s package will help us in our predictions for a *novel* problem. We can only have confidence that a method is suitable for our novel problem if we believe that a particular method was successful on a *similar* problem to ours in the past, or the assumptions that resulted in successful prediction on a previous problem might well be expected to hold for a novel problem – smoothness of the problems for example.

The above also highlights the issue that it is not enough to assess a method only on the reported results of a subset of *independent* research groups. It may be that, with the same method (eg neural nets with a fixed architecture but undetermined parameters) one of a hundred groups which decide to tackle a particular problem is able to find that particular set of parameter values (essentially by chance) that gives good test performance, whilst the other 99 groups naturally do not report their poor results. In principal, real comparison of a method on a problem requires the collation of *all* results from *all* sources (attempts).

Model Comparison : An example

Let us reconsider our favourite digit classification problem. There are 1200 examples of the digit 1 and 7. Let us split this to form a new training set of 400 examples, and a validation set of 200 examples. We will retain a further 600 examples to measure the test error. I used PCA to reduce the dimensionality of the inputs, and then nearest neighbours to perform the classification on the 200 validation examples. Based on the validation results, I selected 19 as the number of PCA components retained, see fig(4.3). The independent test error on 600 independent examples using 19 dimensions is 14. Once we have used the validation data to select the best model, can we use both training and validation data to retrain the optimal model? In this case, we would have decided that 19 is the optimal dimension to use, based on 200 training and 100 validation points. Can we now, having decided to use 19 components, retrain the model on the 300 training and validation points? This is a fairly subtle point. In principle, the answer is no, since the new procedure, strictly speaking, corresponds to a different model. However, in practise, there is probably little to be lost by doing so, and possibly something to be gained since we are making use of more training data in setting many parameters. These issues highlight some of the philosophical complications that can arise based on a frequentist interpretation of probability. No such difficulties arise in the Bayesian framework, where all the training data can be used in a clear and consistent manner for model selection.

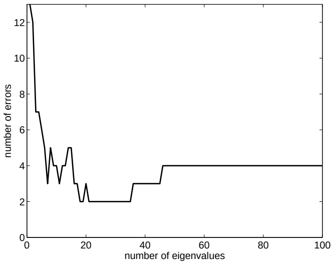


Figure 4.3: 400 training examples are used, and the validation error plotted on 200 further examples. Based on the validation error, we see that a dimension of 19 is reasonable.

4.1.6 Regularisation

If the data generation process includes noise (additive), then the true, clean data generating process will typically be smoother than the observed data would directly suggest. To try to discover this smoother clean underlying process, we need to ensure that our model for the clean underlying process does not fit the noise in the observed data. That is, it is undesirable to have a zero training error, and we need to encourage our model to be smooth. One way to achieve this is through regularisation in which an extra "penalty" term is added on the the standard training error, to form the regularised training error:

$$E_{regtrain}(\boldsymbol{\theta}, \lambda) = E_{train}(\boldsymbol{\theta}) + \lambda E_{reg}(\boldsymbol{\theta}) \tag{4.1.3}$$

The larger λ is, the smoother will be solution which minimises the regularised training error. If we regularise too much, the solution will be inappropriate and too smooth. If we don't regularise at all, the solution will be over complex, and the solution will be fitting the noise in the data. (Regularisation only really makes sense in the case of models which are complex enough that overfitting is a potential problem. There is little point in taming a pussy-cat; taming a lion however, might be worthwhile!). How do we find the "optimal" value for λ? Training is then done in two stages:

- For a fixed value of λ, find $\boldsymbol{\theta}^*$ that optimises $E_{regtrain}$. Repeat this for each value of λ that you wish to consider. This gives rise to a set of models, $\left\{\boldsymbol{\theta}^*_{\lambda_i}, i = 1, \ldots, V\right\}$.
- For each of these models, on a separate validation set of data (different from the training data used in the first step), calculate the validation error:

$$E_{val}(\boldsymbol{\theta}^*) = \sum_{(x^\mu, t^\mu) \in D_{val}} d(f(x^\mu|\boldsymbol{\theta}^*), t^\mu) \tag{4.1.4}$$

The "optimal" model is that which has the lowest validation error.

Regularisation : An example

In fig(4.4), we fit the function $t = a\sin(wx)$ to data, learning the parameters a and w. The unregularised solution badly overfits the data, and has a high validation error. To encourage a smoother solution, I used a regularisation term $E_{reg} = w^2$. I then computed the validation error based on several different values of the regularisation parameter λ, finding that $\lambda = 0.5$ gave a low validation error.

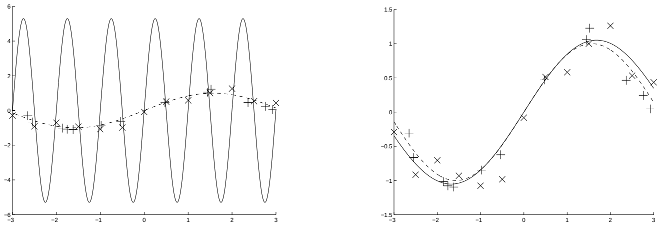


Figure 4.4: Left: The unregularised fit ($\lambda = 0$) to training given by ×. Whilst the training data is well fitted, the error on the validation examples, + is high. Right: the regularised fit ($\lambda = 0.5$). Whilst the training error is high, the validation error (which is all important) is low. The true function which generated this noisy data is the dashed line, and the function learned from the data is the solid line.

4.2 Problems

Exercise 22 *WowCo.com is a new startup prediction company. After years of failures, they eventually find a neural network with a trillion hidden units that achieves zero test error on every learning problem posted on the internet up till January 2002. Each learning problem included a training and test set. Proud of their achievement, they market their product aggressively with the claim that it 'predicts perfectly on all known problems'. Would you buy this product? Justify your answer.*

4.3 Solutions

5 Linear Discriminant Analysis

We will be interested here in how we can exploit class label information to improve the projection. That is to make *supervised* projections of the data. We begin with a discussion of a simpler method in which the projection occurs in an unsupervised way without using class information.

5.1 Unsupervised Dimension Reduction

Dimension reduction has been shown to be useful, not just in terms of reducing the number of parameters that we ultimately might use in our model, but can indeed be positively beneficial in removing irrelevant dimensions in the input space. PCA was one of our main methods for dimension reduction and, more generally, we could use autoencoders for non-linear dimension reduction. These methods are all *unsupervised* in the sense that only the inputs $x^\mu, \mu = 1, \ldots, P$ are used to determine the reduction. The targets $y^\mu, \mu = 1, \ldots, P$ are only used later, after the dimension reduction technique. One potentially useful aspect of dimension reduction is to be able to "see" the data. If we reduce the dimension to only two or three, we can *visualise* the data by plotting the corresponding points $y^\mu, \mu = 1, \ldots, P$ in the two or three dimensional space. This is potentially very useful since the human eye is very good at interpreting spacial relationships in data, and may give us some intuition about an appropriate model for the data.

5.1.1 Using PCA for visualisation

By finding say the largest three principal components of a dataset (independent of any class labels they may have), we can reduce each high dimensional vector x^μ to its low three dimensional representation y^μ. Then, we can plot each point y^μ and colour it according to its class label. Code to do this for our well known threes and fives digit data is given below.

The result of this procedure is plotted in fig(5.1). We can see that the classes are not particularly well separated spatially by this projection. Indeed, we did not make any use of the class labels in determining the projection, so we cannot necessarily expect that the projected data should appear well separated. Increasing the complexity to use a non-linear projection will not necessarily improve the situation either.

5.2 Fishers Linear Discriminant

How can we use class information to produce a projection that improves the separability of the classes? Linear Discrimant Analysis is designed to solve this probken by projecting the data linearly onto a subspace in such a way that two classes have maximum separability.

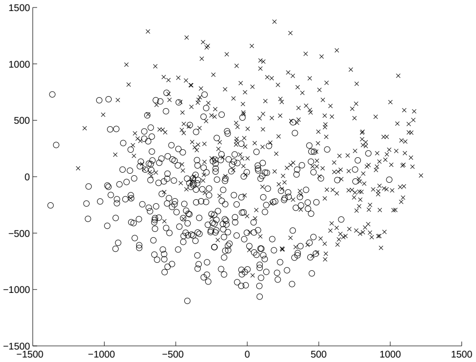


Figure 5.1: Projection of data onto two dimensions, formed by the two principal components of the data. The fives are plotted with an 'x', and the threes as an 'o'. We can see that the classes are partially separated in this two dimensional projection, but there is a good deal of class overlap.

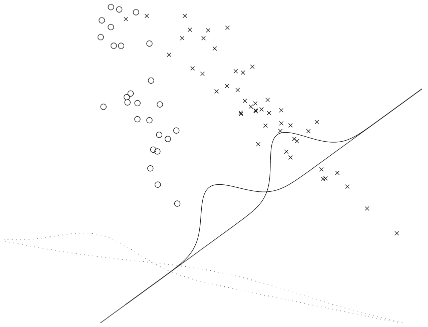


Figure 5.2: Two linear projections of two data classes. The dotted lines represent the distributions for the projection that maximises the difference between the projected means. The full curves are Fisher's projection. Fisher's projection clearly provides a better one-dimensional measure by which we can separate the classes.

5.2.1 One dimensional projection

We restrict attention here to two classes of data – the generalisation to more classes is given in the subsequent section (although only the algorithm is described). Also, for simplicty, we will project the data down to one dimension. The algorithm in a later section deals with the higher dimensional multiple class case, and is known as canonical variates.

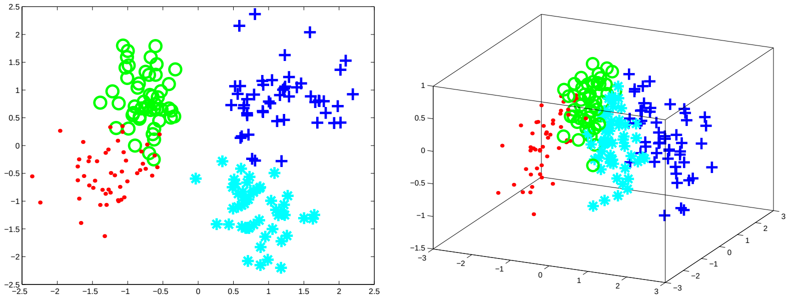
Gaussian Assumption

Assume that, for each class, we can model the data with a Gaussian. That is

$$p(x_1) = N(\mu_1; S_1), \qquad p(x_2) = N(\mu_2; S_2) \tag{5.2.1}$$

We now wish to project these Gaussians down onto one dimension

$$y_1^\mu = w^\mathsf{T} x_1^\mu, \qquad y_2^\mu = w^\mathsf{T} x_2^\mu \tag{5.2.2}$$



(a) Two dimensional projection (b) Three dimensional projection

Figure 5.3: Projection of the Leptograpsus Crab data onto the canonical variate directions. (a) Two dimensional projection. Note that the data is almost perfectly separated by this simple projection method. (b) Increasing the dimension of the projection to three does not significantly improve the separability of the projected classes. This can be expected since the Eigenvalues are 7.6, 3.2 and 0.15. That is, the third direction contributes very little to separating the classes.

where y_1^μ is the projection of an input x^μ that is in class one. Similarly, y_2^μ is the projection of a datapoint that is in class 2. We want to find w such that, in some sense, there is maximal separability of the classes in the one dimensional projections. The aim of this is that a classifier can then be made based on where, along this one dimension, a novel input lies. Because the projection is linear, the projected distributions onto the one dimension are also Gaussian,

$$p(y_1) = N\left(m_1; \sigma_1^2\right), \qquad p(y_2) = N\left(m_2; \sigma_2^2\right) \tag{5.2.3}$$

$m_1 = w^\mathsf{T}\boldsymbol{\mu}_1$, $\sigma_1^2 = w^\mathsf{T}S_1 w$ $m_2 = w^\mathsf{T}\boldsymbol{\mu}_2$, $\sigma_2^2 = w^\mathsf{T}S_2 w$. We could search for a projection w such that the means of the Gaussians in the one dimension are maximally separated (subject to the constraint that $|w| = 1$. However, if the variances in the one dimension are large, there could be a large overlap still in the classes. Fisher proposed therefore to use the objective function

$$\frac{(m_1 - m_2)^2}{\pi_1\sigma_1^2 + \pi_2\sigma_2^2} \tag{5.2.4}$$

where π_i represents the fraction of the dataset in class i. What this does is tries to maximize the separation of the projected means whilst at the same time penalising projections of large variance, see fig(5.2). Note that this objective function is invariant to linear rescaling of w, so that there is no need to include a restriction that $w^\mathsf{T}w = 1$. The optimal w is then given by

$$w \propto S_w^{-1}(m_2 - m_1) \tag{5.2.5}$$

where $S_w = \pi_1 S_1 + \pi_2 S_2$.

5.3 Canonical Variates

Canonical Variates generalises linear discriminants to more than two classes and more than one projected dimension. We only state the algorithm here – the interested reader may consult Bishop's book.

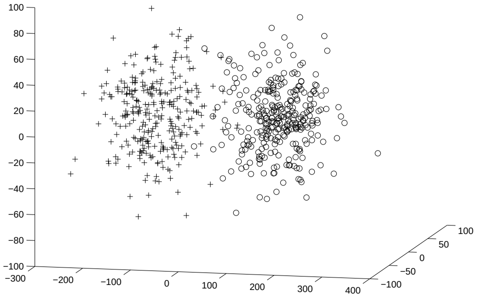


Figure 5.4: The projection of the two classes using canonical variates. Here we project onto three dimensions. Note how the data is well separated in the projections, and indeed, is almost linearly separated using this projection.

1. For each class form a covariance matrix S_k and mean m_k. Define

$$S = \sum_{k=1}^{c} N_k S_k \tag{5.3.1}$$

   where N_k is the number of datapoints in class k, and c is the total number of classes.

2. Find m the mean of the whole dataset and m_k, the mean of the each class k. Form

$$S_B = \sum_{k=1}^{c} N_k \left(m_k - m\right)\left(m_k - m\right)^{\mathsf{T}} \tag{5.3.2}$$

   To project onto an m dimensional space, the optimal projection matrix W corresponds to the first m eigenvectors of $S^{-1}S_B$. The projections are then given by $y = W^{\mathsf{T}}x$.

5.3.1 Using Canonical variates on the Digits Data

We can apply the method of canonical variates as described above to project the digit data onto a small number of dimensions (in the code below we project onto three dimensions). We use here 600 examples of a three and 600 examples of a five. Thus, overall , there are 1200 examples which lie in a 784 (28×28 pixels) dimensional space. Since there are more datapoints than dimensions in the space, the points cannot, a priori, be trivially separated by a linear decision boundary. Note how the projection onto three dimensions enables the data to be separated almost perfectly, see fig(5.4) Canonical variates is a useful method for dimension reduction for labelled data, preserving much more class relevant information in the projection than PCA. We can use the lower dimensional representations to help visualise the data and also for use in further processing such as building a classifier.

6 Linear Parameter Models

6.1 Introduction

Consider the data in fig(6.1), in which we plot the number of chirps per second for crickets, versus the temperature in degrees Fahrenheit. A biologist believes that there is a simple relation between the number of chirps and the temperature. Modelling such a relation is a regression problem. The biologist decides to make a straight line fit :

$$c = a + bt \tag{6.1.1}$$

where she needs to determine the parameters a and b. How can she determine these parameters based on the training data $(c^\mu, t^\mu), \mu = 1, \ldots, 15$? For consistency with our previous notations, let us use y rather than c, and x in place of t, so that our model is $y = a + bx$. The sum squared training error is

$$E(a, b) = \sum_{\mu=1}^{P} (y^\mu - a - bx^\mu)^2 \tag{6.1.2}$$

Differentiating with respect to a, we find

$$\sum_\mu (y^\mu - a - bx^\mu) = 0 \tag{6.1.3}$$

Differentiating with respect to b, we find

$$\sum_\mu (y^\mu - a - bx^\mu)x^\mu = 0 \tag{6.1.4}$$

Dividing by P, we thus have two simultaneous linear equations

$$\begin{aligned} \langle y \rangle - a - b \langle x \rangle &= 0 && (6.1.5) \\ \langle yx \rangle - a \langle x \rangle - b \left\langle x^2 \right\rangle &= 0 && (6.1.6) \\ & && (6.1.7) \end{aligned}$$

where we used the notation $\langle \cdot \rangle$ to denote $\frac{1}{P}\sum_{\mu=1}^{P} \cdot$ .We can easily solve these linear equations to determine a and b. The important thing to note about this regression model is that the *parameters* only appear in a linear fashion. We could also, more conveniently write our model as

$$y = w^\mathsf{T} \boldsymbol{\phi} \tag{6.1.8}$$

where $w = (a, b)^\mathsf{T}$ and $\boldsymbol{\phi} = (1, x)^\mathsf{T}$. The training error then is

$$E(w) = \sum_{\mu=1}^{P} (y^\mu - w^\mathsf{T} \boldsymbol{\phi}^\mu)^2 \tag{6.1.9}$$

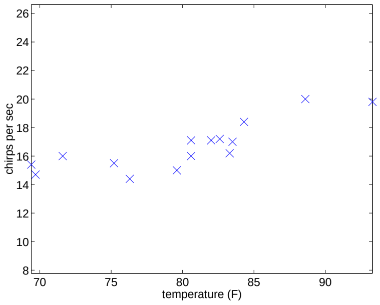


Figure 6.1: Data from crickets – the number of chirps per second, versus the temperature in Fahrenheit.

where $\boldsymbol{\phi}^\mu = (1, x^\mu)^\mathsf{T}$. We now wish to determine the parameter vector w. Writing out the error in terms of the components,

$$E(w) = \sum_{\mu=1}^{P}(y^\mu - \sum_i w_i \phi_i^\mu)(y^\mu - \sum_j w_j \phi_j^\mu) \tag{6.1.10}$$

Differentiating with respect to w_k, this gives

$$\sum_\mu y^\mu \phi_k^\mu = \sum_i w_i \sum_\mu \phi_i^\mu \phi_k^\mu \tag{6.1.11}$$

or, in matrix notation,

$$\sum_\mu y^\mu \boldsymbol{\phi}^\mu = \sum_\mu \boldsymbol{\phi}^\mu (\boldsymbol{\phi}^\mu)^\mathsf{T} w \tag{6.1.12}$$

Hence, the solution is

$$w = \left(\sum_\mu \boldsymbol{\phi}^\mu (\boldsymbol{\phi}^\mu)^\mathsf{T}\right)^{-1} \sum_\mu y^\mu \boldsymbol{\phi}^\mu \tag{6.1.13}$$

Putting in the actual data, we get $a = -0.3091$, $b = 0.2119$. The fit is plotted in fig(6.2). Although the solution is written in terms of the inverse matrix, we never actually compute the inverse numerically; we use instead Gaussian elimination[2]. In MATLABthis is implemented using the slash function \.

6.1.1 Regression and PCA

In an earlier chapter, we discussed using PCA to reduce the dimensionality of data, based on the idea that data may lie close to a low dimensional hyperplane. Since a line is a low dimensional hyperplane, one may wonder what the difference is between using PCA to fit a line and the above regression approach. The answer is that the objective functions are different. Regression finds a line that minimizes the vertical distance between a datapoint and the line; PCA finds a line that minimizes the distance between a datapoint and the line – see fig(6.2).

6.2 Linear Parameter Models

A linear parameter model is defined as

$$y(x) = w^\mathsf{T} \boldsymbol{\phi}(x) \tag{6.2.1}$$

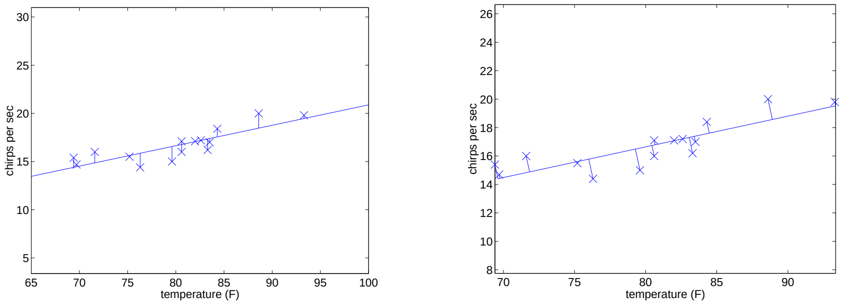


Figure 6.2: Left: Straight line regression fit to the cricket data. Right: PCA fit to the data. In regression we minimize the residuals – the fit represents the shortest vertical distances. In PCA the fit minimizes the orthogonal projections to the line.

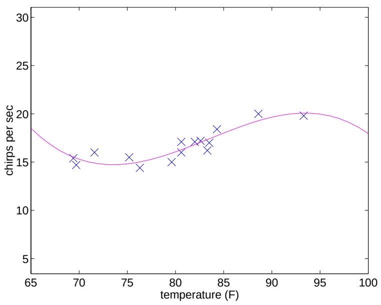


Figure 6.3: Cubic polynomial fit to the cricket data.

As we saw above, straight line regression fits to data are examples of this. If we choose the coefficients of the vector $\boldsymbol{\phi}$ to be non-linear functions of x, then the mapping $x \rightarrow y$ will be non-linear. The phrase "linear" model here refers to the fact that the model depends on its *parameters* in linear way.

6.2.1 Training LPMs

In the derivation above, there was nothing specific about the form of $\boldsymbol{\phi}$. Hence, the solution in equation (6.2.5) holds in general. That is, you simply put in a different $\boldsymbol{\phi}$ vector if you wish to find a new solution. For example, consider the case of fitting a cubic function $y = w_1 + w_2 x + w_3 x^2 + w_4 x^3$ to the data. In this case, we would choose

$$\boldsymbol{\phi} = \left(1, x, x^2, x^3\right)^{\mathsf{T}} \tag{6.2.2}$$

The solution has the same form, except w is now a 4 dimensional vector

Choosing between Different Models

How would we decide if a straight line fit is preferable to a cubic polynomial fit? We saw in the previous chapter that a general way to address this problem is to use some validation data to test how accurate each model predicts the validation data. The more accurate model on the validation data would then be preferred.

6.2.2 Regularisation and numerical stability

It should be fairly clear from the above that all polynomial regression fits are simply special cases of LPMs. Also, the more terms there are in polynomial, the more curved can be the fit to the data. One way to penalise too complex models is to use a penalty term

$$E_{reg}(w) = w^{\mathsf{T}} w \tag{6.2.3}$$

The regularised training error is then

$$E_{regtrain}(w, \lambda) = \sum_{\mu=1}^{P} (y^{\mu} - w^{\mathsf{T}} \boldsymbol{\phi}^{\mu})^2 + \lambda w^{\mathsf{T}} w \tag{6.2.4}$$

If we differentiate the regularised training error to find the optimal w for a given λ, we find: Hence, the solution is

$$w = \left(\sum_{\mu} \boldsymbol{\phi}^{\mu} (\boldsymbol{\phi}^{\mu})^{\mathsf{T}} + \lambda I \right)^{-1} \sum_{\mu} y^{\mu} \boldsymbol{\phi}^{\mu} \tag{6.2.5}$$

where I is the $n \times n$ identity matrix and $n = dim(w)$. Another beneficial aspect of using a quadratic penalty term is that the solution is more numerically stable – this can be a problem in cases where there is limited training data. We can determine λ by using a validation set.

6.2.3 Higher Dimensional Outputs

It is straightforward to generalise the above framework to cases where there is more than one output variable – rather there is an output vector y:

$$y_i(x) = w_i^{\mathsf{T}} \boldsymbol{\phi}(x) \tag{6.2.6}$$

The mathematics follows similarly to before, and this is left as an exercise for the interested reader.

6.2.4 Classification

One way to adapt the LPM model to classification is to use $p(c = 1|x) = \sigma(w^{\mathsf{T}} \boldsymbol{\phi}(x))$. The logistic regression model simply used a special case in which the vector $\boldsymbol{\phi}(x) = x$. However, there is nothing to stop us using this more general method. The nice thing is that the decision boundary is then a non-linear function of x. Clearly, instead of using the euclidean square distance as the error measure, we now use the log-likelihood, exactly as in the chapter on logistic regression. Again, however, the training to find w will not be so straightforward, since the objective function is not quadratic. However, the surface remains well behaved so that finding a solution is not numerically difficult. We leave it as an exercise for the reader to work out the details.

6.3 Radial Basis Functions

A popular choice for the vector $\boldsymbol{\phi}(x)$ is the radial basis function :

$$\phi_i(x) = e^{-\frac{1}{2\alpha^2}\left(x - m^i\right)^2} \tag{6.3.1}$$

where the vectors $m^i, i = 1, \ldots, M$ define M centres in the input space. The parameter α determines the width of each basis function. These basis functions

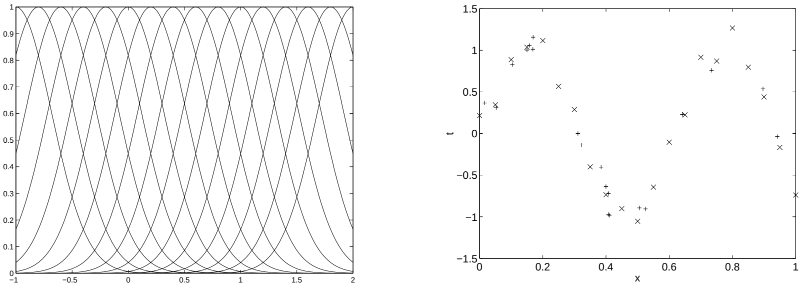


Figure 6.4: Left: A set of radial basis functions, $\alpha = 5$, with $m = -1, -0.8, -0.6, \ldots, 2$. Right: Data to be fitted. The × are the training points, and the + are the validation points.

are bump shaped, with the position of the bump being given by m and the width by α. An example is given in fig(7.8)(Left) in which several RBFs are plotted. In regression, we can then use a linear combination of these "bumps" to fit the data. For example consider fitting the data in fig(7.8)(Right).

Setting α

We use the validation data to set α. Throughout these experiments, I set the regularisation parameter $\lambda = 0.0001$. In principle one could use the validation set to optimise over both α and λ. In fig(6.5) we plot the validation error as a function of α. Based on this graph, we can find the best value of α; that which minimises the validation error. The predictions are also given in fig(6.5).

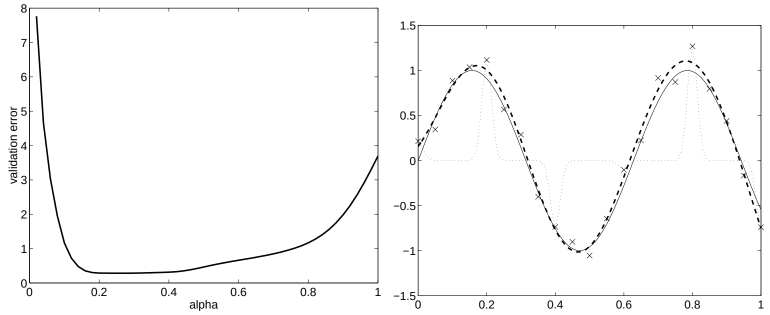


Figure 6.5: Left: The validation error as a function of the basis function width. Right: The predictions. The solid line is the correct underlying function $\sin(10x)$; the dashed line is the best predictor based on the validation set. The dotted line is the worst predictor based on the validation set.

6.4 The curse of Dimensionality

We saw that using radial basis functions we can get good predictions, provided that we choose appropriate basis functions (set the widths correctly). It seems intuitively clear that if the data has non-trivial behaviour over some region in x, then we need to cover the region of x space fairly densely with "bump" type functions. In the above case, we used 16 basis functions for this one dimensional space. In 2 dimensions, we can also use bump type functions. However, we now need to cover a 2 dimensional space. If we wish to cover it to the same discretisation level, we would need $16^2 = 256$ basis functions. In an n dimensional input space, we would need 16^n functions. This is an extremely rapidly growing function of n so that in 10 dimensions, we would need $16^{10} = 10^{12}$ basis functions. This means we would have to solve linear systems in 10^{12} variables! This cannot be easily done. This explosion in the apparent number of basis functions required is the famous "curse of dimensionality".

A possible solution is to make the basis functions very broad to cover more of the high dimensional space. However, this will mean a lack of flexibility of the fitted function. Another approach is to place basis functions centred on the training input points that we have, and add some more basis functions randomly placed close to the training inputs. The rational behind this is that when we come to do prediction, we will most likely see novel x that are close to the training points – we do not need to make "accurate" predictions over all the space.

A further approach is to make the positions of the basis functions adaptive, allowing them to be moved around in the space to minimise the error. This approach motivates the neural network models.

The criticism of the curse of dimensionality is, in my humble opinion, rather weak, and good results are often obtained by using basis functions which are dense around the training inputs.

6.5 Summary

- Linear Parameter models are regression models that are linear in the parameters.
- They are very easy to train (no local minima).
- Criticised in high dimensional input spaces due to the curse of dimensionality.
- Judicious placement of the basis functions on close to the training inputs is a workaround for the curse of dimensionality. Otherwise we need to optimise the placement of the basis functions – that is, use neural networks.
- Easily adapted to classification (though the training is now more difficult and needs to be solved using optimisation).

7 Layered Neural Networks

7.1 Sequential Layered Processing

In natural systems, information processing is often found to occur in stages. For example, in human vision, the light falling on the retina is transformed first in a non-linear logarithmic fashion. Local parts of the image are then "recognised" by neurons specialised to respond to particular local image patterns. The information from these "feature" extraction neurons is then fed into subsequent layers which correspond to higher cognitive functions. Artificial layered networks mimic such sequential processing.

In this chapter, we shall consider that each "neuron" or processing unit computes a deterministic function of its input. In this sense :*Neural Networks are graphical representations of functions.*

7.2 The Perceptron

The perceptron is essentially just a single neuron like unit that computes a non-linear function y of its inputs x,

$$y = g\left(\sum_j w_j x_j + \mu\right) = g\left(w^T x + \mu\right) \tag{7.2.1}$$

where the weights w encode the mapping that this neuron performs. Graphically, this is represented in fig(7.1). We can consider the case of several outputs as follows:

$$y_i = g\left(\sum_j w_{ij} x_j + \mu_i\right)$$

and can be used to model an input-output mapping $x \to y$, see fig(7.1)(right). Coupled with an algorithm for finding suitable weights, we can use a perceptron for regression. Of course, the possible mappings the perceptron encodes is rather restricted, so we cannot hope to model all kinds of complex input-output mappings

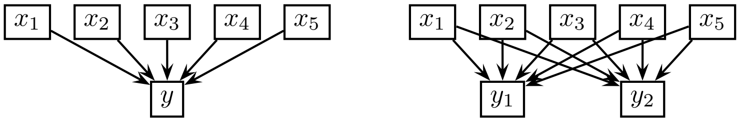
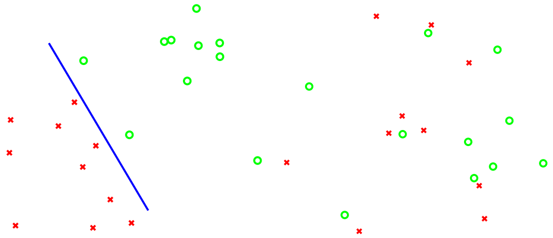


Figure 7.1: (Left) A simple perceptron. We use square boxes to emphasise the deterministic nature of the network. (Right) We can use two perceptrons with weights w_1 and w_2 to model a mapping $(x_1, x_2, x_3, x_4, x_5) \to (y_1, y_2)$



(a) A linearly separable problem

(b) A non-linearly separable problem

Figure 7.2: Linear separability: The data in (a) can be classified correctly using a hyperplane classifier such as the simple perceptron, and the data is termed linearly separable. This is not the case in (b) so that a simple perceptron cannot correctly learn to classify this data without error.

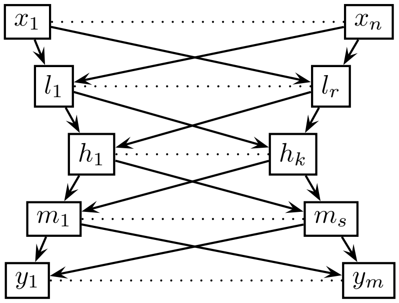


Figure 7.3: A multilayer perceptron (MLP) with multiple hidden layers, modeling the input output mapping $x \to y$. This is a more powerful model than the single hidden layer, simple perceptron. We used here boxes to denote the fact that the nodes compute a deterministic function of their inputs.

successfully. For example, consider the case in which $g(x) = \Theta(x)$ – that is, the output is a binary valued function ($\Theta(x) = 1$ if $x \geq 0$, $\Theta(x) = 0$ if $x < 0$) . In this case, we can use the perceptron for binary classification. With a single output we can then classify an input x as belonging to one of two possible classes. Looking at the perceptron, equation (7.2.1), we see that we will classify the input as being in class 1 if $\sum_j w_j x_j + \mu \geq 0$, and as in the other class if $\sum_j w_j x_j + \mu < 0$. Mathematically speaking, the decision boundary then forms a hyperplane in the x space, and which class we associate with a datapoint x depends on which side of the hyperplane this datapoint lies, see fig(7.2).

7.3 Multilayer Perceptrons

If the data that we are modeling is not linearly separable, we have a problem since we certainly cannot model this mapping using the simple perceptron. Similarly, for the case of regression, the class of function mappings that our perceptron forms is rather limited, and only the simplest regression input-output mappings will be able to be modelled correctly with a simple perceptron. These observations were pointed out in 1969 by Minsky and Papert and depressed research in this area for several years. A solution to this perceived problem was eventually found which included “hidden” layers in the perceptron, thus increasing the complexity of the

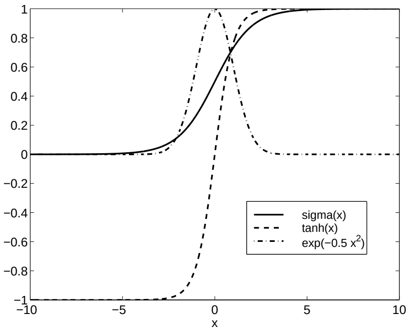


Figure 7.4: Common types of transfer functions for neural networks.

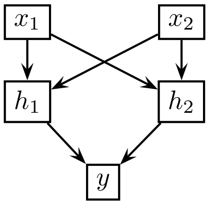


Figure 7.5: A MLP with one hidden layer.

Transfer Functions

mapping. Each hidden node computes a non-linear function of a weighted linear sum of its inputs. The specific non-linearity used is called the *transfer function*. In principle, this can be any function, and different for each node. However, it is most common to use an S-shaped (sigmoidal) function of the form $\sigma(x) = 1/(1 + e^{-x})$. This particular choice is mathematically convenient since it has the nice derivative property $d\sigma(x)/dx = \sigma(x)(1 - \sigma(x))$. Another popular choice is the sigmoidal function $\tanh(x)$. Less "biological" transfer functions include the Gaussian, $e^{-\frac{1}{2}x^2}$, see fig(7.4). For example, in fig(7.3), we plot a simple single hidden layer function,

$$h_1 = \sigma\left(w_1^T x + b_1\right), h_2 = \sigma\left(w_2^T x + b_2\right), y = r(v^T h + b_3) \tag{7.3.1}$$

where the adaptable parameters are $\boldsymbol{\theta} = \{w_1, w_2, v, b_1, b_2, b_3\}$. Note that the output function $r(\cdot)$ in the final layer is usually taken as the idendity function $r(x) = x$ in the case of regression – for classification models, we use a sigmoidal function. The biases, b_1, b_2 are important in shifting the position of the "bend" in the sigmoid function, and b_3 shifts the bias in the output.

Generally, the more layers that there are in this process, the more complex becomes the class of functions that such MLPs can model. One such example is given in fig(7.3), in which the inputs are mapped by a non-linear function into the first layer outputs. In turn, these are then fed into subsequent layers, effectively forming new inputs for the layers below. However, it can be shown that, provided that there are sufficiently many units, a single hidden layer MLP can model an arbitrarily complex input-output regression function. This may not necessarily give rise to the most efficient way to represent a function, but motivates why we concentrate mainly on single hidden layer networks here.

7.3.1 Understanding Neural Networks

There are a great number of software packages that automatically set up and train the networks on provided data. However, following our general belief that our predictions are only as good as our assumptions, if we really want to have some faith in our model, we need to have some insight into what kinds of functions neural networks are.

The central idea of neural networks is that each neuron computes some function of a linear combination of its inputs:

$$h(x) = g(w^T x + b) \tag{7.3.2}$$

where g is the transfer function, usually taken to be some non-linear function. Alternatively, we can write

$$h(x) = g(a(x)) \tag{7.3.3}$$

where we define the *activation* $a(x) = w^T x + b$. The parameters for the neuron are the weight vector w and bias b. Each neuron in the network has its own weight and bias, and in principle, its own transfer function. Consider a vector $w^\perp$ defined to be orthogonal to w, that is, $w^T w^\perp = 0$. Then

$$\begin{aligned} a(x + w^\perp) &= \left(x + w^\perp\right)^T w + b && (7.3.4)\\ &= x^T w + b + \underbrace{\left(w^\perp\right)^T w}_{0} && (7.3.5)\\ &= a(x) && (7.3.6)\end{aligned}$$

Since the output of the neuron is only a function of the activation a, this means that any neuron has the same output along directions x which are orthogonal to w. Such an effect is given in fig(7.6), where we see that the output of the neuron does not change along directions perpendicular to w. This kind of effect is general, and for any transfer function, we will always see a ridge type effect. This is why a single neuron cannot achieve much on its own – essentially, there is only one direction in which the function changes (I mean that unless you go in a direction which has a contribution in the w direction, the function remains the same). If the input is very high dimensional, we only see variation in one direction.

Combining Neurons

In fig(7.7) we plot the output of a network of two neurons in a single hidden layer. The ridges intersect to produce more complex functions than single neurons alone can produce. Since we have now two neurons, the function will not change if we go in a direction which is simultaneously orthogonal to both w_1 and w_2. In this case, x is only two dimensional, so there is no direction we can go along that will be orthogonal to both neuron weights. However, if x where higher dimensional, this would be possible. Hence, we now have variation along essentially two directions.

In general, if we had K neurons interacting in a single hidden layer in this way, we would essentially have a function which can vary along K independent directions in the input space.

7.4 Training multi-layered perceptrons

To a statistician, neural networks are a class of non-linear (adaptive basis function) models. Let us consider, for convenience, only a single output variable y. Given

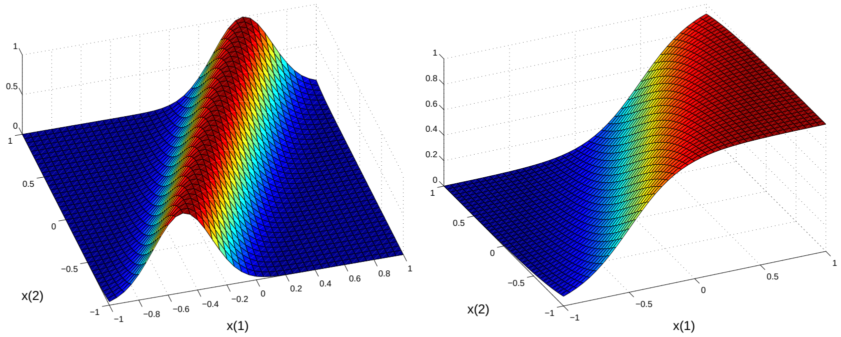


Figure 7.6: The output for a single neuron, $w = (-2.5, 5)^T$, $b = 0$. Left: The network output using the transfer function $\exp(-0.5x^2)$. Right: using the transfer function $\sigma(x)$. Note how the network output is the same along the direction perpendicular (orthogonal) to w, namely $w^{\perp} = \lambda(2, 1)^T$.

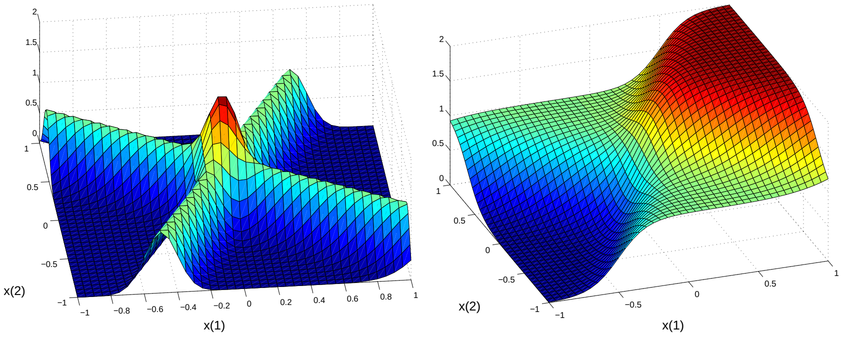


Figure 7.7: The combined output for two neurons, $w_1 = (-5, 10)^T$, $b_2 = 0$, $w_2 = (7, 5)^T$, $b_2 = 0.5$. The final output is linear, with weights $v = (1, 1)^T$ and zero bias. Left: The network output using the transfer functions $\exp(-0.5x^2)$. Right: using the transfer function $\sigma(x)$ – this is exactly the function in equation (7.3.1) with r the identity function.

a set of input-output pairs, $D = \{(x^\mu, y^\mu), \mu = 1, \ldots, P\}$, how can we find appropriate “weights” $\boldsymbol{\theta}$ that minimise the error that the network makes in fitting this function? In neural-network terminology, we would define an “energy” function that measures the errors that the network makes, and then try to minimise this function with respect to $\boldsymbol{\theta}$.

Regression

For example, a suitable choice of energy or error function for regression might be

$$E_{train}(\boldsymbol{\theta}) = \sum_\mu (y^\mu - f(x^\mu, \boldsymbol{\theta}))^2 \tag{7.4.1}$$

where $f(x^\mu, \boldsymbol{\theta})$ is the output of the network for input x^μ, given that the parameters describing the network are $\boldsymbol{\theta}$. We can train this network by any standard (non-linear) optimisation algorithm, such as conjugate gradient descent.

Classification

A suitable choice of energy or error function to minimise for classification is the negative log likelihood (if $y^\mu \in \{0, 1\}$)

$$E_{train}(\boldsymbol{\theta}) = -\sum_\mu (y^\mu \log f^\mu + (1 - y^\mu) \log(1 - f^\mu)) \tag{7.4.2}$$

where $f^\mu = f(x^\mu, \boldsymbol{\theta})$. In this case, we would need that the final output $r(x)$ is bounded between 0 and 1 in order that it represents a probability. The case of more than two classes is handled in a similar way using the so-called soft-max function (see Bishops book for references).

Regularisation

In principle, the problem of training neural networks is equivalent to the general statistical problem of fitting models to data. One of the main problems when fitting complex non-linear models to data is how to prevent “overfitting”, or, more generally, how to select the model that not only fits the data, but also generalises well to new data. We have already discussed this issue in some generality, and found that one approach is to use a penalty term which encourages smoother functions. In the case of MLPs, smoother functions can be encouraged if we penalise large weight values. The reason for this is that the larger the weights w^i are, the more rapidly the function can change as x changes (since we could flip from close to one near saturated region of the sigmoid to the other with only a small change in x).

A term which penalises large weights,

$$E_{regtrain}(\boldsymbol{\theta}) = E_{train}(\boldsymbol{\theta}) + \lambda \boldsymbol{\theta}^T \boldsymbol{\theta} \tag{7.4.3}$$

We can set λ as usual by using a validation set.

7.4.1 Single Hidden Layer

A MLP with a single hidden layer is

$$f(x, \boldsymbol{\theta}) = r\left(\sum_{i=1}^{K} v_i g(w_i \cdot x + b_i) + b\right) \tag{7.4.4}$$

an example of which is given in fig(7.3).

Regression

In the case of regression, we would use an output function r to be the identity, and the squared output to form the error[1]:

$$E(\boldsymbol{\theta}) = \sum_{\mu=1}^{P} (f(x^\mu, \boldsymbol{\theta}) - y^\mu)^2 + \lambda \sum_{k=1}^{K} (w_k)^T w_k \tag{7.4.5}$$

To use the conjugate gradients algorithm to optimise this objective function, we need to know the derivatives with respect to all the parameters $\partial E/\partial \theta_i$.

$$\frac{\partial E}{\partial \theta_i} = 2 \sum_{\mu=1}^{P} (f(x^\mu, \boldsymbol{\theta}) - y^\mu) \frac{\partial f(x^\mu, \boldsymbol{\theta})}{\partial \theta_i} + 2 \sum_{k} \sum_{j=1}^{dim(w_k)} w_{j,k} \frac{\partial w_{j,k}}{\partial \theta_i} \tag{7.4.6}$$

The final term is zero unless we are differentiating with respect to a parameter that is included in the regularisation term. If θ_i is included in the regularisation term, then the final term simply is $2\theta_i$. All that is required then is to calculate the derivatives of f with respect to the parameters. This is a straightforward exercise in calculus, and we leave it to the reader to show that, for example,

$$\frac{\partial f(x^\mu, \boldsymbol{\theta})}{\partial v_1} = g(w_1^T x^\mu + b_1) \tag{7.4.7}$$

and

$$\frac{\partial f(x^\mu, \boldsymbol{\theta})}{\partial w_{1,2}} = v_2 g'(w_2^T x^\mu + b_2) x_1^\mu \tag{7.4.8}$$

where $g'(x)$ is the derivative of $g(x)$.

7.4.2 Back Propagation

In computing the gradient of the error function, naively it appears that we need of the order of PW^2 operations (if W is the number of parameters in the model and P is the number of training patterns), since computing the output of the network involves roughly W summations for each of the P patterns, and the gradient is a W-dimensional vector. The essence of the backpropagation procedure is that the gradient can instead by computed in order PW operations. If the training set is very large, standard computation of the gradient over all training patterns is both time-consuming and sensitive to round-off errors. In that case, "on-line learning", with weight updates based on the gradient for individual patterns, offers an alternative. Back propagation is most useful in cases where there are more than one hidden layer in the network. In this case, the gradient can be computed more efficiently, and time saved therefore to find the optimal parameters.

7.4.3 Training ensembles of networks

A problem with neural networks is that they are difficult to train. This is because the surface of the error function $E(\boldsymbol{\theta})$ is very complicated and typically riddled with local minima. *No (practical) algorithm can guarantee to find the global optimum of the error surface.* Indeed, depending on the initial conditions that we use, the parameters found by the optimisation routine will in general be different. How are we to interpret these different solutions? Perhaps the simplest thing to do is to see which of the solutions has the best error on an independent validation

[1] There is no need to penalise the biases, since they only really affect a translation of the functions, and don't affect how bumpy the functions are.

set. Many algorithms have been proposed on how to combine the results of the separate networks into a single answer and for computing error bars that indicate the reliability of this answer. Imagine that we have used optimisation several times, and found the different solutions $\boldsymbol{\theta}^i, i = 1, \ldots, M$. One simple approach (for regression) is to combine the outputs of each of the trained models,

$$\bar{f}(x) = \frac{1}{M} \sum_{i=1}^{M} f(x, \boldsymbol{\theta}^i) \tag{7.4.9}$$

This is also useful since we can make an estimate of the variance in the predictions at a given point,

$$\text{var}(f(x) = \frac{1}{M} \sum_{i=1}^{M} \left(f(x, \boldsymbol{\theta}^i) - \bar{f} \right)^2 \tag{7.4.10}$$

This can then be used to form error bars $\bar{f}(x) \pm \sqrt{\text{var}(f(x))}$.

7.5 Adaptive Basis Function Networks

Linear weighted inputs

In neural networks, typically the output of each node (or neuron) in the network is some non-linear function of a linear combination of the nodes entering the network (the parents). That is,

$$y_i = g_i \left(\sum_j w_{ij} x_j + \mu_i \right) \tag{7.5.1}$$

As previously discussed, because the output of the node only depends on a linear combination of the inputs to the network node/neuron, essentially there is only variability in one direction in the input space (where by input I mean the inputs to the node). We can make a bump, but only a one dimensional bump, albeit in a high dimensional space. To get variability in more than one direction, we need to combine neurons together. Since it is quite reasonable to assume that we want variability in many dimensions in the input space, particularly in regions close to the training data, we typically want to make bumps near the data.

7.5.1 Adaptive Basis Functions

In the case of linear parametric models, we saw how we can approximate a function using a linear combination of fixed basis functions. Localised Radial Basis Functions($\exp(-(x-m)^2)$) are a reasonable choice for the "bump" function type approach. The output of this function depends on the *distance* between x and the centre of the RBF m. Hence, in general, the value of the basis function will change as x moves in any direction, apart from those that leave x the same distance from m, see fig(7.8). Previously, we suggested that a good strategy for placing centres of basis functions is to put one on each training point input vector. However, if there are a great number of training patterns, this may not be feasible. Also, we may wish to use the model for compression, and placing a basis function on each training point may not give a particularly high compression. Instead we could adapt the positions of the centres of the basis functions, treating these also as adaptable parameters. In general, an adaptive basis function network is of the form

$$y(x, \boldsymbol{\theta}) = \sum_i w_i \phi_i(x, b^i) \tag{7.5.2}$$

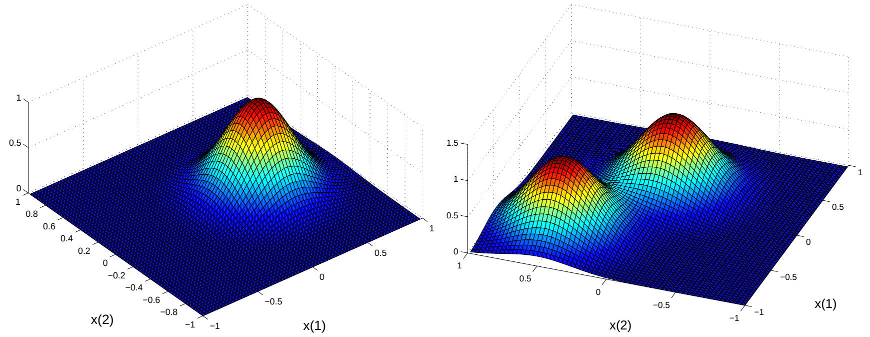


Figure 7.8: Left: The output of an RBF function $\exp(-\frac{1}{2}\left(x - m^1\right)^2/\alpha^2)$. Here $m^1 = (0, 0.3)^T$ and $\alpha = 0.25$. Right: The combined output for two RBFs, $m^2 = (0.5, -0.5)^T$.

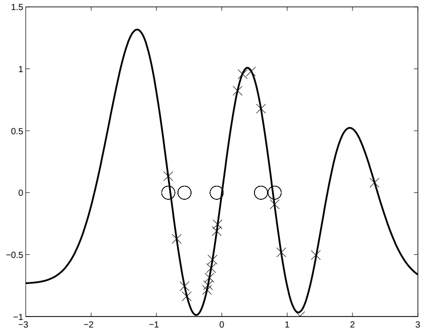


Figure 7.9: A RBF function using five basis functions. Note how the positions of the basis function centres, given by the circles, are not uniform.

where now each basis function $\phi_i(x, b^i)$ has potentially its own parameters that can be adjusted. $\boldsymbol{\theta}$ represents the set of all adjustable parameters. If the basis functions are non-linear, then the overall model is a non-linear function of the parameters.

7.6 Training Adaptive Basis Functions

Let us consider, for convenience, only a single output variable y. Given a set of input-output pairs, $D = \{(x^\mu, y^\mu), \mu = 1, \ldots, P\}$, how can we find appropriate paramters $\boldsymbol{\theta}$ that minimise the error that the network makes in fitting this function?

Regression

A suitable choice of energy or error function for regression is

$$E_{train}\left(\boldsymbol{\theta}\right) = \sum_{\mu} \left(y^\mu - f\left(x^\mu, \boldsymbol{\theta}\right)\right)^2 \qquad (7.6.1)$$

We can train this network by any standard (non-linear) optimisation algorithm, such as conjugate gradient descent.

However, one should always bear in mind that, in general, the training of complex non-linear models with many parameters is extremely difficult.

Classification

A suitable choice of energy or error function to minimise for classification is the negative log likelihood (if $y^\mu \in \{0, 1\}$)

$$E_{train}\left(\boldsymbol{\theta}\right) = -\sum_{\mu}\left(y^\mu \log f^\mu + \left(1 - y^\mu\right)\log(1 - f^\mu)\right) \tag{7.6.2}$$

where $f^\mu = f(x^\mu, \boldsymbol{\theta})$.

Regularisation

The smoothness of the RBF mapping is mainly determined by the width of the basis functions. The easiest approach is to use a validation set to determine α and not to regularise any of the other parameters.

Initialisation

The quality of the solutions is critically dependent on the initial parameter settings, in particular where we initially speculatively place the basis function centres.

One reasonable initialisation strategy is to place the centres on a randomly chosen subset of the data, and then solve for the hidden to output weights w easily (this is just a linearised parameter model if we consider the basis functions fixed).

Another approach is to use K-means clustering (see later chapter) to set the centres of the basis functions. Given the initial centres of the basis functions, we can solve for the weights easily. This gives an initial setting for the basis function and weight values.

Optimisation Strategies

Perhaps the most obvious thing to do is to treat both the weights w and basis function parameters b^i together as one parameter set, $\boldsymbol{\theta}$, and optimise the objective function with respect to $\boldsymbol{\theta}$.

An example is given in fig(7.9) where we see that the optimal solution (as found by the optimisation algorithm) produces a non-uniform placing of the basis function centres. However, there is another strategy which, in practice, may be preferable:

1. For fixed basis function parameters b^i, find the best weights w (this is easy to do since this is just corresponds to solving a linear system).
2. For fixed weights w, find the best basis function parameters. (This is the difficult step since there will typically be many basis function parameters, and the objective function depends in a highly non-linear way on the basis function parameters).

We can iterate these two stages to improve the solution. The slight practical advantage of this is that the parameter space in which we search for a solution to a non-linear optimisation problem is slightly reduced since we only optimise with respect to the b^i parameters.

7.6.1 Non-local Basis Functions

If we use basis functions that decay rapidly from a 'centre', as in the case $\exp(-(x-m)^2)$, the basis function value will always decay to zero once we are far away from the training data. In the case of binary classification and a logistic sigmoid for the class output, this may be reasonable since we would then predict any new datapoint far away from the training data with a complete lack of confidence, and any assignment would be essentially random. However, in regression, using say a linear combination of basis function outputs would always give zero far from the training data. This may give the erroneous impression that we are therefore extremely confident that we should predict an output of zero far away from the training data whilst, in realilty, this is simply an artefact of our model. For this reason, it is sometimes preferable to use basis functions that are non-local – that is, they have appreciable value over all space, for example, $(x-m)^2 log((x-m)^2)$. Whilst any single output will tend to infinity away from the training data, this serves to remind the user that, far from the training data, we should be wary of our predictions.

7.7 Committees

Drawbacks of the non-linear approaches we have looked at – neural networks and their cousins adaptive basis functions – are

1. Highly complex energy/error surfaces give rise to multiple solutions since global optima are impossible to find.
2. We have no sense of the confidence in the predictions we make (particularly in regression).

Whilst there are alternative (and in my view more attractive) approaches around these problems, we can exploit the variability in the solutions found to produce a measure of confidence in our predictions. The idea is to form a committee of networks from the solutions found. For example, for regression, we could train (say) M networks on the data and get M different parameter solutions $\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^M$. The average network function would then be

$$\bar{f}(x) = \frac{1}{M}\sum_{i=1}^{M} f(x, \boldsymbol{\theta}^i). \tag{7.7.1}$$

A measure of the variability in the predictions is given by the variance :

$$\mathrm{var}(f)(x) = \frac{1}{M}\sum_{i=1}^{M}(f(x, \boldsymbol{\theta}^i) - \bar{f}(x))^2. \tag{7.7.2}$$

A useful plot of confidence in our predictions is then to use one standard deviation error bars :

$$\bar{f}(x) \pm \sqrt{\mathrm{var}(f)(x)} \tag{7.7.3}$$

In fig(7.10) we give an example using a committee of six adaptive basis functions.

The committee idea is quite general and applicable to any model. Whilst this approach is rather heuristic and leaves some questions unanswered (why did we choose uniform weighting of the solutions for example) it is nevertheless an intuitive an reasonably robust way of gaining confidence in model predictions (these issues can be solved by a Bayesian treatment beyond the scope of this course). Note that the committee approach does not necessarily solve the issue of over confident regression predictions away from training data. As seen in fig(7.10) both the mean and confidence will collapse around zero as we move far from the training data. This is a good reason to use non-local basis functions in this case since typically, the variability will become unbounded as we move away from the training data.

7.8 Summary and Outlook

The field of neural networks has contributions from and makes contributions to many different areas. Although, ultimately, the motivation for much research has

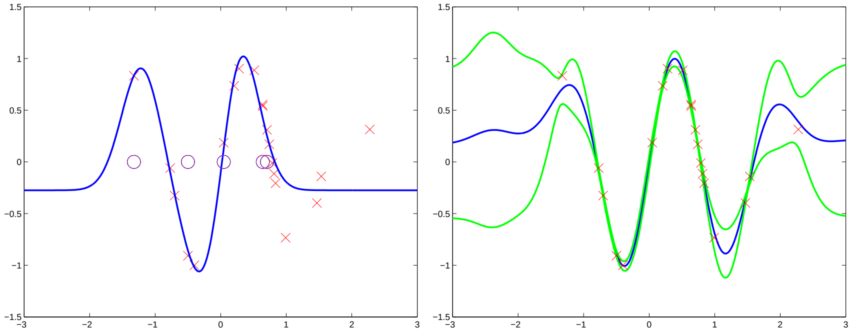


Figure 7.10: Left: A single solution using Adaptive Basis Functions to fitting the training data (crosses). The centres of the five basis functions are given by the circles. Right: A committee prediction from six individual solutions of the form given on the left. The central line is the average prediction – note how this still decays to zero away from the training data. The lines around the central line a one standard deviation confidence intervals.

been biological, there are many areas in which artificial neural networks can and have been used successfully. More specifically, these include areas where the underlying process behind data generation is highly non-linear, and there now exists techniques that are able to give some confidence and insight into the performance of such models.

Two features of artificial neural networks stand out as being of particular importance – their non-linearity, and stochasticity (although this latter aspect is not always exploited in many applications). These properties can be used to define local computation units which, when coupled together suitably, can combine to produce extremely rich patterns of behaviour, whether these be dynamic, or static input-output relationships. One of the most import consequences of neural network research has been to bring the techniques and knowledge of artificial intelligence and statistics much closer together. Typically it was the case that problems in artificial intelligence were tackled from a formal specification aspect. On the other hand, statistics makes very loose formal specifications, and lets the data try to complete the model. Neural networks can be seen as a statistical approach to addressing problems in artificial intelligence, obfuscating the need for formal specifications of how the program works – just learn how to do it from looking at examples. For example, rather then formally specifying what constitutes the figure “2”, a neural network can learn the (statistical) structure of “2”s by being asked to learn (find appropriate weights for) how to differentiate between “2”s and non-”2”s. This idea is especially powerful in the many human computer interaction applications where formally specifying, for example, what constitutes an individuals facial characteristics that differentiate them from others, is extremely difficult.

8 Autoencoders

8.1 Introduction

The general idea of autoencoders is that they are simply approximations of the identity mapping. We do not really need to invoke the concepts of neural networks to talk about these. However, many applications use neural networks to implement autoencoders.

Dimension Reduction

The major use of autoencoders is in dimension reduction, that is to replace a high N-dimensional vector x with a lower M-dimensional vector y. Clearly, this only makes sense when we have a set of data, $x^\mu, \mu = 1, \ldots, P$.

8.1.1 Linear Dimension Reduction (PCA)

In linear dimension reduction, $y = Wx + b$, where W is a $M \times N$ matrix and b is an M dimensional vector. We have already encountered one method of *linear* dimension reduction, PCA,

$$y = E^{\mathsf{T}}(x - m) \tag{8.1.1}$$

where E is the $M \times N$ matrix whose columns are the largest eigenvectors of the covariance matrix of the data, $C = \frac{1}{P}\sum_\mu (x^\mu - m)(x^\mu - m)^{\mathsf{T}}$, $m = \frac{1}{P}\sum_\mu x^\mu$.

The reconstruction of a higher dimensional vector, using the lower dimensional PCA representation y is

$$\tilde{x} = m + Ey \tag{8.1.2}$$

PCA was shown to be optimal in terms of squared reconstruction error, $E = \sum_\mu (\tilde{x}^\mu - x^\mu)^2$. We can represent the process of the mapping $x \to y \to \tilde{x}$ using a single hidden layer neural network, as in fig(8.1), in which the transfer function of each node (neuron) is the identity, $g(s) = s$. Such a network is known as an autoencoder, since the aim is to reproduce at the output layer, the same vector as at the input. That is, the network should try to encode as accurately as possible the identity mapping $x \to x$. The reason that this is non-trivial is that there is a reduced number of dimensions in the hidden layer, creating a lower dimensional "bottleneck" through which information must be squeezed. It is the activations of the units in this lower dimensional layer that we can then use to represent the higher dimensional data.

8.1.2 Manifolds : The need for non-linearity

Consider the situation in fig(8.2), where a "piece of paper" has been wrinkled to form a three dimensional object. However, to describe exactly the position of any point on the surface of the paper, we only need two co-ordinates, namely how far to go along y_1 and how far to go along y_2. Of course, the actual position of the surface point is a three dimensional vector, but it is only a function of y_1 and y_2. Clearly, in this case, x is a non-linear function of y. A *manifold* is a higher dimensional generalisation of the idea of a “piece of paper” – that is, we can describe points in a higher dimensional space using a lower dimensional coordinate system. (More correctly, a manifold is described by potentially a set of overlapping pieces of paper). In general, we might hope that data lies on such lower dimensional manifold. To discover this, or at least to approximate it, we need to make a non-linear model.

Non-linear transfer functions?

A first attempt to make a non-linear manifold would be to use the same autoencoder network as for PCA, however, now replacing the identity transfer function in the hidden layer with a non-linear function. It is a well known result (a little beyond the scope of these lectures) that this will *not* result in a better reconstruction error. That is : *for single hidden layer autoencoders, the optimal minimum squared reconstruction error solution is always PCA, regardless of the transfer function used.*

8.2 Non-linear Dimension Reduction

Non-linear Autoencoder

To obtain a more powerful model than PCA, we need both a non-linearity in the hidden layer transfer functions, *and* in the output layer transfer functions. To see this, consider a simple two dimensional manifold (see fig(8.3)):

$$x = (y_1, \sin(y_1 + y_2), \cos(y_1))^{\mathsf{T}} \tag{8.2.1}$$

In this case the optimal hidden layer activations would be $y_1 = x_1$, $y_2 = \sin^{-1}(x_2) - x_1$. Clearly, there are other possibilities available. Given y_1 and y_2, to make our reconstruction, we use

$$\tilde{x} = (y_1, \sin(y_2), \cos(y_1))^{\mathsf{T}} \tag{8.2.2}$$

If we use a neural network (by which we mean that the outputs of the hidden units are non-linear functions of a weighted linear combination of the units inputs), both the hidden unit transfer functions and output transfer functions need to (in general) be non-linear. Note that the above would need more than one hidden layer to be represented by an autoencoder. Graphically, we can represent a multiple hidden layer neural network autoencoder as in fig(8.4). In principle, no restriction on the form of the mappings from layer to layer need be made. However, it is common to use non-linear perceptron like mappings from layer to layer, so that the output of each node is a non-linear function of its linearly weighted inputs.

8.2.1 Training Autoencoders

The standard approach to training autoencoders is to use the sum squared reconstruction error. If $\boldsymbol{\theta}$ are the parameters of the autoecoder, then the autoencoder

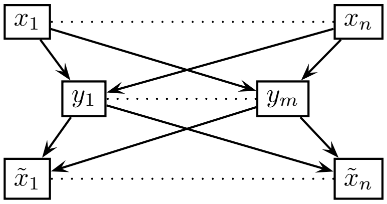


Figure 8.1: Autoencoder with a single hidden layer. If minimal reconstruction error is used, the optimal solution is given by PCA.

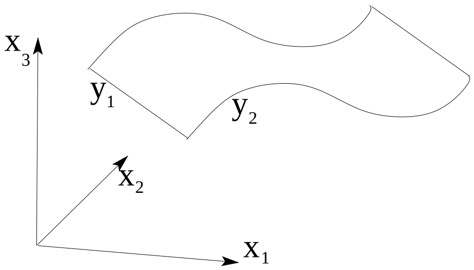


Figure 8.2: A two dimensional manifold embedded in a three dimensional space.

expresses a mapping $f(x, \boldsymbol{\theta})$. Since we want the output to resemble the input as closely as possible, we form the error:

$$E(\boldsymbol{\theta}) = \sum_{\mu=1}^{P} \left(x^\mu - f(x, \boldsymbol{\theta})\right)^2 \tag{8.2.3}$$

We then minimise this error with respect to the parameters $\boldsymbol{\theta}$. Normally, we will use an optimisation routine like scaled conjugate gradients to find the best parameters. This requires us to calculate the derivatives $\partial E / \partial \theta_i$. To calculate these derivatives efficiently, we use the back-propagation routine. Essentially, back-propagation just makes efficient use of the fact that information is passed between layers. We will not go into the details of back-propagation in this course, but this is an important issue in the practical training of networks, since it is much more efficient than calculating the derivatives in a naive way. There are many software packages that implement neural networks, and all use back-propagation.

8.3 Uses of Autoencoders

Preprocessing

Preprocessing of inputs x can have a major impact on the quality of the mapping that we wish to learn from inputs to targets. One of the most common preprocessing steps is dimension reduction, such as PCA. We can use the hidden unit

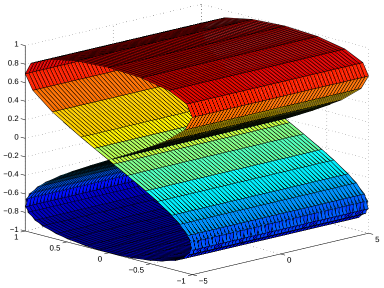


Figure 8.3: A two dimensional manifold embedded in a three dimensional space.

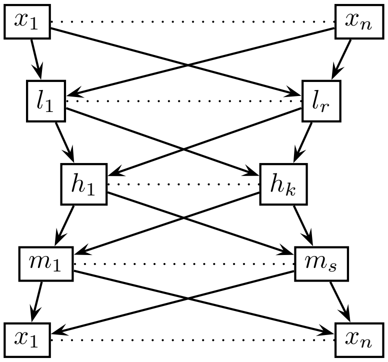


Figure 8.4: Autoencoder with multiple hidden layers. This is a more powerful autoencoder than the single hidden layer case, only provided that the hidden to output layers encode a non-linear function.

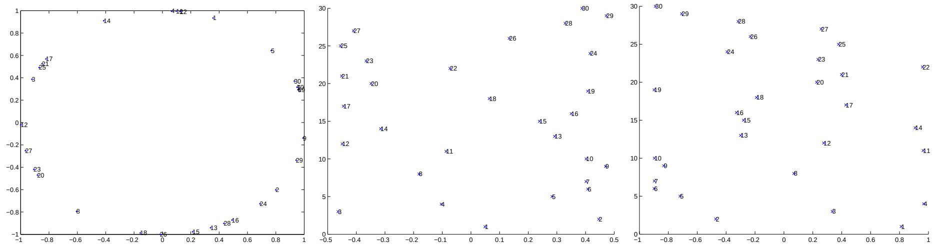


Figure 8.5: A two dimensional data set (left) represented by a one dimensional PCA (middle) and one dimensional autoencoder (right). Note that in the middle and right plots, the y axis is irrelevant and simply used to aid visual separation of the data.

activation values in an autoencoder to form a lower dimensional representation y of the input x, and then use this as our new input for any subsequent processing. Hence, autoencoders can be used for non-linear dimension reduction.

Visualisation

Visualisation of data can be very useful, and is a topic to which we shall return in a later chapter. If we squeeze the data through an autoencoder in which the bottleneck is only of two or three dimensions, then we can plot the resulting two or three dimensional points, and get a feeling for how the low dimensional (and by implication also the high-dimensional) representations of the data are distributed.

8.3.1 A Visualisation example

In fig(8.5)(left) we plot the original two dimensional data. In fig(8.5)(middle) we plot the one dimensional data PCA representation (the y axis is irrelevant and simply used to make the plot more readable). In fig(8.5)(right) we plot the one dimensional data autoencoder representation using a $2-4-1-4-2$ autoencoder architecture.

9 Data Visualisation

In data visualisation we attempt to gain intuition about the structure of a dataset. Typically, this method is unsupervised (does not make use of any target values), although supervised visualisation is also possible. We have seen how to use PCA to reduce the dimensionality of data in a linear fashion to such a degree that we can plot the reduced dimension dataset in two or three dimensions. Canonical Variates (see elsewhere) also performs linear dimension reduction exploiting class information – if we reduce the dimension to only two or three, we can also visualise the data. Non-linear dimension reduction, such as autoencoders can also be used in the same way for visualisation. In autoencoders, we constructed the error function to be the squared error loss between the input and the output. However, there was no explicit requirement that the low dimensional representation of the data should, in some sense, be a good visualisation of the high dimensional data. This issue is addressed here by considering methods that try to preserve (at least locally) the topology of the high dimensional data.

Multidimensional Scaling

In multidimensional scaling (MDS) we are given distances d_{rs} between every pair of observations (that is, we may not have direct access to any high-dimensional data, but we do have access to a measure of the "distances" between every two points). The idea is to try to reconstruct what the original data was, based solely on these distances. For example, given only the distances between towns, can we construct a map for the coordinates of the towns themselves? A practical area for such methods is in the visualisation of proteins and other macromolecules based on measures of similarity between the molecules.

9.1 Classical Scaling

Consider a set of datapoints $\{x^\mu, \mu = 1, \ldots, P\}$, where the dimension of each datapoint, $dim(x^\mu) = n$. From this data we can form the distances between all the datapoints:

$$T^x_{ab} = (x^a - x^b)^2 \tag{9.1.1}$$

to form a $P \times P$ distance matrix T^x between the x datapoints. The idea in Classical Scaling is to find a set of vectors $y^\mu, \mu = 1, \ldots, P$ such that the distance matrix T^y between the y points matches as closely as possible the distance matrix T^x. The dimension of the datapoints, $dim(y)$ is typically chosen to be small, either two or three so that we can visualise the data.

In other words : Given a distance matrix T only, how can we find a set of points y^μ that has this distance matrix?

The interesting thing about Classical Scaling is that the solution to this problem, for the case of using Euclidean squared distance, is analytic, as described below.

9.1.1 Finding the optimal points

Here we briefly describe the mathematics behind the solution of the Classical Scaling. If we consider a single element of the distance matrix, we have

$$T_{ab} = x^a \cdot x^a - 2x^a \cdot x^b + x^b \cdot x^b \tag{9.1.2}$$

For convenience, let us define a matrix

$$X_{ij} = x^i_j \tag{9.1.3}$$

Furthermore, define the matrix

$$E = XX^{\mathsf{T}} \tag{9.1.4}$$

Then we can express one element of T as

$$T_{ab} = E_{aa} - 2E_{ab} + E_{bb} \tag{9.1.5}$$

If it were not for the terms E_{aa} and E_{bb}, life would be easy since, in that case, we would have a known matrix, T expressed as the outer product of an unknown matrix, X which would be easy to solve. What we need to do therefore is to express the unknown matrix elements E_{aa} and E_{bb} in terms of the known matrix T. In order to do this, we make the following extra assumption – the data has zero mean, $\sum_a x^a_i = 0$. Clearly, this does not affect the solution since it is only defined up to an arbitrary shift. In that case, $\sum_a E_{ab} = \sum_{ai} x^a_i x^b_i = 0$. Hence,

$$\sum_a T_{ab} = \sum_a E_{aa} - 2\sum_a E_{ab} + PE_{bb} \tag{9.1.6}$$

$$= \sum_a E_{aa} + PE_{bb} \tag{9.1.7}$$

This means that we could express E_{bb} in terms of T, if only we knew what $\sum_a E_{aa}$ is. But this can also be obtained by now summing over b:

$$\sum_{ab} T_{ab} = P\sum_a E_{aa} + P\sum_b E_{bb} \tag{9.1.8}$$

$$= 2P\sum_a E_{aa} \tag{9.1.9}$$

This means

$$PE_{bb} = \sum_a T_{ab} - \sum_a E_{aa} \tag{9.1.10}$$

$$= \sum_a T_{ab} - \frac{1}{2P}\sum_{ab} T_{ab} \tag{9.1.11}$$

so that

$$T_{ab} = \frac{1}{P}\sum_a T_{ab} - \frac{1}{P^2}\sum_{ab} T_{ab} + \frac{1}{P}\sum_b T_{ab} - 2E_{ab} \tag{9.1.12}$$

In other words, we can write

$$\left[XX^{\mathsf{T}}\right]_{ab} = -\frac{1}{2}\left(T_{ab} - \frac{1}{P}\sum_a T_{ab} + \frac{1}{P^2}\sum_{ab} T_{ab} - \frac{1}{P}\sum_b T_{ab}\right) \tag{9.1.13}$$

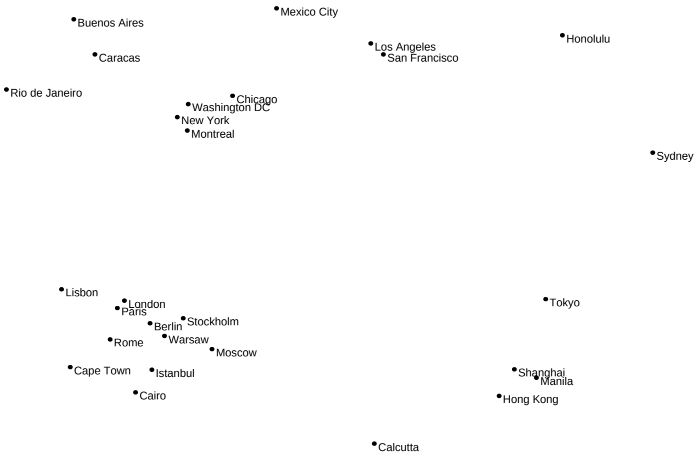


Figure 9.1: Classical scaling solution to representing 28 world cities on a two dimensional map, given only their intercity distances.

The right hand side are elements of a now known matrix, T', for which we can find an eigen-decomposition

$$T' = V\Lambda V^{\mathsf{T}} \tag{9.1.14}$$

where V is an orthogonal matrix and Λ is diagonal matrix containing the eigenvalues. Since each column of T' sums to zero, this matrix has at most rank $P-1$. A solution for the data position is given by taking the first r columns of $V\Lambda^{\frac{1}{2}}$, where r is the rank of T'. This means that if we have P vectors $x^\mu, \mu = 1, \ldots, P$ based only on the Euclidean square distances between them, we can reconstruct a set of P objects in a $P-1$ dimensional space that has exactly the same distance structure.

If we wish to look for lower dimensional approximate reconstructions (in the sense that the distance matrix in the lower dimensional space will not exactly match the given distance matrix T) , we can simply take those columns of $V\Lambda^{\frac{1}{2}}$ corresponding to the largest eigenvalues of T'.

9.1.2 The Algorithm

Given a $P \times P$ distance matrix T :

1. Calculate the $P \times P$ matrix M with elements

$$M_{ab} = -\frac{1}{2}\left(T_{ab} - \frac{1}{P}\sum_a T_{ab} + \frac{1}{P^2}\sum_{ab} T_{ab} - \frac{1}{P}\sum_b T_{ab}\right) \tag{9.1.15}$$

2. Calculate the m largest eigenvalues $\lambda^i, i = 1, \ldots, m$ of M, and their corresponding eigenvectors e^i.

3. The points $y^j, j = 1, \ldots P$ in the m dimensional space are then gives by the positions $y^j_i = \sqrt{\lambda^i} e^i_j$.

Example : intercity

We are given the intercity distances of 28 major cities in the world. This is therefore

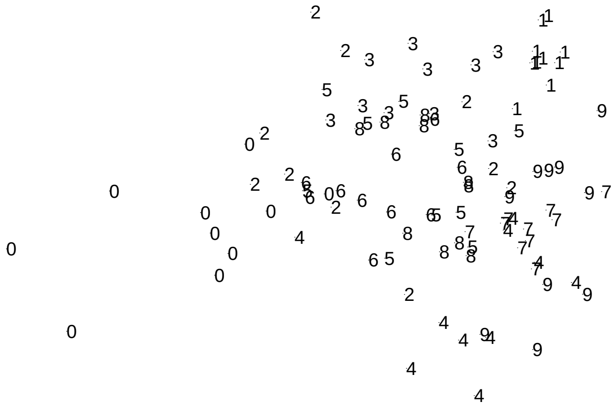


Figure 9.2: Classical scaling solution to representing digits in two dimensions. Note how some digits are more closely clustered together than others.

a 28 × 28 dimensional matrix T. The above algorithm is coded below in Matlab to form a 3 dimensional representation of the cities.

The result is given in fig(9.1) where we have plotted only two of the three dimensions. Note how the representation is roughly correct from our experience of where cities are in the world.

Example : Digit data

We can also use classical scaling to reduce the dimension. I took 10 examples for each of the ten classes of digit – 100 datapoints therefore in total. Each digit is represented as a 784 dimensional vector. I then formed the 100 × 100 dimensional distance matrix T, and used classical scaling to plot the resulting 3 dimensional reconstructions. The results are given in fig(9.2).

9.2 Sammon Mapping

The Sammon mapping is a technique more general (and more widely used) than classical scaling. The idea is very simple. Given a $P \times P$ dissimilarity matrix d_{ij}, and a function $d(y^i, y^j)$ that measures the dissimilarlity of two vectors y^i and y^j, we look to place objects in a space such that their dissimilarities are close to the given dissimilarities d_{ij}. An objective function to achieve this is

$$E = \frac{1}{\sum_{ij} d_{ij}} \sum_{i<j} \frac{\left(d_{ij} - d(y^i, y^j)\right)^2}{d_{ij}} \tag{9.2.1}$$

We minimise E with respect to the positions $y^i, i = 1, \dots p$. The division by d_{ij} is included in order to encourage the solution to deal with small dissimilarities accurately. (We do not divide by d_{ij}^2 since then small and large dissimilarities would be treated roughly equally). In order to train such a model, standard (nonlinear) optimisation techniques can be employed.

For example, we could define dissimilarities as

$$d(y^i, y^j) = \left(y^i - y^j\right)^4. \tag{9.2.2}$$

Then, given a set of target dissimilarities d_{ij} we then need to arrange the vectors y^i to minimize the (weighted) difference between the given dissimilarities and those measured above. The parameters of the optimization are therefore the vectors y^i themselves.

Strictly speaking, the Sammon "Mapping" is not a mapping, since it does not yield a function that describes how general points in one space are mapped to another (it only describes how a limited set of points is related).

Making a Mapping

Given points x^i in a n-dimensional space (possibly very high dimensional) to represent them by points y^i in a m-dimensional space (possibly very low dimensional, say 2) in such a way that the separation between the points in the two spaces is roughly similar. One way to obtain this mapping is to parameterize the positions of the objects in the lower dimensional space

$$y = f(x; W) \tag{9.2.3}$$

The distance then between two mapped points is a function of the parameters of the mapping W. The optimal parameters can then be found by optimization. The method Neuroscale is one such procedure.

9.3 A word of warning

It can be that very high dimensional datasets appear to lie on ring when plotted using visualisation methods. It may well be that the data really does have this kind of structure. However, in high dimensions, the distance matrix (between every pair of points) will be dominated by those points that are furthest apart. This will give the impression that most points are a long way from each other, and a ring or circular type two dimensional representation will likely be the visualisation solution. One should therefore bear in mind that global topological constraints on the data are unlikely to be accurately represented by these visualisation procedures, and one should be wary of reading too much into the precise structure of the visualisation.

II. Inference and Learning in Probabilistic Models

10 Basic Concepts in Probability

10.1 What does random mean?

Arguably, it's ultimately a question of physics as to whether or not the world is inherently random. That is, are phenomena that appear random just examples of highly complex processes, or does, indeed, 'God play dice'[3]? The concepts of randomness and uncertainty are intrinsically linked, and it is our desire to deal with uncertain processes that motivates us here.

A viewpoint that we will often use here is that of a generating process. This viewpoint stems from the idea that we only have a limited ability to observe the world, and that we may not be able to observe all associated conditions relevant to the generating process. For example, Sally tosses a coin – this is the generating process. We can certainly observe the outcome of this process – whether the coin is heads or tails – these states are observable or *visible*. However, we may not be able to observe all the relevant conditions which may have been involved in the generating process, such as the state of Sally's finger, whether or not she subconsciously has a preference for holding the coin in her hand a certain way – the states of such conditions as assumed *hidden*.

One pragmatic definition of randomness would be that, given the current (limited) information about the process which generates the data, the outcome of the experiment (generation) cannot be predicted which certainty. However, we would like to be able to make statements about the outcomes of a generating process, even though it may be predicted without certainty. That is, we wish to form a calculus for uncertainty.

10.2 What is Probability?

Probability is a contentious topic, stretching over several hundred years. It is not necessarily the axioms of probability that are contentious, rather what, if any, physical reality do they correspond to? A summary of the history of these ideas and interpretations is given in [4]. Personally, I very much like the phrase[5]

Probability theory is nothing but common sense reduced to computation (Laplace).

One approach to thinking of probability is that it enables us to make compact descriptions of phenomena.

Science is about finding compact descriptions of phenomena, and the concepts of probability/randomness help us to achieve this.

For example, Sally has a coin, and does some experiments to investigate its properties. She tosses the coin, and find the outcomes

$$H, T, H, H, T, T, H, H, H, T, H, T, H$$

Her professor asks her to make an analysis of the results of her experiments. Initially, Sally is tempted to say that she does not wish to summarise the experiments, since this would constitute a loss of information. However, she realises that there is little to be gained from simply reporting the outcomes of the experiment, without any summarisation of the results.

Independence → compact description

She therefore make an assumption (which she states in her report), namely that the outcome of tossing a coin at one time, did not influence the outcome at any other time (so, if the coin came up heads now, this will not influence whether or not the coin comes up heads or tails the next throw). This is a common assumption and called *independence* of trials. Under this assumption, the ordering of the data has no relevance, and the only quantity which is therefore invariant under this assumption is the total number of heads, and the total number of tails observed in the experiment. See how this assumption has enabled us to make a compact description of the data. Indeed, it is such *independence* assumptions, their characterisation and exploitation that is the subject of graphical models.

Random → compact *model* description

Sally repeats the experiment many times, and notices that the ratio of the number of heads observed to the total number of throws tends to roughly 0.5. She therefore summarises the results of her experiments by saying that, in the long run, on average, she *believes* half the time the coin will end up heads, and half the time tails. Key words in the above sentence are *believes* and *long run.* We say *believes* since Sally cannot repeat the experiment an *infinite* number of times, and it is therefore her *belief* that if she were to toss the coin an infinite number of times, the number of heads occurring would be half that of the total. Hence, she invokes the concept of randomness/probability to describe a *model* that she believes accurately reflects the kind of experimental results she has found.

In a sense, Sally (and her environment) operates like a random number generator. (If we knew all the possible things that could influence Sally, and the coin, and how she might toss it, we might conclude that we could predict the outcome of the coin toss – there is nothing 'random' about it. However, to do so would not render itself to a compact description – hence the usefulness of the concept of randomness).

10.3 Rules of Probability

Events and the Rules

Events are possible outcomes of a generating process. For example 'the coin is heads' is and event, as is 'the coin is tails'. In this case, these two events are mutually exclusive, since they cannot both simultaneously occur.

- We'll take the pragmatic viewpoint that the probability of an event occurring is simply a number $p(\text{event occurs})$ between 0 and 1.
- $p(\text{event occurs}) = 0$ means that it is certain that the event cannot occur. Similarly, $p(\text{event occurs}) = 1$ means that it is certain that the event occurs.
- We need a rule for how events interact :

$$p(x \text{ or } y) = p(x) + p(y) - p(x \text{ and } y)$$

I'll use $p(x, y)$ to denote the joint event $p(x \text{ and } y)$.

Conditional Probability / Bayes Rule

A useful definition is that of conditional probability. The probability of event x conditional on knowing that event y has occurred (or more shortly, the probability of x given y) is defined as

$$p(x|y) \equiv \frac{p(x,y)}{p(y)}$$

Here's one way to think about conditional probability : Imagine we have a dart board, split into 20 sections. We may think of a drunk dart thrower, and describe our lack of knowledge about the throwing by saying that the probability that a dart occurs in any one of the 20 regions is $p(\text{region } i) = 1/20$. Imagine then, that our friend, Randy, the random dart thrower is blindfolded, and throws a dart. With pint in hand, a friend of Randy tells him that he hasn't hit the 20 region. What is the probability that Randy has hit the 5 region?

Well, if Randy hasn't hit the 20 region, then only the regions 1 to 19 remain and, since there is no preference for Randy for any of these regions, then the probability is 1/19. To see how we would calculate this with the rules of probability :

$$\begin{aligned} p(\text{region 5}|\text{ not region 20}) &= \frac{p(\text{region 5, not region 20})}{p(\text{ not region 20})} \\ &= \frac{p(\text{region 5})}{p(\text{not region 20})} = \frac{1/20}{19/20} = \frac{1}{19} \end{aligned}$$

giving the intuitive result.

Degree of Belief

The above dart board example is easy to think about in terms of probability – it's straightforward to imagine many repetitions of the experiment, and one could think about the 'long' run way of defining probability.

Here's another problem :

What's the probability that I will like the present my grandmother has bought me for Christmas? In a purely technical sense, if we were to define probability as a limiting case of infinite repetitions of the same experiment, this wouldn't make much sense in this case. We can't repeat the experiment. However, simply the predictability or degree of belief interpretation sidesteps this issue – it's just a consistent framework for manipulating real values consistent with our intuition about probability.

TODO: Confidence intervals, Central limit theorem, Asymptotic Equipartition Theorem.

10.4 Problems

Exercise 23 *The Gaussian distribution in one dimension is defined as*

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

and satisfies $\int_{-\infty}^{\infty} p(x)dx = 1$.

Show that

1. $\int_{-\infty}^{\infty} xp(x)dx = \mu$.
2. $\int_{-\infty}^{\infty} (x-\mu)^2 p(x)dx = \sigma^2$.

Exercise 24 *Consider data* $x^i, i = 1, \ldots, P$. *Show that the Maximum Likelihood estimator of* μ *is* $\hat{\mu} = \frac{1}{P}\sum_{i=1}^{P} x^i$ *and that the ML estimate of* σ^2 *is* $\hat{\sigma^2} = \frac{1}{P}\sum_{i=1}^{P}(x^i - \mu)^2$.

Exercise 25 *A training set consists of one dimensional examples from two classes. The training examples from class 1 are* $\{0.5, 0.1, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.35, 0.25\}$ *and from class 2 are* $\{0.9, 0.8, 0.75, 1.0\}$. *Fit a (one dimensional) Gaussian using Maximum Likelihood to each of these two classes. Also estimate the class probabilities* p_1 *and* p_2 *using Maximum Likelihood. What is the probability that the test point* $x = 0.6$ *belongs to class 1?*

Exercise 26 *Given the distributions* $p(x|class1) = N(\mu_1, \sigma_1^2)$ *and* $p(x|class2) = N(\mu_2, \sigma_2^2)$, *with corresponding prior occurrence of classes* p_1 *and* p_2 $(p_1 + p_2 = 1)$, *calculate the decision boundary explicitly as a function of* $\mu_1, \mu_2, \sigma_1^2, \sigma_2^2, p_1, p_2$. *How many solutions are there to the decision boundary, and are they all reasonable?*

10.5 Solutions

11 Introducing Graphical Models

11.1 Belief Networks

Belief Networks (also called Bayes' Networks or Bayesian Belief Networks) are the most straightforward kind of graphical model to introduce[10, 11]. Their use is widespread and they have tremendous applicability, ranging from troubleshooting facilities in Microsoft operating systems, expert reasoning under uncertainty to machine learning in general. We will here consider a simple example of a discrete variable BN.

11.1.1 Tracey

Tracey lives in sunny Birmingham. One morning she leaves her house and realizes that her grass is wet. Is it due to rain or has she forgotten to turn off the sprinkler ?

Next she notices that the grass of her neighbour, Jack, is also wet. She concludes therefore that it has probably been raining, and that "explains away" to some extent the possibility that her sprinkler was left on.

Making a model

We can model the above situation using probability by following a general modelling approach.

First we define what variables we wish to include in our model. In the above situation, the natural variables are

$R \in \{0, 1\}$ ($R = 1$ means that it has been raining, and 0 otherwise).

$S \in \{0, 1\}$ ($S = 1$ means that it she has forgotten to turn off the sprinkler, and 0 otherwise).

$J \in \{0, 1\}$ ($J = 1$ means that Jack's grass is wet, and 0 otherwise).

$T \in \{0, 1\}$ ($T = 1$ means that Tracey's Grass is wet, and 0 otherwise).

A model of Tracey's world then corresponds to a probability distribution on the joint set of the variables of interest $p(T, J, R, S)$ (note that the order of the variables is irrelevant).

Since each of the variables in this example can take one of two states, it would appear that we naively have to specify the values for each of the $2^4 = 16$ states, e.g. $p(T = 1, J = 0, R = 1, S = 1) = 0.7$ etc. However, since there are normalisation conditions for probabilities, surely, we do not need to specify all the state probabilities, since some will be determined by normalisation.

To see how many states need to be specified in general, consider the following decomposition. Without loss of generality (WLOG) and repeatedly using Bayes' rule, we may write[1]:

$$\begin{aligned} p(T, J, R, S) &= p(T|J, R, S)p(J, R, S) = p(T|J, R, S)p(J|R, S)p(R, S) \\ &= p(T|J, R, S)p(J|R, S)p(R|S)p(S) \end{aligned}$$

That is, we may write the joint distribution as a product of conditional distributions. The first term $p(T|J, R, S)$ requires us to specify $2^3 = 8$ values, say for $p(T = 1|J, R, S)$ given the 8 possible states jointly of J, R, S. The other value $p(T = 0|J, R, S)$ is given by normalisation : $p(T = 0|J, R, S) = 1-p(T = 1|J, R, S)$. Similarly, we need $4+2+1$ values for the other factors, making a total of 15 values in all. In general, for a set of n binary variables, we need to specify $2^n - 1$ values in the range $[0, 1]$. The important point here is that the number of values that need to be specified in general scales exponentially with the number of variables in the model – this is extremely bad news, and motivates simplifications.

Conditional Independence

The modeler often knows that certain simplifications often occur. Indeed, it is arguably the central role of modelling to make the simplest model that fits with the modelers beliefs about an environment. For example, in the scenario above, Tracey's grass is wet dependent only directly on whether or not is has been raining and or whether or not her sprinkler was on. That is, we make the conditional independence assumption for this model that

$$p(T|J, R, S) = p(T|R, S)$$

Similarly, since whether or not Jack's grass is wet is influenced only directly by whether or not is has been raining, we write

$$p(J|R, S) = p(J|R)$$

and since the rain is not directly influenced by the sprinkler!

$$p(R|S) = p(R)$$

which means that our model now becomes :

$$p(T, J, R, S) = p(T|R, S)p(J|R)p(R)p(S)$$

We can represent these conditional independencies graphically, as in fig(11.1).

This reduces the number of values that we need to specify to $4+2+1+1 = 8$, a big saving over the previous 15 values in the case where no conditional independencies had been assumed.

The heart of modelling is in judging which variables are dependent on each other.

Specifying the values

To complete the model, we need to numerically specify the values of the conditional

[1] Note that a probability distribution simply assigns a value between 0 and 1 for each of the states jointly of the variables. For this reason, $p(T, J, R, S)$ is considered equivalent to $p(J, S, R, T)$ (or any such reordering of the variables), since in each case the joint setting of the variables is simply an index to the same probability. This situation is more clear in the set theoretic notation $p(J \cap S \cap T \cap R)$. We abbreviate this set theoretic notation by using the commas – however, one should be careful not to confuse the use of this indexing type notation with functions $f(x, y)$ which are in general dependent on the variable order. Whilst the variables to the left of the conditioning bar may be written in any order, and equally those to the right of the conditioning bar may be written in any order, moving variables across the bar is not allowed, so that $p(x_1|x_2) \neq p(x_2|x_1)$.

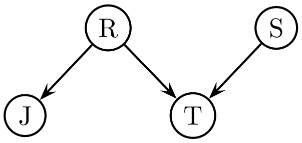


Figure 11.1: Belief network structure for the "wet grass" example. Each node in the graph represents a variable in the joint distribution, and the variables which feed in (the parents) to another variable represent which variables are to the right of the conditioning bar.

probabilty tables (CPTs). Let the prior probabilities for R and S be $p(R) = (0.2, 0.8)$ (that is, $p(rain = yes) = 0.2$ and $p(rain = no) = 0.8$) and $p(S) = (0.1, 0.9)$. Note, for clarity I use here for example $p(R = y)$ instead of $p(R = 1)$ – of course, the labels we use for the states are irrelevant. Let's set the remaining probabilities to

$p(J = y|R = y) = 1$, $p(J = y|R = n) = 0.2$ (sometimes Jack leaves his own sprinkler on too).

$p(T = y|R = y, S) = 1$,$p(T = y|R = n, S = y) = 0.9$ (there's a small chance that even though the sprinkler was left on, it didn't wet the grass noticeably), $p(T = y|R = n, S = n) = 0$

The prior belief that the sprinkler is responsible is $p(S = y) = 0.1$.

Inference

Now that we've made a model of an environment, we can perform inference. Let's calculate the probability that the sprinkler was on overnight, given that Tracey's grass is wet: $p(S = y|T = y)$.

To do this, we use Bayes rule:

$$\begin{aligned}
p(S = y|T = y) &= \frac{p(S = y, T = y)}{p(T = y)} \\
&= \frac{\sum_{J,R} p(T = y, J, R, S = y)}{\sum_{J,R,S} p(T = y, J, R, S)} \\
&= \frac{\sum_{J,R} p(J|R)p(T = y|R, S = y)p(R)p(S = y)}{\sum_{J,R,S} p(J|R)p(T = y|R, S)p(R)p(S)} \\
&= \frac{\sum_{R} p(T = y|R, S = y)p(R)p(S = y)}{\sum_{R,S} p(T = y|R, S)p(R)p(S)} \\
&= \frac{0.9 * 0.8 * 0.1 + 1 * 0.2 * 0.1}{0.9 * 0.8 * 0.1 + 1 * 0.2 * 0.1 + 0 * 0.8 * 0.9 + 1 * 0.2 * 0.9} \\
&= \frac{0.092}{0.272} = 0.3382
\end{aligned}$$

so that the belief that the sprinkler is on *increases* above the prior probability 0.1, due to the fact that the grass is wet.

Let us now calculate the probability that Tracey's sprinkler was on overnight, given that her grass is wet and that Jack's grass is also wet, $p(S = y|T = y, J = y)$. We use Bayes rule again:

$$\begin{aligned} p(S = y|T = y, J = y) &= \frac{p(S = y, T = y, J = y)}{p(T = y, J = y)} \\ &= \frac{\sum_R p(T = y, J = y, R, S = y)}{\sum_{R,S} p(T = y, J = y, R, S)} \\ &= \frac{\sum_R p(J = y|R)p(T = y|R, S = y)p(R)p(S = y)}{\sum_{R,S} p(J = y|R)p(T = y|R, S)p(R)p(S)} \end{aligned}$$

substituting in the numbers, as before, we get

$$= \frac{0.0344}{0.2144} = 0.1604$$

What this shows is that the probability that the sprinkler is on, given the extra evidence that Jack's grass it wet, is *lower* than the probability that the grass is wet given only that Tracey's grass is wet. That is, that the grass is wet due to the sprinkler is (partly) explained away by the fact that Jack's grass is also wet – this increases the chance that the rain has played a factor in making Tracey's grass wet.

11.2 A word on notation

Graphs are widely used, but differ markedly in what they represent. Here we try to cover two common misconceptions.

State Transition Diagrams Such graphical representations are common in Markov Chains and Finite State Automata. A set of states is written as set of nodes(vertices) of a graph, and a directed edge between node i and node j (with an associated weight p_{ij}) represents that a transition from state i to state j can occur with probability p_{ij}. From the graphical models perspective we would simply write down a directed graph $x(t) \to x(t+1)$ to represent this Markov Chain. The state-transition diagram simply provides a graphical description of the conditional probability table $p(x(t+1)|x(t))$.

Neural Networks Neural networks also have vertices and edges. In general, however, neural networks are graphical representations of *functions*, whereas as graphical models are representations of distributions (a much more powerful generalisation). Neural networks (or any other parametric description) may be used to represent the conditional probability tables, as in sigmoid belief networks[12].

11.3 Example : Was it the Burglar?

Here's another example using binary variables. Sally comes home to find that the burglar alarm is sounding ($A = 1$). Has she been burgled ($B = 1$), or was the alarm triggered by an earthquake ($E = 1$)? She turns the car radio on for news of earthquakes ($R = 1$).

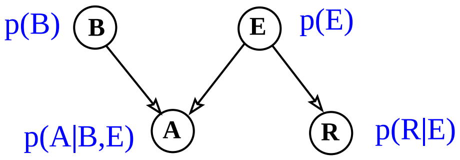


Figure 11.2: Belief Network for the Burglar model. Here, for pedagogical purposes only, we have explicitly written down which terms in the distribution each node in the graph represents.

Using Bayes' rule, we can write, without loss of generality,

$$p(B, E, A, R) = p(A|B, E, R)p(B, E, R)$$

We can repeat this for $p(B, E, R)$, and continue

$$p(B, E, A, R) = p(A|B, E, R)p(R|B, E)p(E|B)p(B)$$

However, the alarm is surely not directly influenced by any report on the Radio – that is, $p(A|B, E, R) = p(A|B, E)$. Similarly, we can make other conditional independence assumptions such that

$$p(B, E, A, R) = p(A|B, E)p(R|E)p(E)p(B)$$

Specifying Conditional Probability Tables

Each node has an associated conditional probability distribution:

Alarm = 1	Burglar	Earthquake
0.9999	1	1
0.99	1	0
0.99	0	1
0.0001	0	0

Radio = 1	Earthquake
1	1
0	0

The remaining tables are $p(B = 1) = 0.01$ and $p(E = 1) = 0.000001$. The tables and graphical structure fully specify the distribution.

Explaining Away

Now consider what happens as we observe evidence.

Initial Evidence: The Alarm is sounding:

$$p(B = 1|A = 1) = \frac{\sum_{E,R} p(B=1,E,A=1,R)}{\sum_{B,E,R} p(B,E,A=1,R)}$$

$$= \frac{\sum_{E,R} p(A=1|B=1,E)p(B=1)p(E)p(R|E)}{\sum_{B,E,R} p(A=1|B,E)p(B)p(E)p(R|E)} \approx 0.99$$

Additional Evidence: The Radio broadcasts an Earthquake warning:

A similar calculation gives $p(B = 1|A = 1, R = 1) \approx 0.01$

Thus, initially, because the Alarm sounds, Sally thinks that she's been burgled. However, this probability drops dramatically when she hears that there has been an Earthquake. That is, the Earthquake "explains away" to a large extent the fact that the Alarm is ringing.

General Calculations

Of course, we don't wish to carry out such inference calculations by hand all the time. General purpose algorithms exist for this, such as the Junction Tree Algorithm, and we shall introduce these later.

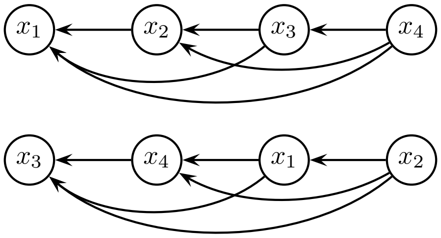


Figure 11.3: Two Belief networks for a 4 variable distribution. In this case, both graphs are representations of the *same* distribution $p(x_1, x_2, x_3, x_4)$. The extension of this 'cascade' to many variables is obvious, and always results in an acyclic graph.

11.4 Belief Networks

The reader may have been aware that in the above two examples, we had a choice as to how we used Bayes' rule. For example, in a general 4 variable case, we could choose the factorisation,

$$p(x_1, x_2, x_3, x_4) = p(x_1|x_2, x_3, x_4)p(x_2|x_3, x_4)p(x_3|x_4)p(x_4)$$

equally valid is (see fig(11.3))

$$p(x_1, x_2, x_3, x_4) = p(x_3|x_4, x_1, x_2)p(x_4|x_1, x_2)p(x_1|x_2)p(x_2).$$

Of course, if one wishes to make independence assumptions, then the initial choice becomes significant. However, one should bear in mind that, in general, two different graphs may represent the same distribution.

Indeed, the observation that any distribution may be written in the cascade form fig(11.3) gives an algorithm for constructing a belief network on variables $x_1, \ldots, x_n$: write down the $n-$variable cascade graph; assign any ordering of the variables to the nodes; you may then delete any of the directed connections.

Variable Order

To ensure maximum sparsity, add "root causes" first, then the variables they influence, and so on, until the leaves are reached. Leaves have no direct causal[2] influence over the other variables.

Conditional Probability Tables (CPTs)

Once the graphical structure is defined, the actual values of the tables $p(x_i|\text{pa}\,(x_i))$ need to be defined. That is, for every possible state of the parental variables pa (x_i), a value for each of the states (except one, since this is determined by normalisation) needs to be specified. For a large number of parents, writing out a table of values is intractable, and the tables are usually parameterised in some simple way. More on this later.

[2] 'Causal' is a tricky word since here there is no temporal 'before' and 'after', merely correlations or dependencies. For a distribution $p(a, b)$, we could write this as either $p(a|b)p(b)$ or $p(b|a)p(a)$. In the first, we might think that b 'causes' a, and in the second case, a 'causes' b. Clearly, this is not very meaningful since they both represent exactly the same distribution, and any apparent causation is merely spurious. Nevertheless, in constructing belief networks, it can be helpful to think about dependencies in terms of causation since our intuitive understanding is that often one variable 'influences' another. This is discussed much more deeply in [13], where a true calculus of causality is developed.

11.4.1 Conditional Independence

Consider the three variable distribution $p(x_1, x_2, x_3)$. We may write this is any of the 6 ways $p(x_{i_1}|x_{i_2}, x_{i_3})p(x_{i_2}|x_{i_3})p(x_{i_3})$ where (i_1, i_2, i_3) is any of the 6 permutations of $(1, 2, 3)$. Hence, whilst all graphically different, they all represent the same distribution which does not make any conditional independence statements. To make an independence statement, we need to drop one of the links. This gives rise in general to 4 graphs in fig(11.4). Are any of these graphs equivalent, in the sense that they represent the same distribution? A simple application of Bayes' rule gives :

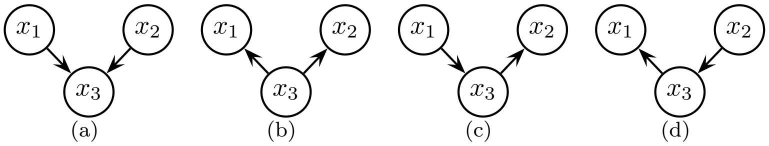


Figure 11.4: By dropping say the connection between variables x_1 and x_2, we reduce the 6 possible graphs amongst three variables to 4.

$$\underbrace{p(x_2|x_3)p(x_3|x_1)p(x_1)}_{graph(c)} = p(x_2, x_3)p(x_3, x_1)/p(x_3) = p(x_1|x_3)p(x_2, x_3)$$

$$= \underbrace{p(x_1|x_3)p(x_3|x_2)p(x_2)}_{graph(d)} = \underbrace{p(x_1|x_3)p(x_2|x_3)p(x_3)}_{graph(b)}$$

and hence graphs (b),(c) and (d) represent the same distribution. However, graph (a) represents something fundamentally different: there is no way to transform the distribution $p(x_3|x_1, x_2)p(x_1)p(x_2)$ into any of the others.

Graphs (b),(c) and (d) all represent the same conditional independence assumption that, given the state of variable x_3, variables x_1 and x_2 are independent. We write this as $I(x_1, x_2|x_3)$.

Graph (a) represents something different, namely marginal independence : $p(x_1, x_2) = p(x_1)p(x_2)$. Here we have marginalised over the variable x_3.

11.4.2 Intuition

In a general Belief Network, with many nodes, how could we check if two variables x and y are independent, once conditioned on another variable z? In fig(11.5)(a,b), it is clear that x and y are independent when conditioned on z. It is clear in fig(11.5)(c) that they are dependent. In this situation, variable z is called a collider – the arrows of its neighbours are pointing towards it. What about fig(11.5)(d)? In (d), when we condition on z, then, in general, x and y will be dependent, since

collider

$$\sum_w p(z|w)p(w|x, y)p(x)p(y) \neq p(x|z)p(y|z)$$

– intuitively, variable w becomes dependent on the value of z, and since x and y are conditionally dependent on w, they are also conditionally dependent on z.

Roughly speaking, if there is a non-collider z which is conditioned on along the path between x and y (as in fig(11.5)(a,b)), then this path does not make x and y

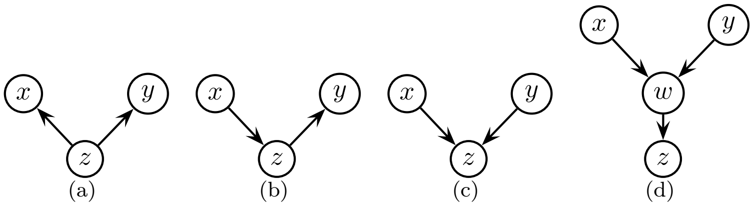


Figure 11.5: In graphs (a) and (b), variable z is not a collider. (c) Variable z is a collider. Graphs (a) and (b) represent conditional independence $I(x, y|z)$. In graphs (c) and (d), x and y are conditionally dependent given variable z.

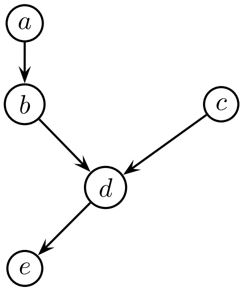


Figure 11.6: The variable d is a collider along the path $a - b - d - c$, but not along the path $a - b - d - e$.

dependent. Similarly, if there is a path between x and y which contains a collider, provided that this collider is not in the conditioning set (and neither are any of its children) then this path does not make x and y dependent.

Note that a collider is defined *relative to a path*. In fig(11.6), the variable d is a collider along the path $a - b - d - c$, but not along the path $a - b - d - e$ (since, relative to this path, the two arrows to not point inwards to d).

These intuitions lead us to a more formal statement about conditional independence:

11.4.3 d-Separation

If two variables are d-separated relative to a set of variables Z in a directed graph, then they are independent conditional on Z in all probability distributions such a graph can represent. Roughly, two variables X and Y are independent conditional on Z if knowledge about X gives you no extra information about Y once you have knowledge of Z. In other words, once you know Z, X adds nothing to what you know about Y.

Formally, to define d-separation, we need to define d-connection:

d-connection If G is a directed graph in which X, Y and Z are disjoint sets of vertices, then X and Y are d-connected by Z in G if and only if there exists an undirected path U between some vertex in X and some vertex in Y such that for every collider C on U, either C or a descendent of C is in Z, and no non-collider on U is in Z.

X and Y are d-separated by Z in G if and only if they are not d-connected by Z in G.

See `http://www.andrew.cmu.edu/user/scheines/tutor/d-sep.html` from where this definition was taken for more details, and also some nice demos.

Bayes Ball The Bayes Ball algorithm, "Bayes-Ball: The Rational Pastime" R.D. Shachter (UAI 98) provides a linear time complexity algorithm which given a set of nodes X and Z determines the set of nodes Y such that $I(X, Y|Z)$. Y is called the set of irrelevant nodes for X given Z

Example (1) Consider the simple graphs in fig(11.7).

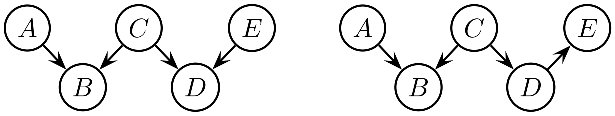


Figure 11.7: Examples for d-separation – Is $I(a, e|b)$? Left: If we sum out variable d, then we see that a and e are *independent* given b, since the variable e will appear as an isolated factor independent of all other variables, hence indeed $I(a, e|b)$. Whilst b is a collider which is in the conditioning set, we need *all* colliders on the path to be in the conditioning set (or their descendents) for d-connectedness. Right: Here, if we sum out variable d, then variables c and e becomes intrinsically linked, and the distribution $p(a, b, c, e)$ will not factorise into a function of a multiplied by a function of e – hence they are dependent.

Example (2) Consider the simple graph in fig(11.8).

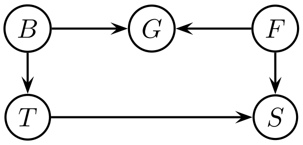


Figure 11.8: Example for d-separation.

Are the variables T and F unconditionally independent, i.e. $I(T, F|\emptyset)$? Remember that the key point are the colliders along the path between the two variables. Here there are two colliders, namely G and S – however, these are not in the conditioning set (which is empty), and hence they are d-separated, and unconditionally independent.

What about $I(T, F|G)$? Well, now there is a collider on the path between T and F which is in the conditioning set. Hence T and F are d-connected conditioned on G, and therefore T and F are not independent conditioned on G.

Note that this may seem a bit strange – initially, when there was no conditioning, T and F were independent. However, conditioning on G makes them dependent. An even simpler example would be the graph $A \to B \leftarrow C$. Here A and B are unconditionally independent. However, conditioning of B makes them dependent. Intuitively, whilst we believe the root causes are independent, given the value of the observation, this tells us something about the state of *both* the causes, coupling them and making them dependent.

What about $I(B, F|S)$? Since there is a collider on the path between T and F which is in the conditioning set, namely S, B and F are conditionally dependent given S.

Deterministic Dependencies

Sometimes the concept of independence is perhaps a little strange. Consider the following : variable x and y are both binary, being either in state 0 or 1. We define the distribution such that x and y are always in opposite states : $p(x = 0|y = 0) = 0$, $p(x = 0|y = 1) = 1$, $p(x = 1|y = 0) = 1$, $p(x = 1|y = 1) = 0$. To complete the specification of $p(x, y) = p(x|y)p(y)$, we define $p(y = 0) = p(y = 1) = 0.5$. Hence, x and y are in opposite states and they are each in state 0 with probability 0.5. Now, are x and y dependent? Indeed, a quick calculation shows that they are. However, if we changed the prior probability, so that y is always in state 1, $p(y = 0) = 0$ and $p(y = 1) = 1$, are x and y dependent? If you carry out the maths, you'll find that they are *independent*! This may seem very strange – we know for sure that x will always be in state 0 and y will always be in state 1 – they are in opposite states, yet they are independent. This apparent strangeness results from the deterministic nature of the distribution – the distribution is trivially concentrated in a single joint state. In that case $p(x|y) = p(x)$ since x can only ever be in a single state, whatever the value of y is (indeed y can also only ever be in a single state). Perhaps the best way to think about dependence is to ask whether or not knowing the state of variable y tells you something more than you knew before about variable x, where 'knew before' means working with the joint distribution of $p(x, y)$ to figure out (without conditioning on y) what we can know about x, namely $p(x)$.

11.5 Graphical Models

We've so far considered Belief Networks (also called Bayes or Bayesian Networks) as distributions defined as directed graphs.

Graphical models are a more general marriage of graph and probability theory. Their usefulness stems from the ease of interpretation of a graphical representation of a distribution. In particular, they provide a framework for unifying a wide class of probabilistic models and associated algorithms.

There are two main types, directed and undirected, based on whether or not the underlying graph is directed or undirected. Chain graphical models contain both directed and undirected links.

Directed Graphical Models

More formally, Belief networks are directed *acyclic* graphs (DAGs), in which the nodes in the graph represent random variables in a probability distribution. To each variable A with parents $B_1 \dots B_n$, there is an associated probability table

$$p\left(A|B_1 \dots B_n\right)$$

If A has no parents then the table reduces to unconditional probabilities $p(A)$.

Chain Rule

Let BN be a Bayesian network over

$$U = \{A_1, \dots A_n\}$$

Then the joint probability distribution $p(U)$ is the product of all conditional probabilities specified by the BN:

$$p(U) = \prod_i P\left(A_i|\mathrm{pa}\left(A_i\right)\right)$$

where $\mathrm{pa}\,(A_i)$ is the parent set of A_i.

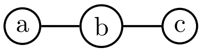
Note : it is the acyclic form which enables us to easily write down a joint distribution which is consistent with the given conditional distributions. (Contrast this with the more general case of Markov Random Fields below).

Undirected Graphical Models

It is clear that Belief Networks correspond to a special kind of factorisation of the joint probability distribution in which each of the factors is itself a distribution. However, we may consider other distributions, for example

$$p(a, b, c) = \frac{1}{Z}\phi(a, b)\phi(b, c)$$

where $\phi(a, b)$ and $\phi(b, c)$ are simply two (non-negative) *potential* functions, and Z is a constant which ensures normalisation of the distribution. In the above case, since we could equally write, for example $\tilde{\phi}(b, a) \equiv \phi(a, b)$ (that is, the variables simply index the value of the function, and are otherwise interchangeable) we can represent the distribution as an undirected graph: In this sense, a potential is associated with each *link* on the graph.

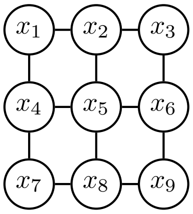


It is clear that directed distributions can be represented as undirected distributions since one can associate each (normalised) factor in a directed distribution with a potential. For example, the distribution $p(a|b)p(b|c)p(c)$ can be factored as $\phi(a, b)\phi(b, c)$, where $\phi(a, b) = p(a|b)$ and $\phi(b, c) = p(b|c)p(c)$, with $Z = 1$.

Undirected models will turn out to be extremely useful as part of general algorithms for computations with graphs. They are also useful models in themselves, and have a long history in different branches of science, especially statistical mechanics on lattices and as models in visual processing.

An undirected graph

Consider a model in which our desire is that states of the binary valued variables $x_1, \ldots, x_9$, arranged on a lattice (as below) should prefer their neighbouring variables to be in the same state



$$p(x_1, \ldots x_9) = \frac{1}{Z}\prod_{<ij>}\phi_{ij}(x_i, x_j)$$

where $< ij >$ denotes the set of indices where i and j are neighbours in the undirected graph. Then a set of potentials that would encourage neighbouring variables to have the same state would be

$$\phi_{ij}(x_i, x_j) = e^{-\frac{1}{T}(x_i - x_j)^2}$$

This model actually corresponds to a well-known model of the physics of magnetic systems, called the Ising Model. For high 'temperatures' T, the variables take on essentially random states, with no constraints. For low values of T, there is a strong constraint that neighbouring variables become aligned. Remarkably, one can show that, in a very large two-dimensional lattice, below the Curie temperature, T_c, the system admits a phase change in that all the variables become aligned. That this phase change happens for non-zero temperature is a fascinating result and has driven over 50 years of research in this area[14]. Essentially, global coherence spontaneously appears from weak local constraints.

Similar local constraints are popular in image restoration algorithms to clean up noise, under the assumption that noise will not show any local spatial coherence, whilst 'signal' will.

Chain Graphical Models

Chain graphical models contain both directed and undirected links.

11.5.1 Markov Random Fields

Just as a *vector field* describes a vector v which can be dependent on (say) a spatial location, $v(x)$, a MRF defines a probability distribution for each 'location', where the location is simply given by an index. That is, a MRF is defined by a set of distributions $p(x_i|pa(x_i)) > 0$ (the positivity constraint is required), where $i \in \{1, \dots, n\}$ indexes the distributions (usually, but not necessarily, on a lattice), and $pa(x_i)$ are the "parents" of variable x_i, namely that subset of the variables $x_1, \dots, x_n$ that the distribution of variable x_i depends on. The term Markov indicates that this is a proper subset of the variables. From this definition it is not obvious if there exists a distribution $p^*(x_1, \dots, x_n)$ which has conditional marginals consistent with the set of given marginals. The Hammersley-Clifford theorem specifies what the functional form of the any such joint distribution must be.

Hammersely Clifford Theorem

Imagine that we define a set of local distributions $p(x_i|pa(x_i)) > 0$. When indeed will this define a consistent joint distribution $p(x_1, \dots, x_n)$? The Hammersley Clifford Theorem states that the MRF defines a consistent joint distribution if and only if $p(x_1, \dots, x_n)$ is a so-called *Gibbs distribution*

$$p(x_1, \dots, x_n) = \frac{1}{Z} \exp \left(-\sum_c V_c(x_c) \right)$$

where the sum is over all cliques (maximal complete subgraphs), c and $V_c(x_c)$ is a real function defined over the variables in the clique c. The graph over which the cliques are defined is an undirected graph with a link between all parents $pa(x_i)$ and a link between x_i and each parent $pa(x_i)$, repeated over all the variables x_i.

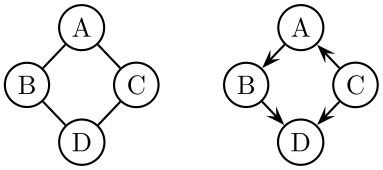


Figure 11.9: Left: An undirected model. Middle: Every DAG with the same structure as the undirected model must have a situation where two arrows will point to a node, such as node d. Summing over the states of variable in this DAG will leave a DAG on the variables A, B, C with no link between B and C – which cannot represent (in general) the undirected model since when one marginals over D in that, this adds a link between B and C.

Besag has originally a nice proof of this, which requires the positivity constraint[15], and a counter example shows that this is necessary[16].

Note : It's easy to go from the Gibbs distribution to find the local conditional distributions. The other way round is not necessarily so easy, since we would have to know the so-called partition function Z. This is reminiscent of Gibbs sampling (see the appendix on sampling) : effectively, one can easily define a sampler (based on a Gibbs distribution), but it does not mean that we know the joint distribution, ie the partition function (normalisation constant) Z.

11.5.2 Expressiveness of Graphical Models

It's clear that every Belief Network can be represented as an undirected graphical model, by simple identification of the factors in the distributions.

Can every undirected model be represented by a Belief Network *with the same link structure*? Consider the example in fig(11.9) (from Zoubin Ghahramani)

As a final note, of course, *every* probability distribution can be represented by some Belief Network, though it may not necessarily have any obvious structure and be simply a "fully connected" cascade style graph.

Discussion

Graphical models have become a popular framework for probabilistic models in artificial intelligence and statistics. One of the reasons for this is that the graphical depiction of the model contains no information about the content of the conditional probability tables. This is advantageous in that algorithms can be formulated for a graphical structure, independent of the details of the parameterisation of the local tables in the model. However, despite the elegance of such an approach, the issue of tractability can be heavily dependent on the form of the local probability tables. For example, for Gaussian tables all marginals are tractable although, in general, marginalising high dimensional distributions is highly non-trivial.

11.6 Problems

(Thanks to Chris Williams for some of these questions)

Exercise 27 (From Tipping, §2.1.3) . *Box 1 contains 8 apples and 4 oranges. Box 2 contains 10 apples and 2 oranges. Boxes are chosen with equal probability.*

What is the probability of choosing an apple? (3/4). If an apple is chosen, what is the probability that it came from box 1? (4/9)

Exercise 28 (R & N, Ex 14.5) *Prove from first principles the conditionalized version of the general product rule*

$$P(X, Y|Z) = P(X|Z)P(Y|X, Z).$$

Also prove the conditionalized version of Bayes' rule

$$P(X|Y, Z) = \frac{P(Y|X, Z)P(X|Z)}{P(Y|Z)}.$$

Exercise 29 (Elvis' twin) *Approximately 1/125 of all births are fraternal twins, and 1/300 births are identical twins. Elvis Presley had a twin brother (who died at birth). What is the probability that Elvis was an identical twin? You may approximate the probability of a boy or girl birth as 1/2. (Biological information: identical twins must be either both boys or both girls, as they are derived from one egg.)*

Exercise 30 (Prosecutor's fallacy) *This question concerns "DNA fingerprinting" evidence. The probability that there is a DNA match given that a person is innocent is estimated as* $1/100,000$*. Assume that the probability that there is a match given that a person is guilty is 1. Suppose that the defendant in a trial lives in a city where there are* $10,000$ *people who could have committed the crime, and that there is a DNA match to the defendant. Calculate* $P(guilty|\ DNA\ match)$*. How does this vary as the size of the population changes?*

Exercise 31 (The Three Prisoners problem) *(From Pearl, 1988) Three prisoners A, B and C are being tried for murder, and their verdicts will be read and their sentences executed tomorrow. They know only that one of them will be declared guilty and will be hanged while the other two will go free; the identity of the condemned prisoner is revealed to a reliable prison guard, but not to the prisoners.*

In the middle of the night Prisoner A makes the following request. "Please give this letter to one of my friends – to one who is to be released. You and I know that at least one of them will be released.". The guard carries out this request. Later prisoner A calls the guard and asks him to whom he gave the letter. The guard tells him that he gave the letter to prisoner B. What is the probability that prisoner A will be released?

Exercise 32 (The Monte Hall problem) *I have three boxes. In one I put a prize, and two are empty. I then mix up the boxes. You want to pick the box with the prize in it. You choose one box. I then open another one of the boxes and show that it is empty. I then give you the chance to change your choice of boxes—should you do so? How is this puzzle related to the Three Prisoners problem?*

Exercise 33 *Consider the following fictitious scientific information: The doctors find that people with Kreuzfeld-Jacob disease (KJ) almost invariably ate lots of hamburgers, thus* $p(\text{HamburgerEater}|\text{KJ}) = 0.9$*. The probability of an individual having* KJ *is currently rather low, about one in 100,000. The instance of eating lots of hamburgers is rather widespread, so* $p(\text{HamburgerEater}) = 0.5$.

1. *What is the probability that a regular hamburger eater will have Kreuzfeld-Jacob disease?*
2. *If the case had been that the number of people eating hamburgers was rather small, say* $p(\text{HamburgerEater}) = 0.001$, *what is the probability that a regular hamburger eater will have Kreuzfeld-Jacob disease? Comment on the difference with the result in the previous part of the question.*

Exercise 34 *There is a synergistic relationship between Asbestos (A) exposure, Smoking (S) and Cancer (C). A model describing this relationship is given by*

$$p(A, S, C) = p(C|A, S)p(A)p(S)$$

1. *What kinds of independence assumptions have been made in this model?*
2. *How could you adjust the above model to account for the fact that people who work in the building industry have a higher likelihood to also be smokers and also a higher likelihood to asbestos exposure?*

Exercise 35 *Explain why any distribution on a set of variables* $x = (x_1, \ldots, x_n)$ *can be written as a belief network :* $p(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p(x_i|pa(x_i))$, *where* $pa(x_i)$ *are the parental variables of* x_i.

Exercise 36 *Inspector Clouseau arrives at the scene of a crime. The victim lies dead in the room, and the inspector quickly finds the murder weapon, a knife. The Butler (B) and Maid (M) are his main suspects. The inspector has a prior belief of 0.8 that the Butler is the murderer, and a prior belief of 0.2 that the Maid is the murderer. These probabilities are independent in the sense that* $p(B, M) = p(B)p(M)$. *(It is possible that both the Butler and the Maid could be the murderer).*

The inspector's extensive criminal knowledge can be formulated mathematically as follows:

$$p(B = \text{murderer}) = 0.8, \qquad \text{p(M = murderer)} = 0.2$$

$$p(\text{knife used}|B = \text{not murderer}, M = \text{not murderer}) = 0.3$$

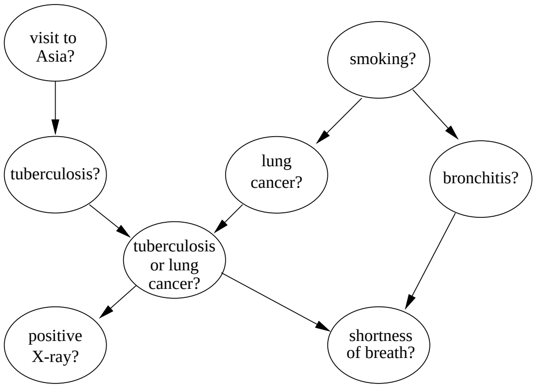
$$p(\text{knife used}|B = \text{not murderer}, M = \text{murderer}) = 0.2$$

$$p(\text{knife used}|B = \text{murderer}, \text{M} = \text{not murderer}) = 0.6$$

$$p(\text{knife used}|B = \text{murderer}, \text{M} = \text{murderer}) = 0.1$$

What is the the probability that the Butler is the murderer?

Exercise 37 *The belief network shown below is the famous "Asia" example of Lauritzen and Speigelhalter (1988). It concerns the diagnosis of lung disease (tuberculosis, lung cancer, or both, or neither). In this model a visit to Asia is assumed* *to increase the probability of tuberculosis.*



State if the following conditional independence relationships are true or false

$$I(tuberculosis?, smoking?|shortness\ of\ breath?),$$

$$I(lung\ cancer?, bronchitis?|smoking?),$$

$$I(visit\ to\ Asia?, smoking?|lung\ cancer?)$$

$$I(visit\ to\ Asia?, smoking?|lung\ cancer?, shortness\ of\ breath?).$$

Exercise 38 *Consider the three variable distribution*

$$p_1(a, b, c) = p(a|b)p(b|c)p(c)$$

where all variables are binary. How many parameters are needed to specify distributions of this form?

Now consider an undirected distribution on the same set of variables,

$$p_2(a, b, c) = \phi(a, b)\phi(b, c)/Z$$

where Z is a normalisation constant. How many parameters are needed to specify the parameters of the all the potential functions here? How many degrees of freedom do you think there are in this representation?

Discuss whether or not any setting of the tables in p_1 can be represented by some setting of the potential functions in p_2, and whether or not the opposite is true.

If you believe that p_1 and p_2 are different representations of the same family of distributions (ie any setting of the distribution in one representation can always be transformed expressed using some setting in the other representation), then discuss whether or not the potential function representation is an over-parameterisation of distributions from that class or not.

(For future discussion : explain the relationship between this question and Conditional Random Fields and whether or not CRF on a chain are an overparameterisation of an equivalent Markov Chain).

12 Inference in Belief Networks

12.1 Inference

Calculating conditional marginals, as in the Wet Grass example seen previously, is a form of inference. Although in simple graphs, such as the Wet Grass DAG, it is straightforward to carry out the calculations to calculate marginals by hand, in general, this problem can be computationally non-trivial. Fortunately, for singly-connected graphs (poly-trees) there exist efficient algorithms for inference, and it is instructive to understand how these algorithms work.

In this chapter we will consider two main algorithms, based on simple ideas. The first, *variable elimination* works on general mutliply-connected distributions (albeit not-necessarily efficiently) and is particularly appropriate for answering *single queries.*

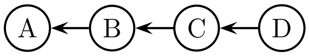
The second algorithm we consider is Pearl's Belief Propagation[8], which works only for singly-connected graphs, yet has the advantage that it can answer multiple queries efficiently.

These two classes of algorithms are useful as a precursor to developing algorithms that run, essentially as efficiently as can be reasonably made, on any graphical model (see the Junction Tree Algorithm chapter) and which are efficient in dealing with answering multiple queries.

12.2 Variable Elimination in a simple chain

Consider the distribution

$$p(a, b, c, d) = p(a|b)p(b|c)p(c|d)p(d)$$



and imagine that our inference task is to calculate the marginal distribution $p(a)$. Also, for simplicity, let's assume that each of the variables can take one of two states $\in \{0, 1\}$. Then

$$p(a = 0) = \sum_{b\in\{0,1\},c\in\{0,1\},d\in\{0,1\}} p(a = 0, b, c, d) \quad (12.2.1)$$

$$= \sum_{b\in\{0,1\},c\in\{0,1\},d\in\{0,1\}} p(a = 0|b)p(b|c)p(c|d)p(d) \quad (12.2.2)$$

It's clear that we could carry out this computation by simply enumerating each of the probabilities for the $2 * 2 * 2 = 8$ states of the variables b,c and d. However, in a more general chain of length T, this would imply that we would need a large number of summations $\propto e^T$. The question is how can we exploit the structure of the graph to reduce the number of summations required?.

In the above chain, it is clear that we may push the summation over d as far to the right as possible:

$$p(a=0) = \sum_{b\in\{0,1\},c\in\{0,1\}} p(a=0|b)p(b|c) \underbrace{\sum_{d\in\{0,1\}} p(c|d)p(d)}_{f_d(c)}$$

where $f_d(c)$ is a (two state) function. Similarly, we can distribute the summation over c as far to the right as possible:

$$p(a=0) = \sum_{b\in\{0,1\}} p(a=0|b) \underbrace{\sum_{c\in\{0,1\}} p(b|c)f_d(c)}_{f_c(b)}$$

Then, finally,

$$p(a=0) = \sum_{b\in\{0,1\}} p(a=0|b)f_c(b)$$

Hence, in this case, we have made $2+2+2=6$ summations. Whilst this saving may not appear much, the important point is that it is clear that the number of computations for a chain of length T would simply scale *linearly* with T, since each variable i that we sum over simply requires two additions to define the corresponding $f_i(i-1)$. This procedure is naturally enough called variable elimination, since each time we sum over the states of a variable, we eliminate it from the remaining distribution. It is clear that we can always perform variable elimination in a chain efficiently, since there is a natural way to distribute the summations "to the right" (or "left" as needs be).

Note that this is related to the associativity of matrix multiplication. We can define matrices

$$[M_{AB}]_{i,j} = p(a=i|b=j), [M_{BC}]_{i,j} = p(b=i|c=j),$$
$$[M_{CD}]_{i,j} = p(c=i|d=j), [M_D]_i = p(d=i), [M_A]_i = p(a=i)$$

Then the marginal M_A can be written

$$M_A = M_{AB}M_{BC}M_{CD}M_D = M_{AB}(M_{BC}(M_{CD}M_D))$$

since matrix multiplication is associative. This matrix formulation of calculating marginals is called the *transfer matrix* method, and is particularly popular in the physics literature.

If we had somehow been rather myopic and not realised that the distribution was a chain and, instead, had placed the summations non-optimally, we may still have ended up with an exponentially large amount of computation in a long chain – as in the case where we do not push variables at all, which as we saw above, may result in extreme inefficiency.

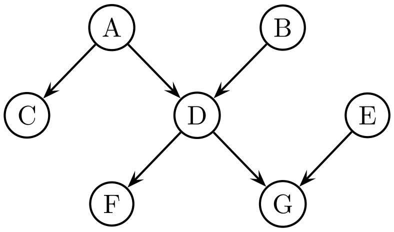


Figure 12.1: A simple Polytree. Inference is the problem of calculating the consequences of (possible) evidence injection on the individual nodes.

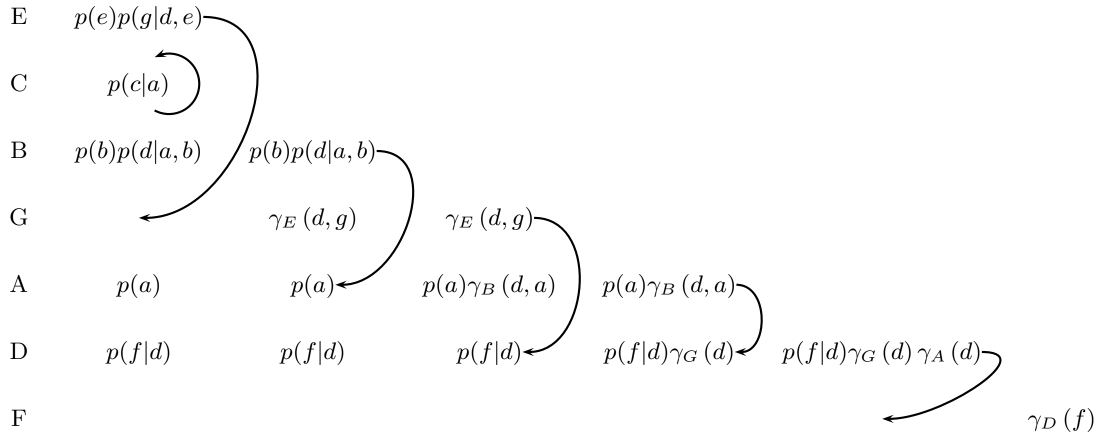


Figure 12.2: The bucket elimination algorithm applied to the graph fig(12.1). At each stage, at least one node is eliminated from the graph.

12.3 Bucket Elimination

We shall here consider a general *variable eliminiation* method for calculating marginals that works for *any* directed distribution (including multiply connected graphs). In itself, this is not a particularly popular nor useful algorithm (since it does not efficiently deal with multiple queries). Nevertheless, it does provide a simple way to calculate marginals, and is easy to remember.

It is helpful to consider a concrete example, such as that shown in fig(12.1). This polytree represents the distribution

$$p(a,b,c,d,e,f,g) = p(f|d)p(g|d,e)p(c|a)p(d|a,b)p(a)p(b)p(e).$$

For simplicity, we will consider calculating only marginals, since the generalisation to conditional marginals is straightforward. Consider the problem of calculating the marginal $p(f)$.

$$p(f) = \sum_{a,b,c,d,e,g} p(a,b,c,d,e,f,g) = \sum_{a,b,c,d,e,g} p(f|d)p(g|d,e)p(c|a)p(d|a,b)p(a)p(b)p(e)$$

We can distribute the summation over the various terms as follows: e,b and c are end nodes, so that we can sum over their values:

$$p(f) = \sum_{a,d,g} p(f|d)p(a) \left(\sum_b p(d|a,b)p(b) \right) \left(\sum_c p(c|a) \right) \left(\sum_e p(g|d,e)p(e) \right)$$

For convenience, lets write the terms in the brackets as $\sum_b p(d|a,b)p(b) \equiv \gamma_B\left(a,d\right)$, $\sum_e p(g|d,e)p(e) \equiv \gamma_E\left(d,g\right)$. The term $\sum_c p(c|a)$ is equal to unity, and we therefore eliminate this node directly. Rearranging terms, we can write

$$p(f) = \sum_{a,d,g} p(f|d)p(a)\gamma_B\left(a,d\right)\gamma_E\left(d,g\right)$$

If we think of this graphically, the effect of summing over b, c, e is effectively to remove or "eliminate" those nodes. We can now carry on summing over a and g since these are end points of the new graph:

$$p(f) = \sum_d p(f|d) \left(\sum_a p(a)\gamma_B\left(a,d\right) \right) \left(\sum_g \gamma_E\left(d,g\right) \right)$$

Again, this defines new functions $\gamma_A\left(d\right)$, $\gamma_G\left(d\right)$, so that the final answer can be found from

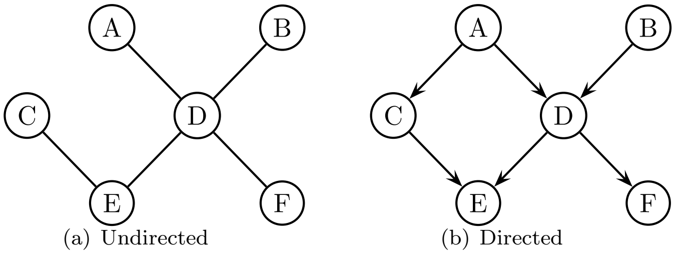
$$p(f) = \sum_d p(f|d)\gamma_A\left(d\right)\gamma_G\left(d\right)$$

We illustrate this graphically in fig(12.2). Initially, we define an ordering of the variables, beginning with the one that we wish to find the marginal for – a suitable ordering is therefore, f, d, a, g, b, c, e. Then starting with the highest node, e, we put all the functions that mention e in the e bucket. Continuing with the next highest bucket, c, we put all the remaining functions that mention c in this c bucket, etc. The result of this initialisation procedure is that terms (conditional distributions) in the DAG are distributed over the buckets, as shown in the left most column of fig(12.2). Eliminating then the highest bucket e, we pass a message to node g. Immediately, we can also eliminate bucket c since this sums to unity. In the next column, we have now two less buckets, and we eliminate the highest remaining bucket, this time b, passing a message to bucket a[17].

There are some important observations we can make about this procedure:

- For simplicity, assume that each variable can take two states. Then the total number of summations that the above procedure makes in calculating $p(f)$ is equal to two times the number of functions γ that we defined, plus two extra summations to eliminate the final node d. That is, the number of summation operations is $(n-1)*s$ where n is the number of variables in the graph, and s is the number of states of each of the variables. Hence, for this simple graph, the marginal $p(f)$ has been computed in a time which scales *linearly* with the number of variables in the graph. If we had attempted to compute $p(f)$ by brute force summation over the other nodes, this would have contained 2^n terms – that is, the brute force summation is exponentially costly. The independencies in this graph have enabled us to find an efficient procedure for computing $p(f)$.

- We could also have carried out the same kind of node elimination procedure to calculate the marginals $p(a)$, $p(b)$, $p(c)$, $p(d)$, $p(e)$ and $p(g)$ in a similar way. Here, however, each query (calculation of a marginal in this case) would require re-running the algorithm.



(a) Undirected (b) Directed

Figure 12.3: (a) An undirected graph. This graph is singly-connected – there is a unique path between any two nodes. (b) An acyclic directed graph. This graph is multiply connected, or *loopy* – we can link A and E through ACE or ADE.

- In general, bucket elimination constructs multi-variable messages – that is, not dependent on a single node, but potentially several nodes. For general polytrees, therefore, the complexity of this problem is exponential in the maximal family size, both in time and storage. It is also clear that if we repeated the above procedure for computing the marginals of other variables, we would end up redefining some of the same messages. It would be more efficient to simply reuse these messages, rather than recalculating them each time. There is an algorithm that can achieve this, and we describe this in the following section.
- It is clear that in trees (and in general, singly connected graphs) that we can always choose an ordering to render the amount of computation to scale linearly with the number of variables (nodes) in the graph. Such an ordering is called *perfect*, and indeed it can be shown that a perfect ordering can always easily be found for singly-connected graphs (see [18]). However, it is also clear that orderings exist for which Bucket Elimination will be extremely inefficient.
- The Bucket Elimination procedure will work also on undirected graphs, with the proviso that marginals will be computed upto a missing normalisation constant.

12.4 Belief Propagtion : Inference in Singly Connected Graphs

Graphical models come in two main flavours – directed and undirected, see fig(12.3). An undirected graph specifies a partial factorisation of the functional form of the joint probability distribution, up to an overall normalisation constant. The graph in fig(12.3a) represents distributions of the form

$$p\left(a, b, c, d, e, f\right) = \frac{1}{Z}\Psi\left(a, d\right)\Psi\left(b, d\right)\Psi\left(c, e\right)\Psi\left(d, e\right)\Psi\left(d, f\right) \qquad (12.4.1)$$

where Z is a constant which ensures correct normalisation of the distribution. The potential functions Ψ are non-negative functions, and describe the strength of links between neighbouring nodes. The potential functions do not correspond directly to probability distributions on their arguments, since the potentials themselves are not necessarily normalised.

Directed models correspond to a partial factorisation of the joint distribution into a set of conditional distributions, one for each node in the graph. The graph in fig(12.3b) represents distributions

$$p\left(a,b,c,d,e,f\right) = p(d|a,b)p(a)p(b)p(c|a)p(e|c,d)p(f|d).$$

The *parents* of a node are those nodes that feed into a node. For example, the parents of d, denoted $\mathrm{pa}\left(d\right)$, are a and b. Similarly, *children* are nodes which a node feeds into, so that the children of node d, denoted $\mathrm{ch}\left(d\right)$, are e and f. We will only consider directed graphs that are acyclic. That is, there is no path such that, starting at any node, and following the directed links to other nodes, we eventually return to the start node.

An important issue in the tractability of graphical models, directed or undirected, is whether or not the graph is "non-loopy" (singly-connected) or "loopy" (multiply-connected). Neglecting the directions of any links on the graph, a graph is non-loopy provided that for every pair of nodes, there is only a single path connecting them. Thus fig(12.3b) is loopy, since $ADECA$ forms a loop.

Roughly speaking, inference (finding marginals) in non-loopy graphs is straightforward, since such graphs are topologically equivalent to a tree. Belief Propagation (BP) is a procedure which exploits this non-loopy structure, and is an exact algorithm for inference on non-loopy graphs. Two versions of BP exist – one for directed graphs (DBP), and one for undirected graphs (UBP). Recent interest has been the (strictly erroneous) application of BP to loopy graphs as an approximate scheme for inference in otherwise intractable situations. Results suggest that, under certain conditions, such as long shortest-loop-length (meaning the number of links in the path that take the loop back to the starting node), the application of BP to loopy graphs can provide an accurate approximation. Indeed, a loop may be considered long if BP converges in a number of iterations less than the loop length. One of the difficulties in applying directed belief propagation is that, unlike the undirected version, its complexity is exponential in the number of parental connections of a node.

For completeness, we briefly present an intuitive derivation of the belief propagation algorithms. The central idea is to form a self-consistent message passing scheme, based on only locally available information to a node, and from the resulting iterated messages, calculate marginals of the graph.

12.4.1 Undirected Belief Propagation

In undirected graphs, the situation is a relatively straightforward. Consider calculating the marginal $p(d) = \sum_{a,b,c,e,f} p(a,b,c,d,e,f)$ for the pairwise Markov network in fig(12.3a). The partial functional factorisation of the distribution equation (12.4.1), enables us to distribute the summation. To keep the notation simple, we denote both a node and its state by the same symbol, so that $\sum_b \Psi(d,b)$ denotes summation over the states of the variable b. This results in a message $\lambda_{b\to d}\left(d\right)$ which contains information passing from node b to node d and is a function of the state of node d. This works as follows:

$$p(d) = \frac{1}{Z} \underbrace{\sum_b \Psi\left(b,d\right)}_{\lambda_{b\to d}(d)} \underbrace{\sum_a \Psi\left(a,d\right)}_{\lambda_{a\to d}(d)} \underbrace{\sum_f \Psi\left(d,f\right)}_{\lambda_{f\to d}(d)} \underbrace{\sum_e \Psi\left(d,e\right) \underbrace{\sum_c \Psi\left(c,e\right)}_{\lambda_{c\to e}(e)}}_{\lambda_{e\to d}(d)}$$

where we have defined messaged $\lambda_{n_1\to n_2}\left(n_2\right)$ sending information from node n_1 to node n_2 as a function of the state of node n_2. It is intuitively clear that we can in general define messages as

$$\lambda_{a \to b}(b) = \sum_a \Psi(a, b) \prod_{c \in N(a) \setminus b} \lambda_{c \to a}(a)$$

where $N(a)$ is the set of neighbouring nodes to a. Iterating these equations results in a convergent scheme for non-loopy graphs. The marginal is then found from $p(d) \propto \prod_{c \in N(d)} \lambda_{c \to d}(d)$, the prefactor being determined from normalisation. In contrast to directed belief propagation (described in the following section) whose complexity scales exponentially with the number of parents of a node, the complexity of calculating a message in undirected belief propagation scales only linearly with the number of neighbours of the node.

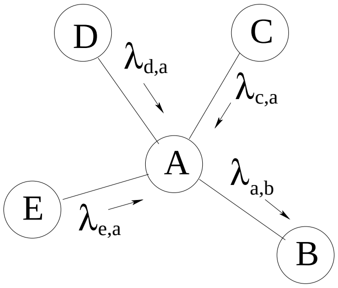


Figure 12.4: Undirected BP : calculating a message $\lambda_{a \to b}(b) = \sum_a \Psi(a, b) \lambda_{c \to a}(a) \lambda_{d \to a}(a) \lambda_{e \to a}(a)$.

12.4.2 Directed Belief Propagation

Intuition into the derivation of the directed belief propagation algorithm can also be gained by considering marginalisation on a simple graph, such as depicted in fig(12.5).

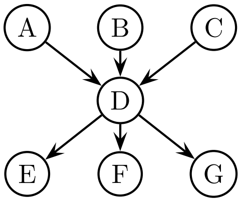


Figure 12.5: Intuition for DBP can be gleaned from consideration of simple graphs. The marginal $p(d)$ can be calculated from information passing from its parents a, b, c and children e, f, g.

This represents distributions of the form

$$p(a, b, c, d, e, f, g) = p(d|a, b, c)p(a)p(b)p(c)p(e|d)p(f|d)p(g|d).$$

Consider calculating the marginal $p(d)$. This involves summing the joint distribution over the remaining variables a, b, c, e, f, g. Due to the partial factorisation that the directed structure encodes, we can distribute this summation as follows

$$p\left(d\right)=\sum_{abc}p\left(d|a,b,c\right)\ \underbrace{p(a)}_{\rho_{a\to d}(a)}\ \underbrace{p(b)}_{\rho_{b\to d}(b)}\ \underbrace{p(c)}_{\rho_{c\to d}(c)}\ \underbrace{\sum_{e}p\left(e|d\right)}_{\lambda_{e\to d}(d)}\underbrace{\sum_{f}p\left(f|d\right)}_{\lambda_{f\to d}(d)}\underbrace{\sum_{g}p\left(g|d\right)}_{\lambda_{g\to d}(d)}. \tag{12.4.2}$$

We have defined here two types of messages for node d, λ messages that contain information passing up from children, and ρ messages that contain information passing down from parents.

If the children of node d are not fixed in any particular state (there is no "evidence"), then the λ messages are trivially 1. If however, there is evidence so that, for example, node e is fixed in state 1, then $\lambda_{e\to d}\left(d\right)=p\left(e=1|d\right)$.

It is clear that the marginal for any node can be calculated from the local messages incoming to that node. The issue that we now address is how to find a recursion for calculating such messages.

Consider the case where the graph of fig(12.5) has some extra connections, as in fig(12.5).

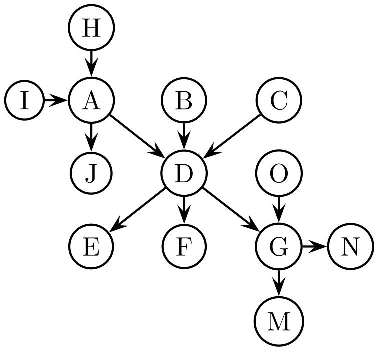


Figure 12.6: To calculate $p(d)$, we only need to adjust the messages passing from node a to node d and node g to node d. Information from all the other nodes is as in fig(12.5)

The only messages that need to be adjusted to find the marginal $p(d)$ are those from a to d, namely $\rho_{a\to d}\left(a\right)$ and from g to d, namely $\lambda_{g\to d}\left(d\right)$. The marginal for d will have exactly the same form as equation (12.4.2), except with the following adjustments to the messages

$$\rho_{a\to d}\left(a\right)=\sum_{h,i}p\left(a|h,i\right)\ \underbrace{p(h)}_{\rho_{h\to a}(h)}\ \underbrace{p(i)}_{\rho_{i\to a}(i)}\ \underbrace{\sum_{j}p(j|a)}_{\lambda_{j\to a}(a)}$$

$$\lambda_{g\to d}\left(d\right)=\sum_{g,o}p\left(g|o,d\right)\ \underbrace{p(o)}_{\rho_{o\to g}(o)}\ \underbrace{\sum_{m}p(m|g)}_{\lambda_{m\to g}(g)}\underbrace{\sum_{n}p(n|g)}_{\lambda_{n\to g}(g)}.$$

The structure of the above equations is that to pass a message from a node n_1 to a child node n_2, we need to take into account information from all the parents of n_1 and all the children of n_1, except n_2. Similarly, to pass a message from node n_2

to a parent node n_1, we need to gather information from all the children of node n_2 and all the parents of n_2, except n_1.

Essentially, to formulate the messages for a node, it is as if the parents were disconnected from the rest of the graph, with the effect of this disconnection being a modified prior for each parental node. Similarly, it is as if the children are disconnected from the rest of the graph, and the effect of the child nodes is represented by a modified function on the link (*e.g.,* instead of $p(e|d)$ we have $\lambda_{e,d}\left(d\right)$).

From these intuitions, we can readily generalise the situation to a formal algorithm.

The Directed Belief Propagation Algorithm

A general node d has messages coming in from parents and from children, and we can collect all the messages from parents that will then be sent through d to any subsequent children as[18]

$$\rho_d\left(d\right) = \sum_{\mathrm{pa}(d)} p\left(d|\mathrm{pa}\left(d\right)\right) \prod_{i \in \mathrm{pa}(d)} \rho_{i,d}\left(i\right).$$

Similarly, we can collect all the information coming from the children of node d that can subsequently be passed to any parents of d

$$\lambda_d\left(d\right) = \prod_{i \in \mathrm{ch}(d)} \lambda_{i,d}\left(d\right).$$

The messages are defined as

$$\lambda_{c,a}\left(a\right) = \sum_{c} \lambda_c\left(c\right) \sum_{i \in \mathrm{pa}(c) \backslash a} p\left(c|\mathrm{pa}\left(c\right)\right) \prod_{i \in \mathrm{pa}(c) \backslash a} \rho_{i,c}\left(i\right)$$

$$\rho_{b,d}\left(b\right) = \rho_b\left(b\right) \prod_{i \in \mathrm{ch}(b) \backslash d} \lambda_{i,b}\left(b\right)$$

The initialisation is as follows: For all evidential nodes i set

- $\rho_i\left(i\right) = 1$ for node i in the evidential state, 0 otherwise.
- $\lambda_i\left(i\right) = 1$ for node i in the evidential state, 0 otherwise.

For all non-evidential nodes i with no parents, set $\rho_i\left(i\right) = p(i)$. For all non-evidential nodes i with no children, set $\lambda_i\left(i\right) = 1$.

For every non-evidential node i we then iterate:

a) : If i has received the ρ-messages from all its parents, calculate $\rho_i\left(i\right)$.

b) : If i has received λ-messages from all its children, calculate $\lambda_i\left(i\right)$.

c) : If $\rho_i\left(i\right)$ has been calculated, then for every child j of i such that i has received the λ-messages from all of its other children, calculate and send the message $\rho_{i,j}\left(i\right)$.

d) : If $\lambda_i\left(i\right)$ has been calculated, then for every parent j of i such that i has received the ρ-messages from all of its other parents, calculate and send the message $\lambda_{i,j}\left(j\right)$.

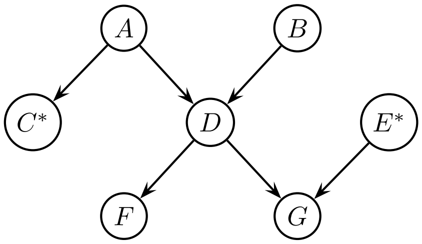


Figure 12.7: A simple singly connected distribution. Here variables C and E are clamped into evidential states, and we wish to infer the marginals $p(x|c, e)$ for the remaining unclamped variables $x \in \{a, b, d, f, g\}$

Repeat the above (a,b,c,d) until all the λ and ρ messages between any two adjacent nodes have been calculated. For all non-evidential nodes i compute $\rho_i\,(i)\,\lambda_i\,(i)$. The marginal $p(i|\text{evidence})$ is then found by normalising this value.

For binary valued nodes, both the λ and ρ messages for each node are binary vectors, expressing the messages as a function of the two states that the node can exist in. It is only the relative value of the messages in their two states which is important. For this reason, we are free to normalise both λ and ρ messages, which is useful in avoiding overflow and underflow problems.

The complexity of belief propagation is time exponential in the maximum family size and linear in space.

One of the main benefits of BP is that we can define a completely *local* algorithm. In the BP algorithm above, nodes had to wait for certain messages before they could pass other messages. However, if we start with some randomly chosen messages used for the nodes which are not in the initialisation procedure, then we claim that we can calculate the messages in *any* order. Provided that a sweep is made through all the nodes in the graph, then the messages will, after at most n iterations, converge to the correct messages given by BP. We will not prove this here, although the intuition why this is true is clear: Imagine a simple graph which is a chain. If we start somewhere in the middle of the chain, and calculate the messages, then these will be incorrect. However, provided that we sweep through all the nodes, eventually, we will hit the end points of the chain. Since the initialisation of the end nodes is correct, the messages from these end nodes will be calculated correctly. Similarly, when we repeat a sweep through all the nodes in the graph, we will pass through nodes which are adjacent to the end nodes. Since the messages from the end nodes are correct, the messages from the nodes adjacent to the end nodes will also be correct. We see therefore that the correct messages are filled in from the ends, one by one, and that eventually, after at most n sweeps though all the nodes, all the messages will be calculated correctly. This intuition holds also for the more general case of singly-connected graphs.

12.4.3 Example : Directed Belief Propagation

Let's perform inference for the distribution fig(12.7), in which c and e are evidential. To denote that c and e are clamped into some particular state, we use the notation $p^*(c|a)$ and $p^*(e)$ which sets these tables so that the states of c and e which do not correspond to the clamped states, have zero probability.

$$
\begin{aligned}
&\lambda_{c\to a}\left(a\right) = p^*(c|a)\\
&\lambda_{f\to d}\left(d\right) = \sum_f p(f|d) = 1\\
&\rho_{b\to d}\left(b\right) = p(b)\\
&\rho_{e\to g}\left(e\right) = p^*(e)\\
&\lambda_{g\to d}\left(d\right) = \sum_{g,e} p(g|d,e)\rho_{e\to g}\left(e\right) = \sum_e \rho_{e\to g}\left(e\right) = 1\\
&\rho_{a\to d}\left(a\right) = \sum_c \lambda_{c\to a}\left(a\right) p(a)\\
&\lambda_{d\to b}\left(b\right) = \sum_d \lambda_{g\to d}\left(d\right)\lambda_{f\to d}\left(d\right) p(d|a,b)\rho_{a\to d}\left(a\right)\\
&\lambda_{d\to a}\left(a\right) = \sum_d \lambda_{g\to d}\left(d\right)\lambda_{f\to d}\left(d\right) p(d|a,b)\rho_{b\to d}\left(b\right)\\
&\rho_{d\to f}\left(d\right) = \lambda_{g\to d}\left(d\right)\sum_{a,b} p(d|a,b)\rho_{a\to d}\left(a\right)\rho_{b\to d}\left(b\right)\\
&\rho_{d\to g}\left(d\right) = \lambda_{g\to d}\left(d\right)\sum_{a,b} p(d|a,b)\rho_{a\to d}\left(a\right)\rho_{b\to d}\left(b\right)
\end{aligned}
$$

Multiple Queries

Now that the messages have been calculated, we can easily find all the marginals using final values for the messages.

For example,

$$p(d|c,e) \propto \lambda_{f\to d}\left(d\right)\lambda_{g\to d}\left(d\right)\sum_{a,b} p(d|a,b)\rho_{a\to d}\left(a\right)\rho_{b\to d}\left(b\right)$$

Party Animal example

The party animal corresponds to the network in fig(12.8). Then, given that we observe that the Boss is Angry, and that the worker has a Headache, we wish to find the probability hat the worker has been to a party.

To complete the specifications, the probabilities are given as follows:

$$
\begin{aligned}
&p(u=T|p=T,d=T) = 0.999\\
&p(u=T|p=F,d=T) = 0.9\\
&p(u=T|p=T,d=F) = 0.9\\
&p(u=T|p=F,d=F) = 0.01
\end{aligned}
$$

$$p(d=T) = 0.05,\ \ p(h=T|p=T) = 0.9,\ \ p(h=T|p=F) = 0.1$$

$$p(a=T|u=T) = 0.99,\ \ p(a=T|u=F) = 0.2$$

Let's perform inference for the distribution fig(12.7), in which c and e are evidential. To denote that c and e are clamped into some particular state, we use the

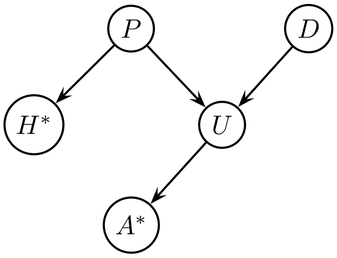


Figure 12.8: All variables are binary. When set to 1 the statements are true: P = Been to Party, H = Got a Headache, D = Demotivated at work, U = Underperform at work, A =Boss Angry. The stars denote that the variables are observed in the true state.

notation $p^*(c|a)$ and $p^*(e)$ which sets these tables so that the states of c and e which do not correspond to the clamped states, have zero probability.

$$\lambda_{h\to p}(p) = p(h^*|p) = \begin{pmatrix} 0.9 \\ 0.1 \end{pmatrix} \begin{array}{l} \text{true state, } T \\ \text{false state, } F \end{array}$$

$$\lambda_{a\to u}(u) = p(a^*|u) = \begin{pmatrix} 0.99 \\ 0.2 \end{pmatrix}$$

$$\rho_{d\to u}(d) = p(d) = \begin{pmatrix} 0.05 \\ 0.95 \end{pmatrix}$$

$$\lambda_{u\to p}(p) = \sum_u \lambda_{a\to u}(u) \sum_d p(u|p,d)\rho_{d\to u}(d) = \begin{pmatrix} 0.9150 \\ 0.2431 \end{pmatrix}$$

$$p(p|h,a) \propto p(p)\lambda_{h\to p}(p)\,\lambda_{u\to p}(p) = \begin{pmatrix} 0.1 \\ 0.9 \end{pmatrix} \cdot \begin{pmatrix} 0.9 \\ 0.1 \end{pmatrix} \cdot \begin{pmatrix} 0.9150 \\ 0.2431 \end{pmatrix}$$

$$p(p|h,a) = \begin{pmatrix} 0.7901 \\ 0.2099 \end{pmatrix}$$

12.5 Belief Revision

Belief propagation finds the marginal distribution of nodes in the graph. However, we may be interested in what is the most likely state of the nodes in the graph. That is

$$\operatorname*{argmax}_x p(x) = \operatorname*{argmax}_x \prod_{i=1}^{n} p(x_i|\text{pa}(x_i)) \qquad (12.5.1)$$

We can exploit the independency structure of the graph just as we did in belief propagation. That is, we can distribute the maximization operator over the network, so that only local computations are required. In fact, the only difference between the belief revision and the belief propagation algorithms is that wherever there was a summation in belief propagation, it is replaced with a maximization operation in belief revision.

To see more clearly why this is the case, consider a simple function which can be represented as an undirected chain,

$$f(x_1, x_2, x_2, x_4) = \phi(x_1, x_2)\phi(x_2, x_3)\phi(x_3, x_4)$$

and that we wish to find the joint state x^* which maximises f. Firstly, let's calculate the *value* of the state that maximises f (the corresponding state is straightforward to find by backtracking) :

$$
\begin{aligned}
\max_x f(x) &= \max_{x_1,x_2,x_3,x_4} \phi(x_1,x_2)\phi(x_2,x_3)\phi(x_3,x_4) \\
&= \max_{x_1,x_2,x_3} \phi(x_1,x_2)\phi(x_2,x_3) \underbrace{\max_{x_4} \phi(x_3,x_4)}_{\gamma(x_3)} \\
&= \max_{x_1,x_2} \phi(x_1,x_2) \underbrace{\max_{x_3} \phi(x_2,x_3)\gamma(x_3)}_{\gamma(x_2)} \\
&= \max_{x_1,x_2} \phi(x_1,x_2)\gamma(x_2)
\end{aligned}
$$

It is clear that the chain structure of the function, coupled with the maximisation operation, which may be distributed over the function, that the maximal value (and its state) can be computed in time which scales *linearly* with the number of factors in the function.

Note that there is no requirement here that the function f corresponds to a probability distribution. This hints that there is a more general class of functions and operations on them that permit computational simplifications, as described in the next section.

12.6 The Generalised Distributive Law

The GDL generalises the idea of how operations may be distributed over functions described in terms of products[19], and is based on the algebra of commutative semirings.

Commutative Semiring

A commutative semiring is a set K, together with two binary operations called "+" and "·", which satisfy the following three axioms:

S1 The operation "+" is associative and commutative, and there is an additive identity element called "0" such that $k + 0 = k$ for all $k \in K$. (This axiom makes $(K, +)$ a commutative monoid.)

S2 The operation "·" is also associative and commutative, and there is a multiplicative identity element called "1" such that $k \cdot 1 = k$ for all $k \in K$. (Thus $(K, \cdot)$ is also a commutative monoid.)

S3 The distributive law holds,

$$(a \cdot b) + (a \cdot c) = a \cdot (b + c)$$

i.e., for all triples (a, b, c) from K.

Then consider that we wish to "marginalize a product function" (MPF), as in the case where "+" indeed corresponds to addition and · to multiplication in the case of finding marginals in probability distributions written as products of factors (undirected representation). The standard computation of marginals in undirected graphical models corresponds in the GDL case to the so-called *sum-product* algorithm, since this is the form that the semiring takes in this case.

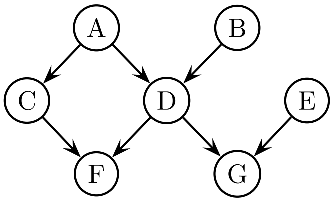


Figure 12.9: A multiply connected graph.

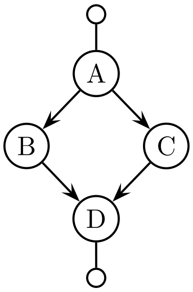


Figure 12.10: A part of a network which contains a loop.

The most-probable value of an undirected graphical model corresponds to MPF by setting " + " to the max operation, 0 to 0, $\cdot$ to multiplication and 1 to 1. This is called the *max-product* realisation of the GDL.

The properties of the semiring enable us to distribute the operations to compute the MPF (remember that what the MPF is corresponds to the particular realisation of the semiring). The idea is that one can then define *local* messages within the semiring structure (as we saw above for the case of sum-product and max-product definitions of γ messages on a chain). These messages can be generalised to graphs which are multiply connected. This is most easily achieved using the Junction Tree structure, as described in the next chapter.

An important point here is that there is no requirement that the algebra corresponds to a probability distribution. Indeed, this is why the area of *dynamic programming*[20] is not thought of as directly related to computations in graphical models.

12.7 Inference in Multiply-Connected Graphs

When we consider graphs which contain a loop (there is more than one path between two nodes in the graph when the directions are removed), we run into some difficulty with the belief propagation idea.

To offer an intuitive explanation why cycles cause difficulties, consider the application of belief propagation to the graph in fig(12.10). Since there is a loop, the initialisation procedure cannot be carried out such that we will be in a position to calculate messages. In order to begin the procedure, therefore, we need to make an assumption for at least one of the messages. Assume that we have assigned a value to $\rho_{a,b}(a)$. Since B has received its ρ messages from all its parents, and λ messages from all its other children (there are none), we can calculate the message

$\rho_{b,d}(b)$:

$$\rho_{b,d}(b) = \sum_a p(b|a)\rho_{a,b}(a)$$

Now we can find the message

$$\lambda_{d,c}(c) = \sum_{b,d} p(d|b,c)\rho_{b,d}(b)\,\lambda_D(d)$$

$$\lambda_{c,a}(a) = \sum_c p(c|a)\lambda_{d,c}(c)$$

Finally, we are in a position to calculate the message

$$\rho_{a,b}(a) = \rho(a)\lambda_{c,a}(a)$$

We now have a consistency requirement:

$$\begin{aligned}\rho_{a,b}(a) &= \rho(a)\lambda_{c,a}(a)\\ &= \rho(a)\sum_c p(c|a)\lambda_{d,c}(c)\\ &= \rho(a)\sum_c p(c|a)\sum_{b,d} p(d|b,e)\rho_{b,d}(b)\,\lambda_D(d)\\ &= \rho(a)\sum_c p(c|a)\sum_{b,d} p(d|b,c)\sum_a p(b|a)\rho_{a,b}(a)\,\lambda_D(d)\end{aligned}$$

Our initial assignment for $\rho_{a,b}(a)$ gets updated after we have cycled once around the loop. If there are other loops in the graph, the messages from this loop will get passed to other loops, which will feed back messages to this loop in return. There is no guarantee that the messages will ultimately converge to something consistent.

12.7.1 Conditioning

One way to solve the difficulties of multiply connected (loopy) graphs is to identify nodes that, if were not present, would mean that the reduced graph was singly-connected[8].

Consider the example if fig(12.11). Imagine that we wish to calculate a marginal, say $p(D)$. Then

$$p(d) = \sum_c \sum_{a,b,e,f,g} \underbrace{p(c|a)p(a)}_{p^*(a)} p(d|a,b)p(b) \underbrace{p(f|c,d)}_{p^*(f|d)} p(g|d,e)$$

where the p^* definitions are not necessarily distributions. It is clear that, for each state of c, the form of the products of factors remaining as a function of a, b, e, f, g is singly-connected, and that therefore, standard propagation (sum-product) methods can be used to perform inference. We will need to do this for as many states as there are in variable c, each state defining a new singly-connected graph (with the same structure) but modified potentials.

More generally, we can define a *set* of variables C, called the *loop-cut* set. So, for the price of a factor exponential in the loop-cut size, we can calculate the

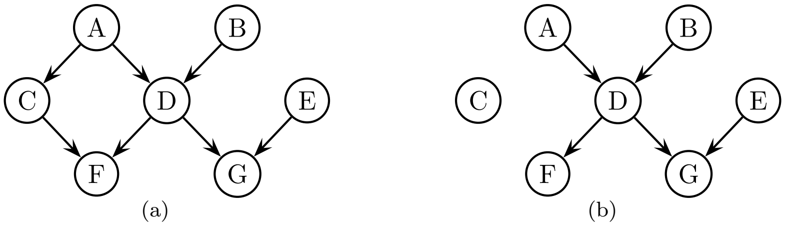


Figure 12.11: (a) A multiply connected graph reduced to a singly connected graph (b) by conditioning on the variable C.

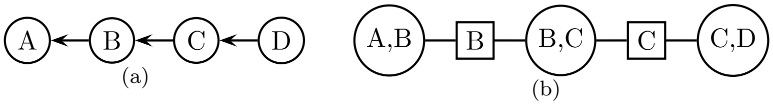


Figure 12.12: (a) A belief network. (b) A cluster graph representation of the network. The cluster potentials are defined on the round/oval nodes, and the separator potentials are defined on the square nodes, which share common variables with their neighbours.

marginals for any (loopy) DAG. The computationally difficult part of conditioning is in determining a small cut set, and there is no guarantee that this will anyway be small for a given graph.

Whilst this method is able to handle loops in a general manner, it is not particularly elegant, and no more efficient than the Junction Tree approach, as described in the next chapter.

12.8 Cluster Graphs

Consider a directed chain

$$p(U) = p(a|b)\, p(b|c)\, p(c|d)\, p(d) \tag{12.8.1}$$

where U, the "universe" represents all the variables in the graph. A cluster graph is defined as a set of potentials ψ_{C_i}, $i = 1, \ldots C$, where C_i is a cluster of variables, and ψ_{C_i} is a function on this set of variables. Each cluster potential is represented by a node in the cluster graph. A separator is defined as a potential on the variables from the intersection of C_i and C_j. The separator is drawn as a square node between the two clusters it separates. The cluster graph distribution is then defined as the product of all the cluster potentials, divided by the product of the separator potentials.

In fig(12.12), for the cluster graph to represent the BN we need

$$p(U) = p(a|b)\, p(b|c)\, p(c|d)\, p(d) = \frac{\Psi(a,b)\,\Psi(b,c)\,\Psi(c,d)}{\Psi(b)\,\Psi(c)}$$

One such assignment of the cluster and separator potentials to satisfy this would be $\Psi(a,b) = p(a|b)$, $\Psi(b,c) = p(b|c)$, $\Psi(c,d) = p(c|d)\,p(d)$, and $\Psi(b) = 1$, $\Psi(c) = 1$. Note that here we have defined the potentials to be functions of the (cluster) node of variables, in contrast to the example of an undirected graph given in a previous chapter, where the potentials where functions of the links and not of the nodes.

For every cluster representation, we claim that there exists another cluster representation for which the clusters contain the marginals of the distribution. For example, from the definition of conditional probability, we can rewrite equation (12.8.1) as

$$p(U) = \frac{p\left(a,b\right)p\left(b,c\right)p\left(c,d\right)}{p(b)p(c)} = \frac{\Psi^*\left(a,b\right)\Psi^*\left(b,c\right)\Psi^*\left(c,d\right)}{\Psi^*\left(b\right)\Psi^*\left(c\right)}$$

Where the cluster and separator potentials are set to $\Psi^*\left(a,b\right) = p\left(a,b\right)$, $\Psi^*\left(b,c\right) = p\left(b,c\right)$, $\Psi\left(c,d\right) = p\left(c,d\right)$, and $\Psi^*\left(b\right) = p(b)$, $\Psi^*\left(c\right) = p(c)$. It turns out that every singly-connected graph can always be represented as a product of the the clique marginals divided by the product of the separator marginals – this is a useful and widely applied result in the development of exact and approximate inference algorithms.

Directed graphs which are trees will always have a cluster representation of this form, since each factor in the distribution can be written using Bayes' rule, which will add a clique marginal and corresponding separator.

Cluster representation of Undirected Graphs

In the previous section, we saw an example of a directed graphical model that has a cluster representation. What about undirected graphical models – do they also have a cluster graph representation? In an undirected graph, each link contains a potential. Hence, the distribution in fig(12.13) is

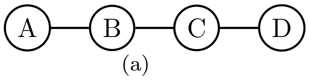


Figure 12.13: (a) An undirected graph.

$$p(a,b,c,d) = \phi(a,b)\phi(b,c)\phi(c,d)$$

Let's try to transform this into a different representation based on marginals of this distribution :

$$p(a,b) = \sum_{c,d} p(a,b,c,d) = \phi(a,b)\sum_{c,d}\phi(b,c)\phi(c,d)$$

$$p(b,c) = \sum_{a,d} p(a,b,c,d) = \phi(b,c)\sum_{a,d}\phi(a,b)\phi(c,d) = \phi(b,c)\sum_{a}\phi(a,b)\sum_{d}\phi(c,d)$$

$$p(c,d) = \sum_{a,b} p(a,b,c,d) = \phi(c,d)\sum_{a,b}\phi(a,b)\phi(b,c)$$

Using these, we can rewrite the distribution as

$$p(a,b,c,d) = \frac{p(a,b)}{\sum_{c,d}\phi(b,c)\phi(c,d)}\frac{p(b,c)}{\sum_{a}\phi(a,b)\sum_{d}\phi(c,d)}\frac{p(c,d)}{\sum_{a,b}\phi(a,b)\phi(b,c)}$$

Let's examine the denominator. This can be rewritten as

$$\underbrace{\sum_{a,c,d}\phi(a,b)\phi(b,c)\phi(c,d)}_{p(b)} \quad \underbrace{\sum_{a,b,d}\phi(a,b)\phi(b,c)\phi(c,d)}_{p(c)}$$

which, gives simply:

$$p(a,b,c,d) = \frac{p(a,b)p(b,c)p(c,d)}{p(b)p(c)}$$

and hence has the same cluster potential representation as fig(12.12b).

Indeed, one can show that any singly-connected undirected graph can be represented in the same way, as a product of clique potentials divided by separator potentials, and that there exists a setting for these values for which the cliques contain the marginals (on the variables in each clique) of the distribution.

Variable Elimination on Singly-Connected Graphs

The previous sections discussed how we can always represent a singly-connected graphical model (whether directed or undirected) in the form of a cluster representation[1].

However, we didn't specify explicitly an algorithm. It is clear that we can calculate any marginal by simple variable elimination, in which we start at the leaves of the tree, and eliminate the variable there, and then work inwards, nibbling off each time a leaf of the remaining tree. This is guaranteed to enable us to calculate any marginal $p(x_i)$ in roughly the number of variables in the graph multiplied by the largest summation.

A key point for a tree (singly-connected graph) is that provided we perform elimination from the leaves inwards, then the structure of the remaining graph is simply a subtree of the original tree, albeit with modified potentials.

The dissatisfying thing about this is that we would need to rerun variable elimination each time afresh to calculate a new marginal.

12.9 KL divergence approach to marginalisation on Trees

Since we know that the distribution has the form of the cluster graph representation, we could possibly minimise the KL divergence between the cluster representation and the original distribution (see appendix). Consider again the distribution

$$p = \phi_1(a,b)\phi_2(b,c)\phi_3(c,d)$$

We know that this has a representation of the form

$$q = \frac{q(a,b)q(b,c)(c,d)}{q(b)q(c)}$$

In order to find the settings of q that make a perfect match between the distributions q and p, we can minimise the Kullback-Leibler divergence between the two

[1] In the directed case, we need to carry out a moralisation step– see later section

distributions (see appendix). Thanks to the structured form of q and p, this is

$$
\begin{aligned}
KL(q||p) = \sum_{a,b} q(a,b)\log q(a,b) + \sum_{b,c} q(b,c)\log q(b,c) + \sum_{c,d} q(c,d)\log q(c,d) \\
- \sum_{b} q(b)\log q(b) - \sum_{c} q(c)\log q(c) - \\
\sum_{a,b} q(a,b)\log\phi_1(a,b) - \sum_{b,c} q(b,c)\log\phi_2(b,c) - \sum_{c,d} q(c,d)\log\phi_3(c,d) -
\end{aligned}
\tag{12.9.1}
$$

However, we have bear in mind that there are consistency constraints, namely $\sum_a q(a,b) = q(b) = \sum_c q(b,c)$ and $\sum_b q(b,c) = q(c) = \sum_d q(c,d)$. In addition, there are constraints that each probability must sum to 1. (I'll ignore these since they do no interact with other terms, and just simply will constitute a rescaling of the tables).

We can enforce these constraints by adding Lagrange multipliers (the reason for the labeling of the Lagrange multipliers will become clearer later):

$$
\begin{aligned}
&\sum_b \gamma_{21}(b)\left(\sum_a q(a,b) - q(b)\right) \\
&\sum_b \gamma_{12}(b)\left(\sum_c q(b,c) - q(b)\right) \\
&\sum_c \gamma_{32}(c)\left(\sum_b q(b,c) - q(c)\right) \\
&\sum_c \gamma_{23}(c)\left(\sum_d q(c,d) - q(c)\right)
\end{aligned}
$$

Let's now differentiate wrt $q(b,c)$. This gives, at the extreme point

$$\log q(b,c) - \log\phi_2(b,c) - \gamma_{12}(b) - \gamma_{32}(c) = 0$$

Or

$$q(b,c) \propto \phi_2(b,c)\lambda_{12}(b)\lambda_{32}(c)$$

where $\lambda(x) = \exp(\gamma(x))$. Similarly,

$$q(a,b) \propto \phi_1(a,b)\lambda_{21}(b)$$

and

$$q(c,d) \propto \phi_3(c,d)\lambda_{23}(c)$$

Similarly, we can differentiate wrt $q(b)$:

$$\log q(b) - \gamma_{21}(b) - \gamma_{12}(b)$$

Or

$$q(b) \propto \lambda_{21}(b)\lambda_{12}(b)$$

Similarly,

$$q(c) \propto \lambda_{32}(c)\lambda_{23}(c)$$

The marginalisation constraint $\sum_a q(a,b) = q(b)$ gives

$$\sum_a \phi_1(a,b)\lambda_{21}(b) \propto \lambda_{21}(b)\lambda_{12}(b)$$

or

$$\sum_a \phi_1(a,b) \propto \lambda_{12}(b)$$

Similarly, $\sum_c q(b,c) = q(b)$ gives

$$\sum_c \phi_2(b,c)\lambda_{12}(b)\lambda_{32}(c) \propto \lambda_{21}(b)\lambda_{12}(b)$$

or

$$\sum_c \phi_2(b,c)\lambda_{32}(c) \propto \lambda_{21}(b)$$

The constraint, $\sum_b q(b,c) = q(c)$ gives

$$\sum_b \phi_2(b,c)\lambda_{12}(b)\lambda_{32}(c) \propto \lambda_{32}(c)\lambda_{23}(c)$$

or

$$\sum_b \phi_2(b,c)\lambda_{12}(b) \propto \lambda_{23}(c)$$

And finally, $\sum_d q(c,d) = q(c)$ gives

$$\sum_d \phi_3(c,d)\lambda_{23}(c) \propto \lambda_{32}(c)\lambda_{23}(c)$$

or

$$\sum_d \phi_3(c,d) \propto \lambda_{32}(c)$$

Note that the normalisation constants in the λ messages can be dropped since we do not need to know these – they are only used at the end to calculate the normalisation of the marginals. Hence, the proportions may be written as equalities. This gives

$$\lambda_{12}(b) = \sum_a \phi_1(a,b) \tag{12.9.2}$$

$$\lambda_{23}(c) = \sum_b \phi_2(b,c)\lambda_{12}(b) \tag{12.9.3}$$

$$\lambda_{32}(c) = \sum_d \phi_3(c,d) \tag{12.9.4}$$

$$\lambda_{21}(b) = \sum_c \phi_2(b,c)\lambda_{32}(c) \tag{12.9.5}$$

These equations have a definite starting point (at each end), and can then be solved deterministically. This means that, for trees (and singly-connected structures) there is only one fixed point, and this is indeed the minimum zero KL divergence solution. These equations define the Belief Propagation Algorithm and demonstrates why (which was based on the fact that a tree has this representation) the solution found by iterating these equations is unique, and is the global minimum. Note that when the equations are used in the order defined above, then we always have enough information in the preceding equations in order to define the next equation. This is always the case for trees, and any ordering which starts from the leaves inwards will be fine. Note that the final marginals are determined directly from the λ messages, as given in the previous equations – just a simple normalisation is required.

There is an interesting property of these Belief Propagation equations, namely that they can be parallelised, since the forward and backward equations for the messages do not interfere. Once these two sets of messages have been calculated, they can be locally combined to produce marginals. This is in contrast to the Hugin scheme below which cannot be parallelised in the same way – however, in the case of trees with many branches, the Hugin approach may be slightly more efficient since it does not need to calculate products of messages, which is the case for the Belief Propagation style approach.

An Alternative : Hugin

We can rewrite the above equations in a manner that will be more efficient in some cases, particularly when there are many branches in the cluster graph.

Recall that for the running example we have been considering, the cluster graph is of the form :

$$q(a,b,c,d) = \frac{\phi(a,b)\phi(b,c)\phi(c,d)}{\phi(b)\phi(c)}$$

where we initially set $\phi(b)$ and $\phi(c)$ to be the identity functions, and potentials in the numerator to those that make a match with the distribution p.

The forward equations for λ can be run independently of the backward equations since the two sets of equations do not interfere. We consider here an alternative, in which we run first the forward equations, and subsequently the backward equations.

Let's start with $\lambda_{12}(b)$. We shall call this a new potential:

$$\phi^*(b) = \sum_a \phi(a,b)$$

If we wish this new potential to replace the old $\phi(b)$, but q to remain unchanged (this is desired if the initialisation corresponds to the correct distribution p), then we need to make the change:

$$q = \frac{\phi(a,b)\phi(b,c)\frac{\phi^*(b)}{\phi(b)}\phi(c,d)}{\phi^*(b)\phi(c)}$$

Let's therefore define

$$\phi^*(b,c) = \phi(b,c)\frac{\phi^*(b)}{\phi(b)}$$

so that q is simply

$$q = \frac{\phi(a,b)\phi^*(b,c)\phi(c,d)}{\phi^*(b)\phi(c)}$$

Now consider $\lambda_{23}(c)$. We shall call this a new potential $\phi^*(c)$. A simple substitution reveals

$$\phi^*(c) = \sum_b \phi^*(b,c)$$

Again, in order to leave the distribution unchanged, we make define

$$\phi^*(c,d) = \phi(c,d)\frac{\phi^*(c)}{\phi(c)}$$

Now, according to our equation $q(c,d) \propto \phi(c,d)\lambda_{2,3}(c)$, and since $\phi(c) = 1$, then

$$\phi^*(c,d) \propto p(c,d)$$

Now, rather than simply using the backward λ equations, we will use the marginal equations

$$q(b,c) \propto \underbrace{\phi(b,c)\lambda_{1,2}(b)}_{\phi^*(b,c)} \lambda_{3,2}(c)$$

Using $q(c) \propto \lambda_{3,2}(c)\lambda_{2,3}(c)$, we have

$$q(b,c) \propto \frac{\phi^*(b,c)q(c)}{\phi^*(c)}$$

But, since $q(c) \propto \phi^{**}(c)$, we may define

$$\phi^{**}(b,c) = \phi^*(b,c)\frac{\phi^{**}(c)}{\phi^*(c)}$$

and this will be proportional to the marginal $p(b,c)$. Similarly, a final step shows that

$$\phi^{**}(b) = \sum_c \phi^{**}(b,c), \qquad \text{and} \qquad \phi^{**}(a,b) = \phi(a,b)\frac{\phi^{**}(b)}{\phi^*(b)}$$

is proportional to $p(a,b)$.

These equations have the pleasant form that, in general one defines a new *separator*:

$$\phi^*(s) = \sum_{v \backslash s} \phi(v)$$

and *potential*

$$\phi^*(w) = \phi(w)\frac{\phi^*(s)}{\phi(s)}$$

Once one has then done a full forward sweep and backward sweep of these equations, the final potentials will be proportional to the marginal. Similarly, one can show that the final separators will be also proportional to the marginal of the separator variable). This is a reformulation of the Belief Propagation equations based on defining separators and new cluster potentials. The benefit of doing this in lieu of the BP approach is not immediately obvious. Hopefully it will become more apparent in the general case below.

The General Case on a Singly-Connected graph

Let v_i be a set of variables on a cluster $\phi(v_i)$. Our task is, from the general distribution

$$p(v) = \prod_i \phi(v_i)$$

To find a representation

$$q(v) = \frac{\prod_i q(v_i)}{\prod_{<ij>} q(v_{ij})}$$

here $v_{ij} = v_i \cap v_j$, $< ij >$ means the set of pairs i, j for which cluster i is a neighbour of cluster j. v is the union of all the v_i. Below $\sum_{v_i} f(v_i)$ means the sum of the clique function $f(v_i)$ over all the joint states of the set of variables v_i.

Then $KL(q||p)$ may be written as

$$\sum_i \sum_{v_i} q(v_i) \log q(v_i) - \sum_{<ij>} \sum_{v_{ij}} q(v_{ij}) \log q(v_{ij}) - \sum_i \sum_{v_i} q(v_i) \log \psi(v_i) \\ - \sum_{<ij>} \sum_{v_{ij}} \gamma_{ji}(v_{ij}) \left(\sum_{v_i \setminus v_{ij}} q(v_i) - q(v_{ij}) \right) \quad (12.9.6)$$

(as before, the normalisation constraints have been omitted since they play only a trivial non-interacting role).

Differentiating wrt to $q(v_i)$ and $q(v_{ij})$ we arrive at

$$q(v_i) \propto \psi(v_i) \prod_{j \in n(i)} \lambda_{ji}(v_{ij})$$

where $n(i)$ is the set of neighbours of cluster i, and

$$q(v_{ij}) \propto \lambda_{ij}(v_{ij}) \lambda_{ji}(v_{ij})$$

Using the marginalisation constraint

$$\sum_{v_i \setminus v_{ij}} q(v_i) = q(v_{ij})$$

we have

$$\sum_{v_i \setminus v_{ij}} \psi(v_i) \prod_{k \in n(i)} \lambda_{ki}(v_{ik}) = \lambda_{ij}(v_{ij}) \lambda_{ji}(v_{ij})$$

The term $\lambda_{ji}(v_{ji})$ cancels to give

$$\lambda_{ij}(v_{ij}) = \sum_{v_i \setminus v_{ij}} \psi(v_i) \prod_{k \in n(i), k \neq j} \lambda_{ki}(v_{ik})$$

which are the usual Belief Propagation equations. This is also called the Shenoy-Shafer updating scheme. However, if we didn't cancel the λ terms, we could rewrite

$$\sum_{v_i \setminus v_{ij}} q^{old}(v_i) = q^{new}(v_{ij}) \quad (12.9.7)$$

Hence, we can form updates for the separator potentials in this fashion. What about the potentials in the numerator? In order to do this, we need to invoke another piece of information. One approach is that if we made an initial assignment that the q was equal to p (say by making a simple assignment of the numerators, with the separators set to unity), then we need q to be *invariant* under the transformation equation (12.9.7). The separator $q(v_{ij})$ always occurs with a numerator term $q(v_j)$, and hence the requirement that the distribution q remains equal to p reduces to the requirement

$$\frac{q^{new}(v_j)}{q^{new}(v_{ij})} = \frac{q^{old}(v_j)}{q^{old}(v_{ij})}$$

or

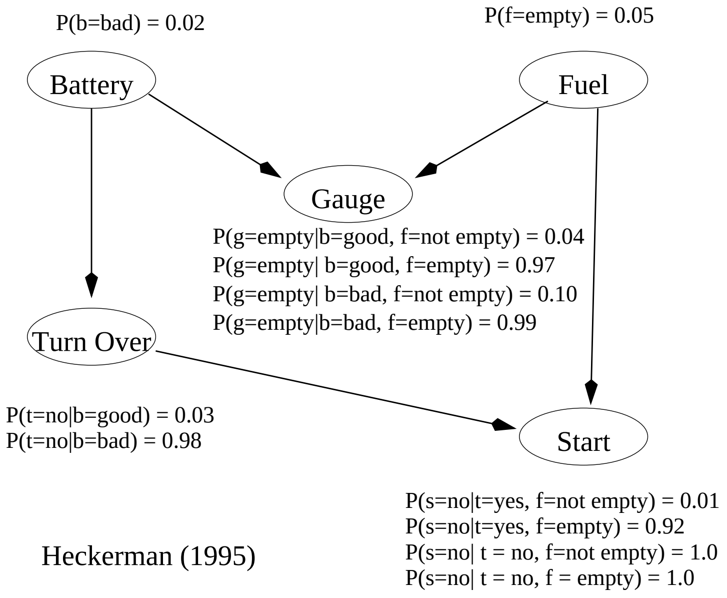
$$q^{new}(v_j) = q^{old}(v_j)\frac{q^{new}(v_{ij})}{q^{old}(v_{ij})} \tag{12.9.8}$$

This form of updates is called *Hugin propagation*, and together the equations equation (12.9.8) and equation (12.9.7) form a procedure called *absorption*.

12.10 Problems

Exercise 39 *Consider the distribution $p(a, b, c) = p(c|a, b)p(a)p(b)$. If we clamp b into an evidential state, what effect will this have on a? Explain your answer intuitively.*

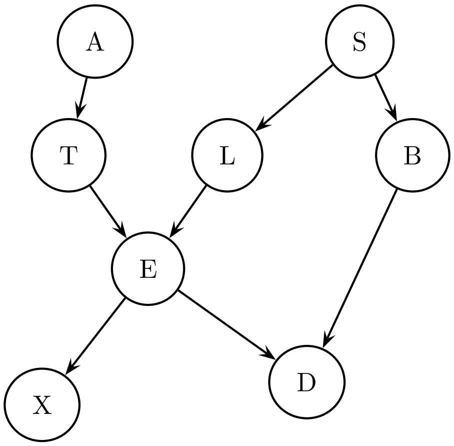
Exercise 40 *Consider the belief network given below, which concerns the probability of a car starting.*


P(b=bad) = 0.02
P(f=empty) = 0.05
Battery
Fuel
Gauge
P(g=empty|b=good, f=not empty) = 0.04
P(g=empty| b=good, f=empty) = 0.97
P(g=empty| b=bad, f=not empty) = 0.10
P(g=empty|b=bad, f=empty) = 0.99
Turn Over
P(t=no|b=good) = 0.03
P(t=no|b=bad) = 0.98
Start
P(s=no|t=yes, f=not empty) = 0.01
P(s=no|t=yes, f=empty) = 0.92
P(s=no| t = no, f=not empty) = 1.0
P(s=no| t = no, f = empty) = 1.0

Heckerman (1995)

Calculate $P(f = empty|s = no)$, the probability of the fuel tank being empty conditioned on the observation that the car does not start. Do this calculation "by hand", i.e. do not use or create a computer program to do this.

Exercise 41 *Consider the Asia Bayesian Network represented below. Calculate by hand the values for* $p(D)$, $p(D|S = \text{yes})$, $p(D|S = \text{no})$.



The table values are:

$$
\begin{aligned}
&p(a = \text{yes}) = 0.01\\
&p(s = \text{yes}) = 0.5\\
&p(t = \text{yes}|a = \text{yes}) = 0.05\\
&p(t = \text{yes}|a = \text{no}) = 0.01\\
&p(l = \text{yes}|s = \text{yes}) = 0.1\\
&p(l = \text{yes}|s = \text{no}) = 0.01\\
&p(b = \text{yes}|s = \text{yes}) = 0.6\\
&p(b = \text{yes}|s = \text{no}) = 0.3\\
&p(e = \text{yes}|t, l) = 0 \textit{ only if both } T \textit{ and } L \textit{ are``no'', 1 otherwise.}\\
&p(x = \text{yes}|e = \text{yes}) = 0.98\\
&p(x = \text{yes}|e = \text{no}) = 0.05\\
&p(d = \text{yes}|e = \text{yes}, b = \text{yes}) = 0.9\\
&p(d = \text{yes}|e = \text{yes}, b = \text{no}) = 0.3\\
&p(d = \text{yes}|e = \text{no}, b = \text{yes}) = 0.2\\
&p(d = \text{yes}|e = \text{no}, b = \text{no}) = 0.1
\end{aligned}
$$

12.11 Solutions

41 (Thanks to Peter Mattsson for typesetting this) The marginal $p(d)$ can be calculated as

$$\begin{aligned}
p(d) &= \sum_{a,s,t,l,b,e,x} p(a,s,t,l,b,e,x,d) \\
&= \sum_{a,s,t,l,b,e,x} p(a)p(s)p(t|a)p(l|s)p(b|s)p(e|t,l)p(x|e)p(d|b,e) \\
&= \sum_{a,s,t,l,b,e} \left(p(a)p(s)p(t|a)p(l|s)p(b|s)p(e|t,l)p(d|b,e) \sum_x p(x|e) \right) \\
&= \sum_{s,t,l,b,e} \left(p(s)p(l|s)p(b|s)p(e|t,l)p(d|b,e) \sum_a p(t|a)p(a) \right),
\end{aligned}$$

where we have noted that $\sum_x p(x|e) = 1$. The final term on the RHS is just the marginal $p(t)$, which is

$$\begin{aligned}
p(t = \text{yes}) &= p(t = \text{yes}|a = \text{yes}) \times p(a = \text{yes}) + p(t = \text{yes}|a = \text{no}) \times p(a = \text{no}) \\
&= 0.05 \times 0.01 + 0.01 \times 0.99 = 0.01 \times 1.04 \\
&= 0.0104.
\end{aligned}$$

Armed with this, we can further simplify the expression for $p(d)$ to

$$p(d) = \sum_{s,l,b,e} \left(p(s)p(l|s)p(b|s)p(d|b,e) \sum_t p(e|t,l)p(t) \right).$$

The last term on the RHS is now $p(e|l)$, which is

$$\begin{aligned}
p(e = \text{yes}|l = \text{yes}) =& p(e = \text{yes}|l = \text{yes}, t = \text{yes}) \times p(t = \text{yes}) \\
&+ p(e = \text{yes}|l = \text{yes}, t = \text{no}) \times p(t = \text{no}) \\
=& 1 \times 0.0104 + 1 \times 0.9896 = 1, \\
p(e = \text{yes}|l = \text{no}) =& p(e = \text{yes}|l = \text{no}, t = \text{yes}) \times p(t = \text{yes}) \\
&+ p(e = \text{yes}|l = \text{no}, t = \text{no}) \times p(t = \text{no}) \\
=& 1 \times 0.0104 + 0 \times 0.9896 = 0.0104.
\end{aligned}$$

The marginal $p(d)$ is now

$$p(d) = \sum_{s,b,e} p(s)p(b|s)p(d|b,e) \sum_l p(e|l)p(l|s),$$

which we can further simplify by calculating $p(e|s)$, which is

$$\begin{aligned}
p(e = \text{yes}|s = \text{yes}) =& p(e = \text{yes}|l = \text{yes}) \times p(l = \text{yes}|s = \text{yes}) \\
&+ p(e = \text{yes}|l = \text{no}) \times p(l = \text{no}|s = \text{yes}) \\
=& 1 \times 0.1 + 0.0104 \times 0.9 = 0.10936, \\
p(e = \text{yes}|s = \text{no}) =& p(e = \text{yes}|l = \text{yes}) \times p(l = \text{yes}|s = \text{no}) \\
&+ p(e = \text{yes}|l = \text{no}) \times p(l = \text{no}|s = \text{no}) \\
=& 1 \times 0.01 + 0.0104 \times 0.99 = 0.020296.
\end{aligned}$$

This now gives us

$$p(d) = \sum_{s,b} p(b|s)p(s) \sum_{e} p(d|b,e)p(e|s),$$

leading us to calculate $p(d|b,s)$, which is

$$\begin{aligned}
p(d = \text{yes}|b = \text{yes}, s = \text{yes}) =& p(d = \text{yes}|b = \text{yes}, e = \text{yes}) \times p(e = \text{yes}|s = \text{yes}) \\
&+ p(d = \text{yes}|b = \text{yes}, e = \text{no}) \times p(e = \text{no}|s = \text{yes}) \\
=& 0.9 \times 0.10936 + 0.2 \times 0.89064 = 0.276552, \\
p(d = \text{yes}|b = \text{yes}, s = \text{no}) =& p(d = \text{yes}|b = \text{yes}, e = \text{yes}) \times p(e = \text{yes}|s = \text{no}) \\
&+ p(d = \text{yes}|b = \text{yes}, e = \text{no}) \times p(e = \text{no}|s = \text{no}) \\
=& 0.9 \times 0.020296 + 0.2 \times 0.979704 = 0.2142072, \\
p(d = \text{yes}|b = \text{no}, s = \text{yes}) =& p(d = \text{yes}|b = \text{no}, e = \text{yes}) \times p(e = \text{yes}|s = \text{yes}) \\
&+ p(d = \text{yes}|b = \text{no}, e = \text{no}) \times p(e = \text{no}|s = \text{yes}) \\
=& 0.3 \times 0.10936 + 0.1 \times 0.89064 = 0.121872, \\
p(d = \text{yes}|b = \text{no}, s = \text{no}) =& p(d = \text{yes}|b = \text{no}, e = \text{yes}) \times p(e = \text{yes}|s = \text{no}) \\
&+ p(d = \text{yes}|b = \text{no}, e = \text{no}) \times p(e = \text{no}|s = \text{no}) \\
=& 0.3 \times 0.020296 + 0.1 \times 0.979704 = 0.1040592.
\end{aligned}$$

We now have

$$p(d) = \sum_{s} p(s) \sum_{b} p(d|b,s)p(b|s),$$

so we calculate $p(d|s)$, which is

$$\begin{aligned}
p(d = \text{yes}|s = \text{yes}) =& p(d = \text{yes}|b = \text{yes}, s = \text{yes}) \times p(b = \text{yes}|s = \text{yes}) \\
&+ p(d = \text{yes}|b = \text{no}, s = \text{yes}) \times p(b = \text{no}|s = \text{yes}) \\
=& 0.276552 \times 0.6 + 0.121872 \times 0.4 = 0.21468, \\
p(d = \text{yes}|s = \text{no}) =& p(d = \text{yes}|b = \text{yes}, s = \text{no}) \times p(b = \text{yes}|s = \text{no}) \\
&+ p(d = \text{yes}|b = \text{no}, s = \text{no}) \times p(b = \text{no}|s = \text{no}) \\
=& 0.2142072 \times 0.3 + 0.1040592 \times 0.7 = 0.1371036.
\end{aligned}$$

Now, at last, we can calculate $p(d) = \sum_s p(d|s)p(s)$, which is

$$\begin{aligned}
p(d = \text{yes}) =& p(d = \text{yes}|s = \text{yes}) \times p(s = \text{yes}) + p(d = \text{yes}|s = \text{no}) \times p(s = \text{no}) \\
=& 0.21468 \times 0.5 + 0.1371036 \times 0.5 = 0.1758918.
\end{aligned}$$

Thus we have $p(d = \text{yes}) = 0.1758918$, $p(d = \text{yes}|s = \text{yes}) = 0.21468$ and $p(d = \text{yes}|s = \text{no}) = 0.1371036$.

13 The Junction Tree Algorithm

Parts of this chapter are based on *Expert Systems and Probabilistic Network Model* by E. Castillo, J. Gutierrez, and A. Hadi (Springer, 1997), and also *An introduction to bayesian networks* by F.V.Jensen (Springer 1996). Both are excellent introductions to the field.

One may have had the nagging suspicion during the discussion of the algorithms relating to DAGs, that what we are really exploiting is the underlying graphical structure, whether this be directed or not. Does it really matter if the graph is directed? For example, in bucket elimination, we are essentially removing the directedness of the graph and defining (undirected) functions in their place. What this suggests is that the complexity of the calculations on directed graphs can be transformed into an undirected graph, possibly of greater connectivity than the directed graph from which it was derived.

Indeed, there is an algorithm that does this. A graph, directed or undirected, is transformed into an undirected graph on which the relevant computations can be performed. This is called the junction tree algorithm.

I'll derive the algorithm in the context of graphical models (probability distributions). The reader should bear in mind that a more general approach shows how the Junction Tree algorithm is also appropriate for forming a structure for which the Generalised Distributive Law of the previous chapter is guaranteed to work for a graph of any structure, singly or multiply-connected.

13.1 Absorption and Marginal Consistency

As we saw in chapter(12), any singly connected graph can be represented in terms of a product of clique marginal distributions, divided by the product of the separator distributions. This can be proved by induction. Furthermore, we can begin with a cluster potential assignment, and modify the cluster potentials so that, at the end of the calculation, the modified cluster potentials contain the marginals distributions, and similarly the separators are marginal distributions.

An alternative exposition comes from the following consistency argument, whose generaliation will lead to the Junction Tree Algorithm (JTA).

Consistent link

Consider a cluster representation with neighbouring clusters v and w, sharing the variables s in common. If our aim is that the JTA modifies the potentials such that the marginal of the distribution $p(w)$ is given by the (modified) potential $\Psi(w)$, then[1]

$$p(s) = \sum_{w \backslash s} \Psi(w)$$

[1] Note that, in the beginning, the assignment of the cluster potentials does not satisfy the consistency requirement. The aim is to find an algorithm that modifies them so that ultimately consistency is achieved.

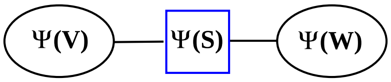
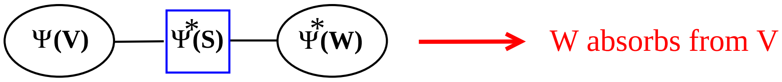


Figure 13.1: In absorption, we consider two neighbouring cliques in the cluster graph, which contain the potentials $\Psi(V)$ and $\Psi(W)$, with intersection variables $S \equiv V \cap W$.



Similarly,

$$p(s) = \sum_{v \backslash s} \Psi\left(v\right)$$

This then requires

$$\sum_{w \backslash s} \Psi\left(w\right) = p(s) = \sum_{v \backslash s} \Psi\left(v\right),$$

We identify the (modified) separator potential $\Psi\left(s\right) = p(s)$.

Imagine that by some process, we have managed to achieve consistency, but that now some new evidence changes $\Psi\left(v\right)$ to $\Psi^*\left(v\right)$ (this is achieved by clamping one of the variables in v to a particular state). In order that the link remains consistent, we need to change $\Psi\left(w\right)$ and $\Psi\left(s\right)$ in order to satisfy

$$\sum_{w \backslash s} \Psi^*\left(w\right) = \Psi^*\left(s\right) = \sum_{v \backslash s} \Psi^*\left(v\right).$$

Absorption

Let v and w be neighbours in a cluster tree, let s be their separator, and let $\Psi^*\left(v\right)$, $\Psi\left(w\right)$ and $\Psi\left(s\right)$ be their potentials. Absorption replaces the tables $\Psi^*\left(s\right)$ and $\Psi^*\left(w\right)$ with

$$\Psi^*\left(s\right) = \sum_{v \backslash s} \Psi\left(v\right)$$

$$\Psi^*\left(w\right) = \Psi\left(w\right)\frac{\Psi^*\left(s\right)}{\Psi\left(s\right)}$$

The idea behind this definition is that, under the update of the table for v, the table for the separator s and neighbour w are updated such that the link remains consistent. To see this consider

$$\sum_{w \backslash s} \Psi^*\left(w\right) = \sum_{w \backslash s} \Psi\left(w\right)\frac{\Psi^*\left(s\right)}{\Psi\left(s\right)} = \frac{\Psi^*\left(s\right)}{\Psi^*\left(s\right)}\sum_{w \backslash s} \Psi\left(w\right) = \frac{\Psi^*\left(s\right)}{\Psi\left(s\right)}\Psi\left(s\right) = \Psi^*\left(s\right) = \sum_{v \backslash s} \Psi^*\left(v\right)$$

Absorption passes a "message" from one node to another:

Note that if $\Psi(s)$ can be zero, then we need also $\Psi(w)$ to be zero when $\Psi(s)$ is zero for this procedure to be well defined. In this case, the potential takes the value unity at that point. (This requirement is on $\Psi(w)$ and not $\Psi^*(s)$ since we are considering whether or not it is possible to transmit the information through the current state of the link). We say that a link is supportive if it allows absorption in both directions (that is $\Psi(v)$ and $\Psi(w)$ will both be zero when $\Psi(s)$ is zero). Note that supportiveness is preserved under absorption.

Invariance of Cluster Tree under Absorption

Let T be a supportive cluster tree. Then the product of all cluster potentials divided by the product of all separator potentials is invariant under absorption.

Proof: When w absorbs v though the separator s only the potentials of w and s are changed. It is enough therefore to show that the fraction of the w and s tables is unchanged. We have

$$\frac{\Psi^*(w)}{\Psi^*(s)} = \frac{\Psi(w)\frac{\Psi^*(s)}{\Psi(s)}}{\Psi^*(s)} = \frac{\Psi(w)}{\Psi(s)}$$

So that if we start with a BN over U, construct a corresponding cluster tree T, and then perform a series of absorptions, then T remains a representation of $p(U)$ and is given by the product of all cluster potentials divided by the product of all separator potentials.

Note how this consistency argument has reproduced the Hugin form of Belief Propagation from chapter(12).

Propagation on Cluster Trees

Having defined the local message propagation approach, we need to define an update ordering for absorption. In general, a node V can send exactly one message to a neighbour W, and it may only be sent when V has received a message from each of its other neighbours. We continue this sequence of absorptions until a message has been passed in both directions along every link. See, for example fig(13.1). Note that the message passing scheme is generally not unique.

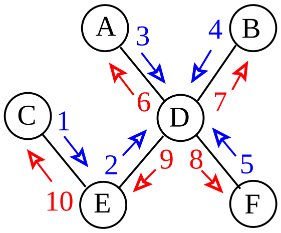


Figure 13.2: An example of the updating procedure for Hugin message passing.

The Hugin scheme is slightly advantageous over the standard Belief Propagation scheme since one doesn't need to take the products of all the messages at junctions. However, it is not as readily parallisable as the BP scheme, since the messages are not independent.

What we would like to do for a general distribution is to define a potential representation of the graph such that, coupled with a suitable algorithm to modify these potentials, the effect will be, as above, that the marginals of individual or groups (in fact cliques) can be read off directly from the modified potentials. This is the idea of the junction tree algorithm.

Comment : When the original distribution p is singly-connected, one can show that there is a cluster potential representation that is also singly-connected (actually, it will also be guaranteed to satisfy the running-intersection property so that locally consistent links will propagate to ensure global consistency). In that case, Hugin Belief Propagation may be performed. However, in the more general case of multiply-connected graphs, it may not be clear that an appropriate cluster representation exists (apart from trivial ones which essentially put all the variables into one cluster). How to construct efficient and suitable cluster-trees is at the heart of the JTA.

13.2 Junction Trees

Let T be a cluster tree over U and let a be a variable in U and suppose that a is an element of the nodes v and w. If T is consistent, we would expect that $\sum_{v \setminus a} \Psi(v) = \sum_{w \setminus a} \Psi(w)$. Certainly this is true it v and w are neighbours, but otherwise there is no guarantee.

Global Consistency

We say that a consistent cluster tree is globally consistent if for any nodes v and w with intersection I we have

$$\sum_{v \setminus I} \Psi(v) = \sum_{w \setminus I} \Psi(w)$$

Junction Tree
Running Intersection

A cluster tree is a junction tree if, for each pair of nodes, v and w, all nodes on the path between v and w contain the intersection $v \cap w$. This is also called the running intersection property.

Local = Global consistency

From this definition, it is clear that, in a consistent junction tree, the local consistency will be passed on to any neighbours. That is, a consistent junction tree is globally consistent.

Marginals

Let T be a consistent junction tree over U, and let $\Psi(U)$ be the product of all potentials divided by the product of all separator potentials. Let v be a node with potential $\Psi(v)$. Then

$$\Psi(v) = \sum_{U \setminus v} \Psi(U) = p(v)$$

To gain some intuition about the meaning of this theorem, consider the junction tree in fig(13.3). After a full round of message passing on this tree, each link is consistent, and the product of the potentials divided by the product of the separator potentials is just the original distribution itself. Imagine that we are interested in calculating the marginal for the node ABC. That requires summing over all the other variables, D, E, F, G, H. If we consider summing over H then, because the link is consistent,

$$\sum_h \Psi(eh) = \Psi(e)$$

so that the ratio $\sum_h \frac{\Psi(eh)}{\Psi(e)}$ is unity, so that the effect of summing over node H is that the link between EH and DCE can be removed, along with the separator.

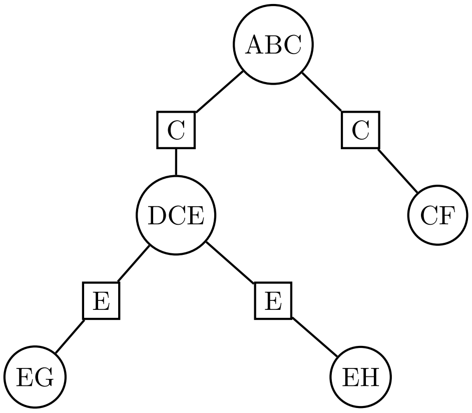


Figure 13.3: A junction tree. This satisfies the running intersection property that for any two nodes which contain a variable a, the path linking the two nodes also contains the variable a.

The same happens for the link between node EG and DCE, and also for CF to ABC. The only nodes remaining are now DCE and ABC and their separator C, which have so far been unaffected by the summations. We still need to sum out over D and E. Again, because the link is consistent,

$$\sum_{de} \Psi\left(d,c,e\right) = \Psi\left(c\right)$$

so that the ratio $\sum_{de} \frac{\Psi(d,c,e)}{\Psi(c)} = 1$. The result of the summation of all variables not in ABC therefore produces unity for the cliques and their separators, and the summed potential representation reduces simply to the potential $\Psi\left(a,b,c\right)$ which is the marginal $p(a,b,c)$. It is clear that a similar effect will happen for other nodes. Formally, one can prove this using induction.

We can then obtain the marginals for individual variables by simple brute force summation over the other variables in that potential. In the case that the number of variables in each node is small, this will not give rise to any computational difficulties. However, since the complexity is exponential in the clique size of the Junction Tree, it is prudent to construct the Junction Tree to have minimal clique sizes. Although, for a general graph, this is itself computationally difficult, there exist efficient heuristics for this task.

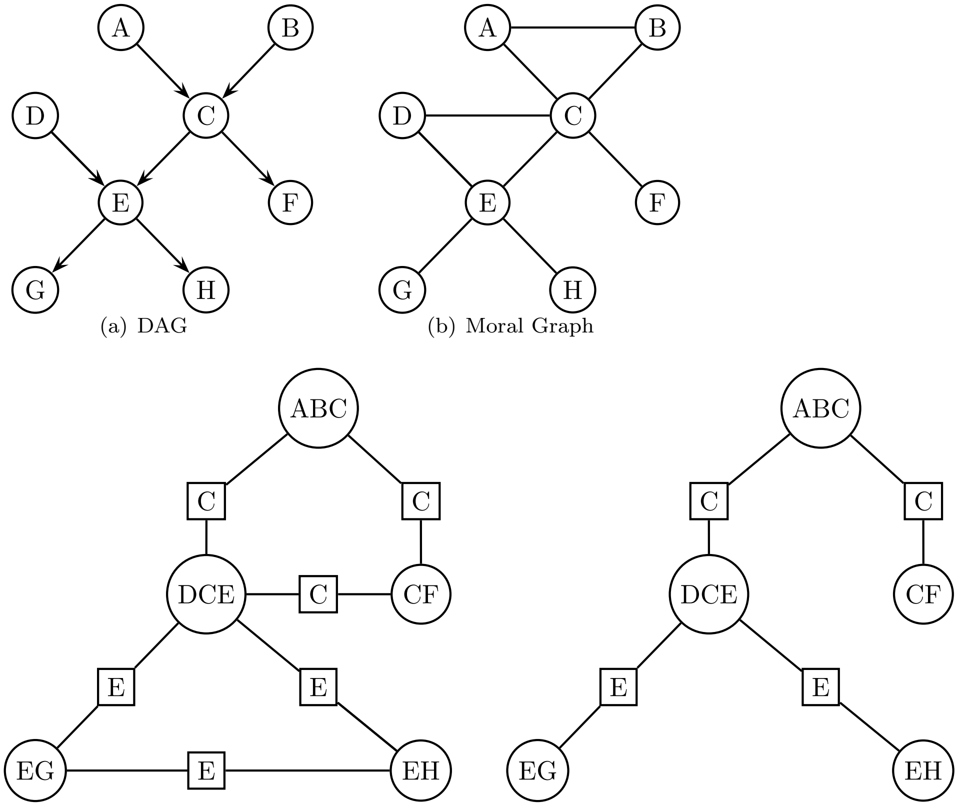
13.3 Constructing Junction Trees for Singly-Connected Distributions

For directed distributions, an initial step is required, which is not required in the case of undirected graphs.

Moral Graph

When we construct a cluster tree corresponding to a DAG, then for all variables a there must be a cluster v containing $\mathrm{pa}\left(a\right) \cup a$. We can illustrate this on a graph by having a link between any pair of variables which must appear in the same cluster. This means that we take the DAG, add a link between any pair of variables with a common child, and drop the direction of the original links. The resulting graph is the moral graph. From the moral graph you can find the clusters, namely the cliques in the graph.

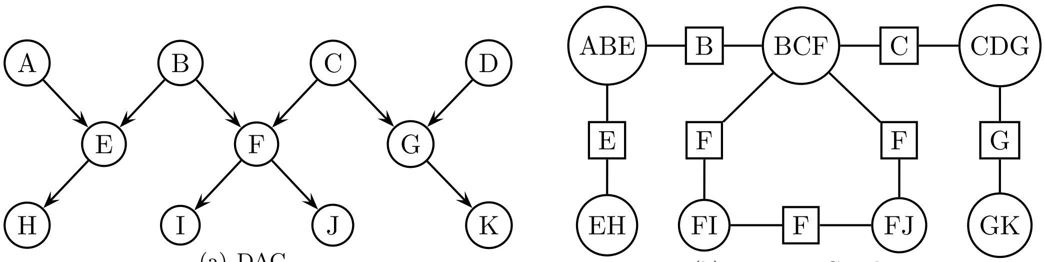
Then for both directed and undirected graphs, we may continue.



(a) DAG (b) Moral Graph

(c) Junction Graph (d) Junction tree

Figure 13.4: Construction of a junction tree for a singly connected DAG.



(a) DAG (b) Junction Graph

Figure 13.5: (a) A singly connected graph and (b) its junction graph. By removing any of the links in (b) with separator F you get a junction tree.

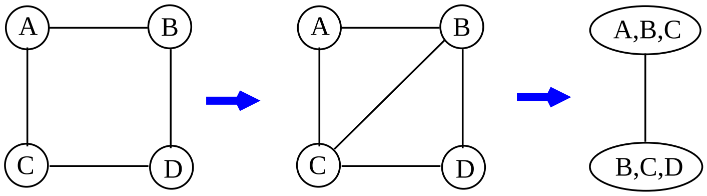


Figure 13.6: If we were to form a clique graph from the graph on the left, this would not satisfy the running intersection property, namely that if a node appears in two cliques, it appears everywhere on the path between the cliques. By introducing the extra link (middle picture), this forms larger cliques, of size three. The resulting clique graph (right) does satisfy the running intersection property (separator set not shown). Hence it is clear that loops of length four or more certainly require the addition of such chordal links to ensure the running intersection property in the resulting clique graph. It turns out that adding a chord in for all loops of length four or more is sufficient to ensure the running intersection property for any resulting clique graph.

Junction Graph and Tree

Then, between any two clusters with a non-empty intersection add a link with the intersection as the separator. The resulting graph is called a junction graph. All separators consist of a single variable, and if the junction graph contains loops, then all separators on the loop contain the same variable. Therefore any of the links can be removed to break the loop, and by removing links until you have a tree, you get a junction tree.

Consider the graph in fig(13.5a). Following the above procedure, we get the junction graph fig(13.5b). By breaking the loop $BCF - -F - -FI - -F - -FJ - -F - -BCF$ anywhere we obtain a junction tree.

The previous section showed how to construct a JT for a singly-connected graph. If we attempt to do this for a multiply connected (loopy) graph, we find that the above procedure generally does not work since the resulting graph will not necessarily satisfy the running intersection property. The idea is to grow larger clusters, such that these the resulting graph does satisfy the running intersection property. Clearly, a trivial solution would be to include all the variables in the graph in one cluster, and this will complete our requirements. However, of course, this does not help in finding an efficient algorithm for computing marginals. What we need is a sufficient approach that will guarantee that we can always form a junction tree from the resulting junction graph. This operation is called triangulation, and it generally increases the minimum clique size, sometimes substantially.

13.4 Junction Trees for Multiply-Connected Distributions

When there are loops, a variable elimination approach will, in general, change the structure of the remaining graph. To see this, consider the following distribution shown in fig(13.8),

$$p(a, b, c, d) = \phi(a, b)\phi(b, c)\phi(c, d)\phi(d, a)$$

Let's try to make a cluster style representation as before. We clearly here have a choice about which variable first to marginalise over. Let's choose d:

$$p(a, b, c) = \phi(a, b)\phi(b, c)\sum_d \phi(c, d)\phi(d, a)$$

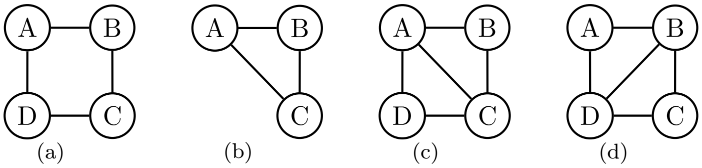


Figure 13.7: (a) An undirected graph with a loop. (b) Eliminating node D adds a link between A and C in the subgraph. (c) The induced representation for the graph in (a). (d) An alternative equivalent induced representation.

The remaining subgraph therefore has an extra connection between a and b.

$$p(a,b,c,d) = \frac{p(a,b,c)}{\sum_d \phi(c,d)\phi(d,a)}\phi(c,d)\phi(d,a)$$

Let's try to replace the numerator terms with probabilties. We can do this by considering

$$p(a,c,d) = \phi(c,d)\phi(d,a)\sum_b \phi(a,b)\phi(b,c)$$

Plugging this into the above equation, we have

$$p(a,b,c,d) = \frac{p(a,b,c)p(a,c,d)}{\sum_d \phi(c,d)\phi(d,a)\sum_b \phi(a,b)\phi(b,c)}$$

We recognise that the denominator is simply $p(a,c)$, hence

$$p(a,b,c,d) = \frac{p(a,b,c)p(a,c,d)}{p(a,c)}.$$

Hence, if we were to form a cluster graph based on products of cliques divided by products of separators, we could use an *induced* representation fig(13.7c). If we take the induced representation, and write down the cliques ABC and ACD, divided by the separator AC, this will form a correct cluster graph.

Alternatively, we could have marginalised over variables a and c, and ended up with the equivalant representation fig(13.7d).

Let's look at a slightly more complex loopy undirected distribution depicted in fig(13.8),

$$p = \phi(a,b)\phi(b,c)\phi(c,d)\phi(d,e)\phi(e,f)\phi(a,f)$$

As above, there are many different representations depending on which variables

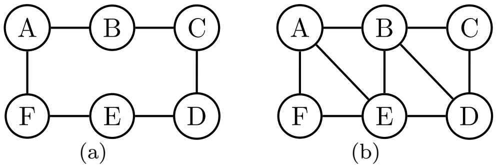
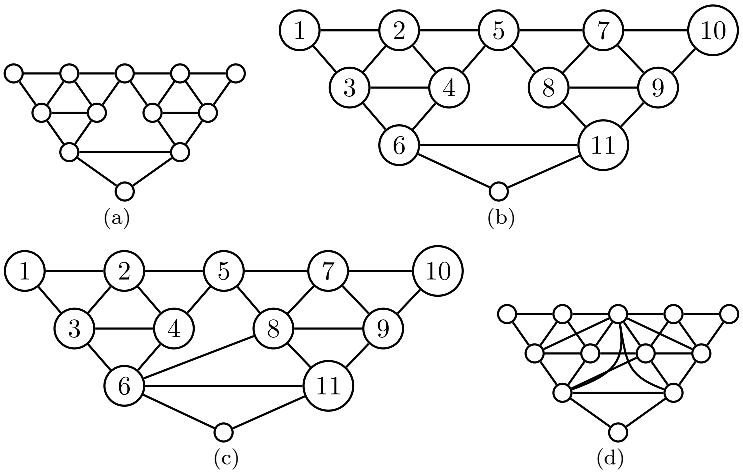


Figure 13.8: (a) An undirected graph with a loop. (b) An induced representation.



(a) (b) (c) (d)

Figure 13.9: (a) An undirected graph which is not triangulated. (b) We start the algorithm, labeling nodes until we reach node 11. This has neighbours 6 and 8 that are not adjacent. (c) We can correct this then by adding a link between 6 and 8, and restarting the algorithm. (d) The reader may check that this is a correct triangulation.

we decide to eliminate. However, the reader may convince herself that one such induced representation is given by fig(13.8b).

Generally, the result from variable elimination and re-representation in terms of the induced graph is that a link between any two variables on a loop (of length 4 or more) which do not have a chord is added. This is called *triangulation*. Any triangulated graph can be written in terms of the product of marginals divided by the product of separators.

Armed with this new induced representation, we can carry out a message propagation scheme as before.

13.5 Triangulation Algorithms

Formally, we need define the triangulation operations as follows:

Chord This is a link joining two non-consecutive vertices of a loop.

Triangulation An undirected graph is triangulated if every loop of length 4 or more has a chord.

The importance of this definition derives from the following theorem.

Triangulated= ∃ Junction Tree An undirected graph is triangulated if and only if its junction graph has a junction tree.

Alternative algorithm : Maximum Cardinality Checking

The following is an algorithm that terminates with success if and only if the graph is triangulated[21]. It processes each node and the time to process a node is quadratic in the number of adjacent nodes
(see `http://www.cs.wisc.edu/∼dpage/cs731/`).

```
Choose any node in the graph and label it 1
For $i = 2$ to $n$
    • Choose the node with the most labeled neighbours and label it $i$.
    • If any two labeled neighbours of $i$ are not adjacent to each other, FAIL.
end
SUCCEED
```

Where there is more than one node with the most labeled neighbours, the tie may be broken arbitrarily.

This algorithm gives us a way to make a triangulation – we simply add a link between the two neighbours that caused the algorithm to FAIL, and then restart the algorithm.

Comment : This is one algorithm for performing triangulation. Since the complexity of inference will scale exponentially (for discrete variables) with the size of the cliques in the resulting triangulated graph, clearly it is of some interest to find a triangulated graph with small clique sizes. However, finding the smallest possible triangulated graph, in this sense, is an NP-hard problem[]. The above algorithm is one of a number which are believed to be generically reasonable, although there may be cases where an alternative algorithm may be considerably more efficient.

13.6 Finding a JT from a Triangulated Graph

Above, we claimed that, provided that we have a carried out triangulation, then any Junction Tree consistent with the triangulation is sufficient to ensure that propagation on the JT will produce the correct marginals. One missing piece in this jigsaw is how to form a JT from a triangulated graph. This can be achieved from the following Theorem

Theorem 13.6.1 *Any maximal weight spanning tree is a junction tree*

The weight of a tree is defined to be the sum of all the separator weights of the tree, where the separator weight is defined as the number of variables in the separator.

A simple algorithm to find the spanning tree with maximal weight is as follows. Start by picking the edge with the largest weight, and add this to the edge set. Then pick the next candidate edge which has the largest weight and add this to the edge set – if this results in an edge set with cycles, then reject the candidate edge, and find the next largest edge weight.

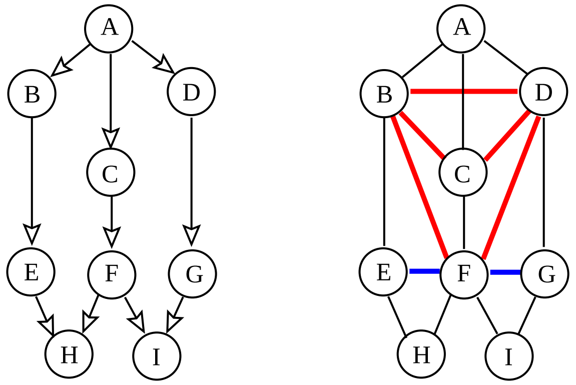
Note that there may be many maximal weight spanning trees. This algorithm provides one.

13.7 The Junction Tree Algorithm

We now have all the material we need for inference in multiply connected graphs. We need to do the following steps :

Moralisation This is required only for directed distributions.

Triangulation From the undirected (moralised) representation, ensure that every loop of length 4 or more has a chord.



(a) Original Graph

(b) Moralised and Triangulated

Figure 13.10: Example of the JTA. In (a) is the original loopy graph. (b) The moralisation links are between nodes E and F and between nodes F and G. The other additional links come from triangulation. The clique size of the resulting clique tree (not shown) is four.

Form the Junction Tree

Form a Junction Tree by forming a cluster representation from cliques of the triangulated graphs, removing any unnecessary links in a loop on the cluster graph. Algorithmically, this can be achieved by finding a tree with maximal spanning weight.

Potential Assignment

Assign the potentials to the cliques on the Junction Tree and assign the separator potentials on the JT to unity.

Message Propagation

Then carry out the absorption procedure until updates have been passed along both directions of every link on the JT.

Then the clique marginals can be read off from the JT. An example is given in fig(13.10).

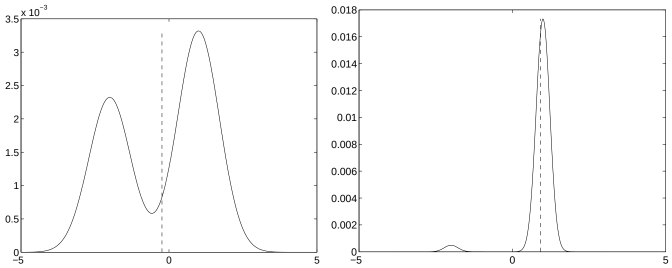
Comments on the JTA

There are some interesting points about the JTA. It provides an upper bound on the computation required to calculate marginals in the graph. This means that there may indeed exist more efficient algorithms in particular cases, although generally it is believed that there cannot be much more efficient approaches than the JTA since every other approach must perform a triangulation[22, 23].

However, there are, in general, many different ways to carry out the triangulation step. Ideally, we would like to find a triangularised graph which has minimal clique size. However, it can be shown to be a hard-computation problem (NP-hard) to find the most efficient triangulation. In practice, the triangulation algorithms used are somewhat heuristic, and chosen to provide reasonable, but clearly not optimal, performance.

13.7.1 Finding the Most Likely State

Previously, we have mainly concentrated on finding marginal probabilities, $p(x_i)$. However, another natural question to ask is what is the most likely state of the



(a) $p(x)$ (b) $p^*(x)$

Figure 13.11: $p^*(x) \propto p(x)^{10}$. In both figures the vertical dashed line indicates (on the x-axis the mean value for x. Note how p^* becomes much more peaked around its most probable valuem, and how the mean value in p^* shifts to be close to the most likely value. In the limit $p^*(x) \propto (p(x))^\beta$,$\beta \to \infty$, then the mean of the distribution p^* tends to the most-likely value.

distribution? There is a simple trick which will enable us to convert the JTA to enable us to answer this[2].

In general, a probability distribution may be written as

$$p = \frac{1}{Z}\prod_c \phi(x_c)$$

where $\phi(x_c)$ is the potential for cluster c. Consider a modified distribution in which we wish to re-weight the states, making the higher probability states exponentially more likely than lower probability states. This can be achieved by defining

$$p^* = \frac{1}{Z^\beta}\prod_{x_c} \phi^\beta(x_c)$$

where β is a very large positive quantity. This makes the distribution p^* very peaked around the most-likely value of p, see fig(13.11).

In the JTA, we need to carry out summations over states. However, in the limit $\beta \to \infty$ it is clear that only the most-likely state will contribute, and hence that the summation operation can be replaced by a maximisation operation in the definition of absorption. The algorithm thus proceeds as normal, replacing the summations with maximisations, until the final stage, whereby from the table one reads off $\underset{x_c}{\operatorname{argmax}}\, \phi(x_c)$ for the variables in the modified final potential on cluster c to find the most likely state.

A simple example of the JTA

Consider running the JTA on the simple graph

$$p(a, b, c) = p(a|b)p(b|c)p(c)$$

[2] As with the comments at the beginning of the chapter, the reader should bear in mind that the Generalised Distributive Law can be extended to the loopy case by using the updating equations on the Junction Tree. In this sense, any operations within the semiring algebra are admissible.

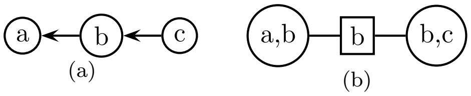


Figure 13.12: (a) A belief network. (b) JTA for the network.

There are three questions we are interested in (i) What is p(b)? (ii)? What is $p(b|a = 1, c = 1)$ (iii) What is the likelihood of the evidence $p(a = 1, c = 1)$?

For this simple graph, the moralisation and triangulation steps are trivial, and the JTA is given immediately by fig(13.12b). A valid assignment is $\Psi(a, b) = p(a|b)$, $\Psi(b) = 1$, $\Psi(b, c) = p(b|c)p(c)$.

First let's absorb from (a, b) through the separator b to (b, c):

Finding a marginal $p(b)$

First we just run the JTA as usual.

- The new separator is given by $\Psi^*(b) = \sum_a \Psi(a, b) = \sum_a p(a|b) = 1$.
- The new potential on (b, c) is given by $\Psi^*(b, c) = \frac{\Psi(b,c)\Psi^*(b)}{\Psi(b)} = \frac{p(b|c)p(b)\times 1}{1}$.
- The new separator is given by $\Psi^{**}(b) = \sum_c \Psi^*(b, c) = \sum_c p(b|c)p(c)$.
- The new potential on (a, b) is given by $\Psi^*(a, b) = \frac{\Psi(a,b)\Psi^{**}(b)}{\Psi^*(b)} = \frac{p(a|b)\sum_c p(b|c)p(c)}{1}$. This is therefore indeed equal to the marginal since $\sum_c p(a, b, c) = p(a, b)$.

Also, the new separator $\Psi^{**}(b)$ contains the marginal $p(b)$ since $\Psi^{**}(b) = \sum_c p(b|c)p(c) = \sum_c p(b, c) = p(b)$.

Finding a conditional marginal $p(b|a = 1, c = 1)$

First we clamp the evidential variables in their states. Then we claim that the effect of running the JTA is to produce on the cliques, the joint marginals $p(a = 1, b, c = 1)$, $p(a = 1, b, c = 1)$ and $p(a = 1, b, c = 1)$ for the final potentials on the two cliques and their separator. We demonstrate this below:

- In general, the new separator is given by $\Psi^*(b) = \sum_a \Psi(a, b) = \sum_a p(a|b) = 1$. However, since a is clamped in state $a = 1$, then the summation is not carried out over a, and we have instead $\Psi^*(b) = p(a = 1|b)$.
- The new potential on the (b, c) clique is given by $\Psi^*(b, c) = \frac{\Psi(b,c)\Psi^*(b)}{\Psi(b)} = \frac{p(b|c=1)p(c=1)p(a=1|b)}{1}$.
- The new separator is normally given by $\Psi^{**}(b) = \sum_c \Psi^*(b, c) = \sum_c p(b|c)p(c)$. However, since c is clamped in state 1, we have instead $\Psi^{**}(b) = p(b|c = 1)p(c = 1)p(a = 1|b)$
- The new potential on (a, b) is given by $\Psi^*(a, b) = \frac{\Psi(a,b)\Psi^{**}(b)}{\Psi^*(b)} = \frac{p(a=1|b)p(b|c=1)p(c=1)p(a=1|b)}{p(a=1|b)} = p(a = 1|b)p(b|c = 1)p(c = 1)$.

Hence, here in this special case, all the cliques contain the joint distribution $p(a = 1, b, c = 1)$.

In general, the effect of clamping a set of variables V in their evidential states, and running the JTA is that, for a clique i which contains the set of non-evidential variables H^i, the potentials after the end of the JTA contains the marginal $p(H^i, V)$.

Then calculating the conditional marginal $p(b|a = 1, c = 1)$ is a simple matter since $p(b|a = 1, c = 1) \propto p(a = 1, b, c = 1)$, where the proportionality is determined by the normalisation constraint.

Finding the likelihood $p(a = 1, c = 1)$

By the above procedure, the effect of clamping the variables in their evidential states and running the JTA produces the joint marginals, such as $\Psi^* (a, b) = p(a = 1, b, c = 1)$. Then calculating the likelihood is easy since we just sum out over the non-evidential variables of any converged potential : $p(a = 1, c = 1) = \sum_b \Psi^* (a, b) = \sum_b p(a = 1, b, c = 1)$.

Whilst we have demonstrated these results only on such a simple graph, the same story holds in the general case. Hence calculating conditional marginals and likelihoods can be obtained in exactly the same way. The main thing to remember is that clamping the variables in evidential states means that the *joint* distribution on the non-evidential variables in a clique with all the evidential variables clamped in their evidential states is what is found a the end of the JTA. From this conditionals and the likelihood are straightforward to calculate.

13.8 Problems

Exercise 42 *Consider the following undirected graphical model:*

$$p(x_1, x_2, x_3, x_4) = \phi(x_1, x_2)\phi(x_2, x_3)\phi(x_3, x_4)$$

1. *Draw a clique graph that represents this distribution, and indicate the separators on the graph.*
2. *Write down an alternative formula for the distribution $p(x_1, x_2, x_3, x_4)$ in terms of the marginal probabilities $p(x_1, x_2)$, $p(x_2, x_3)$, $p(x_3, x_4)$, $p(x_2)$, $p(x_3)$*

Exercise 43 *Consider the distribution*

$$p(x_1, x_2, x_3, x_4) = \phi(x_1, x_2)\phi(x_2, x_3)\phi(x_3, x_4)\phi(x_4, x_1)$$

1. *Write down a Junction Tree for the above graph.*
2. *Carry out the absorption procedure and demonstrate that this gives the correct value for the marginal $p(x_1)$.*

13.9 Solutions

14 Variational Learning and EM

14.1 Maximum Likelihood

Imagine that we have some training data, $X = x^1, \ldots, x^P$. Our model $p(x|\Theta)$ has some parameters Θ. These could result from a parameterisation of the conditional probability tables in a Belief Network, or indeed could be the table entries themselves. In general, they are simply some parameters of a probability distribution. How can we use the training data to determine the best setting of the parameters Θ?

Bayes rule provides an elegant and immedidate response to this question. In Bayesian Learning, we would consider the posterior distribution,

$$p(\Theta|V) \propto p(V|\Theta)p(\Theta).$$

What this says is that, in general, there is no such 'optimal' single setting of the parameters, rather that there is a *distribution* of parameter values, where each value of Θ has a weight according to the above distribution. That is, each value for Θ is assessed by how likely it is to generate the data, but also is moderated by any *prior* beliefs $p(\Theta)$. Note that, in a purely probabilistic sense, the specification of prior beliefs $p(\Theta)$ is unavoidable, and there is no sense in 'assumption free' determination of the parameters.

However, dealing with such posterior distributions is often computationally too ambitious, and the simpler Maximum A Posteriori (MAP) solution is preferred,

$$\Theta^{MAP} = \arg\max_{\Theta} p(X|\Theta)p(\Theta)$$

in which a single 'best' estimate for Θ is chosen. If the user does not feel able to specify any prior preference for Θ (a so-called "flat" prior $p(\Theta) = const$), the parameters are given by Maximum Likelihood

$$\Theta^{ML} = \arg\max_{\Theta} p(V|\Theta)$$

which simply says that we set the parameter Θ to that value for which the observed data was most likely to have been generated.

Belief Nets

Previously, we have emphasised two aspects of graphical models – the independence assumptions, and the conditional probability tables. Here we will assume that we are happy about the independence assumptions, but that we do not know the values of the conditional probability tables. We address the issue of how to learn values for the CPTs given a set of data. This brings the subject much closer to the ethos of machine learning.

Consider the following model of the relationship between exposure to Asbestos (A), being a smoker (S) and the incidence of lung cancer (C)

$$p(A, S, C) = p(C|A, S)p(A)p(S)$$

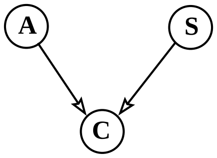


Figure 14.1: A simple model for the relationship between lung Cancer, Asbestos exposure and Smoking.

which is depicted in fig(14.1). Here we have made the assumption that Cancer is dependent on both exposure to Asbestos and being a Smoker, but that there is no direct relationship between Smoking and exposure to Asbestos. This is the kind of assumption that we may be able to elicit from experts such as doctors who have good intuition/understanding of the relationship between variables.

Furthermore, we assume that we have a list of individuals characteristics in the population, where each row represents a training example. This is perhaps taken from hospital records or a general survey of the population.

A	S	C
1	1	1
1	0	0
0	1	1
0	1	0
1	1	1
0	0	0
1	0	1

Figure 14.2: A database containing information about the Asbestos exposure (1 signifies exposure), being a Smoker (1 signifies the individual is a smoker), and lung Cancer (1 signifies the individual has lung Cancer). Each row contains the information for an individual, so that there are 7 individuals in the database.

Intuitive Table Settings

Looking at instances where $A = 0, S = 0$, we find always $C = 0$, and hence $p(C = 1|A = 0, S = 0) = 0$. Similarly, we can count other cases to form a CPT table. Counting the instances of $A = 1$, we find $p(A = 1) = 4/7$, and similarly, $p(S = 1) = 4/7$. These three CPTs then complete the full distribution specification.

A	S	$p(C = 1\|A, S)$
0	0	0
0	1	0.5
1	0	0.5
1	1	1

Maximum Likelihood Learning

Actually, what we did intuitively, by counting the relative number of occurrences, corresponds mathematically to maximum likelihood learning.

For a set of P observations (training data), $X = x^1, \ldots, x^P$, and independently gathered observations, the log likelihood of X is

$$\log p(X) = \sum_{\mu=1}^{P} \log p(x^\mu)$$

For DAGs, the factorisation enables us to separate the CPT terms:

$$\log p(X) = \sum_{\mu=1}^{P} \sum_{i=1}^{n} \log p\left(x_i^\mu | \mathrm{pa}\left(x_i^\mu\right)\right)$$

We want to learn the entries of the CPTs. For convenience, let $\mathrm{pa}\left(x_1\right) = \{x_2, x_3\}$, and say we want to find the CPT entry $p(x_1 = 1|x_2 = 1, x_3 = 0)$.

Counting the Occurrences

Naively, the number of times $p(x_1 = 1|x_2 = 1, x_3 = 0)$ occurs in the log likelihood is equal to $c(1, 1, 0)$, the number of such occurrences in the training set. However, since (by the normalisation constraint) $p(x_1 = 0|x_2 = 1, x_3 = 0) = 1 - p(x_1 = 1|x_2 = 1, x_3 = 0)$, the total contribution of $p(x_1 = 1|x_2 = 1, x_3 = 0)$ to the log likelihood is

$$c(1, 1, 0) \log p(x_1 = 1|x_2 = 1, x_3 = 0) + c(0, 1, 0) \log \left(1 - p(x_1 = 1|x_2 = 1, x_3 = 0)\right)$$

Differentiating wrt $p(x_1 = 1|x_2 = 1, x_3 = 0)$ gives

$$p(x_1 = 1|x_2 = 1, x_3 = 0) = \frac{c(x_1 = 1, x_2 = 1, x_3 = 0)}{c(x_1 = 1, x_2 = 1, x_3 = 0) + c(x_1 = 0, x_2 = 1, x_3 = 0)}$$

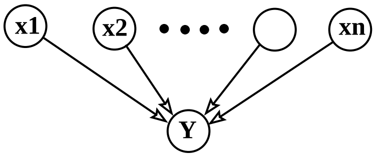
Corresponding to the previous intuitive counting procedure. Alernatively, we may write

$$p(x_1 = 1|x_2 = 1, x_3 = 0) = \frac{\sum_{\mu: x_1^\mu = 1, x_2^\mu = 1, x_3^\mu = 0} 1}{\sum_{\mu: x_1^\mu = 1, x_2^\mu = 1, x_3^\mu = 0} 1 + \sum_{\mu: x_1^\mu = 0, x_2^\mu = 1, x_3^\mu = 0} 1}$$

From the above example, it is clear that we can set values for the all the table entries. However, consider a smaller dataset:

A	S	C
1	1	1
1	0	0
0	1	1
0	1	0

According to the ML principal above, we will not be be able to determine entry $p(c|a = 0, s = 0)$, since there are no entries in the database which jointly contain the setting $a = 0$ and $s = 0$. In this case, we either need additional information, or assumptions about how to set the missing table entries. One approach that may lead to a fuller specification is to require that not only all the jointly observed training data should be maximally likely, but also that any marginal observations should also be maximally likely – that is, we restrict attention to a subset of the variabes, say here C alone, and require that the model is maximally likely to generate the observed statistics for the C variable alone. Since calculating the marginal likelihood $p(c)$ involves summing over all the states $\sum_{s,a} p(c|s, a)p(s)p(a)$, we obtain an objective function that contains at least the parameters $p(c|s = 0, a = 0)$. How to choose such marginals, and how to weight this requirement with the



joint ML requirement is an open question, but could potentially be related to standard generalisation issues. An interesting question is, even if we have enough data to specify all the table entries by ML, whether or not the learned model is consistent with all the marginals in the training data? It is left as an exercise for the reader to show that, in general, it is not the case that all the marginals from the learned model are consistent with the sample marginals. In general these can only be made consistent if there are no restrictions on the form of the probability distribution fitted – that is, no independence assumptions are made.

Conditional Probability Functions

Imagine that we have a node with n parents, in state $x = (x_1, \ldots, x_n)$. For binary variables, there are therefore 2^n entries in the CPT to specify for that node. This rapidly becomes infeasible, and we need to use a functional specification of the table. For example, a sigmoidal function

$$p(Y = 1|x) = 1/(1 + \exp(w \cdot x))$$

where we only need to specify the n-dimensional parameter w.

So, rather than use Maximum Likelihood to learn the entries of the CPTs directly, we instead learn the value of the parameter w. Since the number of parameters in w is small (n), then we have some hope that with a small number of training examples (say of the order of n), we can learn a reliable value for w.

14.1.1 Missing Data/Variables

Previously, we assumed that all the variables are 'visible', or 'evidential'. That is, each training data point specified that values of all the variables in the distribution. In many cases, we simply will not be able to directly measure certain variables, although we know that they are important in the modelling process. Such variables are 'hidden'. In cases where we have a training set, and sometimes the variables are specified, and in some cases not, then we have a 'missing data' problem. Both of these cases can be dealt with by using Maximum Likelihood – however, we calculate the likelihood for each training example on only those visible variables.

For example, consider two training examples, x^1 and x^2, in which x^1 is fully observed, but x^2 has an unobserved first component, i.e. $x^2 = (?, x_2^2, x_3^2, \ldots x_n^2)$. The log likelihood of the data is then

$$\log p(x_1^1, x_2^1, \ldots, x_n^1) + \log p(x_2^2, x_3^2, \ldots x_n^2)$$

Using the general BN structure, this would be

$$\sum_{i=1}^{n} \log p\left(x_i^1 | \mathrm{pa}\left(x_i^1\right)\right) + \log \sum_{x_1^2} p(x_1 | \mathrm{pa}\left(x_1^2\right)) \prod_{i>1} p\left(x_i^2 | \mathrm{pa}\left(x_i^2\right)\right)$$

The structure of the second term is a summation over the missing values for the variable missing for that example. If there are many missing datapoints, calculating the summations may be difficult. However, one can see that the log-likelihood remains a function of the tables, and one can optimise this as usual. However, this direct approach is rarely taken in practice. An alternative, general and more elegant approach to this problem, is given by the EM algorithm, as described below.

14.2 Variational Learning and Expectation Maximisation

For simplicity here, we will deal only with the case that certain variables are consistently unobserved in the training data set. The generalisation to the case that for some examples a certain variable is observed and in others not, is straightforward. We use the notation v for those components of the variables x that are 'visible' (i.e for which we have observed values), and h for those components of the variables x that are 'hidden' (i.e for which we do not have observed values). In ML learning, our aim is to maximise the likelihood of the observed/visible data, $V = \{v^\mu, \mu = 1, \dots, P\}$ with respect to any parameters Θ (CPTs or their parameters) of the distribution.

Calculating $p(V|\Theta)$

Assuming that the data instances are gathered independently,

$$\log p(V|\Theta) = \sum_{\mu=1}^{P} \log p(v^\mu|\Theta)$$

In the above, we could use marginalisation to calculate the likelihood of the visible data,

$$p(v^\mu|\Theta) = \sum_{h^\mu} p(v^\mu, h^\mu|\Theta)$$

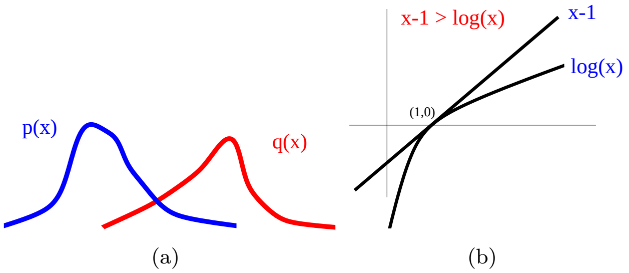
However, there are reasons why this may not be such a good idea. For example, there are so many hidden units that we cannot carry out the summation (Junction Tree Cliques are too large). Or the resulting log likelihood is difficult to optimise using standard approaches – the objective function is extremely complicated.

There exists a useful general procedure for learning with hidden units. Special cases of this approach include the Expectation Maximisation (EM) algorithm, Generalised EM (GEM) algorithms and (probably all) the other EM variants. In the machine learning literature, Neal and Hinton[24] made the connection between the traditional EM algorithm[], and the more general variational treatment. See [25] for a more standard exposition.

The variational EM has several potential positive aspects

- Can (often but not always) help deal with intractability
- Provides a rigorous lower bound on the likelihood
- May make larger parameter updates than gradient based approaches.

Before we can talk about the Variational Learning algorithm, which is a special variational technique based on the Kullback-Leibler divergence between two distributions, we need a digression into Information Theory.



(a) (b)

Figure 14.3: (a) The probability density functions for two different distributions $p(x)$ and $q(x)$. We would like to numerically characterise the difference between these distributions. (b) A simple linear bound on the logarithm enables us to define a useful distance measure between distributions (see text).

The Kullback Leibler Divergence $KL(q, p)$

The KL diveregence $KL(q, p)$ measures the "difference" between distributions q and p, fig(D.1a). In many ways, this is a natural measure to use, and is well motivated from Information theoretic arguments.

For two distributions $q(x)$ and $p(x)$, it is defined as

$$KL(q, p) \equiv \langle \log q(x) - \log p(x) \rangle_{q(x)}$$

where the notation $\langle f(x) \rangle_{r(x)}$ denotes average of the function $f(x)$ with respect to the distribution $r(x)$. For a continuous variable, this would be $\langle f(x) \rangle_{r(x)} = \int f(x)r(x)dx$, and for a discrete variable, we would have $\langle f(x) \rangle_{r(x)} = \sum_x f(x)r(x)$. The advantage of this notation is that much of the following holds independent of whether the variables are discrete or continuous.

$KL(q, p) \geq 0$ The KL divergence is always ≥ 0. To see this, consider the following simple linear bound on the function $\log(x)$ (see fig(D.1b)):

$$\log(x) \leq x - 1$$

Replacing x by $p(x)/q(x)$ in the above bound

$$\frac{p(x)}{q(x)} - 1 \geq \log \frac{p(x)}{q(x)} \Rightarrow p(x) - q(x) \geq q(x) \log p(x) - q(x) \log q(x)$$

Now integrate (or sum in the case of discrete variables) both sides. Using $\int p(x)dx = 1$, $\int q(x)dx = 1$, and rearranging gives

$$\int \{q(x) \log q(x) - q(x) \log p(x)\} \, dx \equiv \langle \log q(x) - \log p(x) \rangle_{q(x)} \geq 0$$

Furthermore, one can show that the KL divergence is zero if and only if the two distributions are exactly the same.

Bounding the Likelihood

The KL divergence enables us immediately to find a lower bound on the marginal likelihood for a single training example, $p(v)$. Consider

$$KL\left(q(h|v), p(h|v)\right) \equiv \langle \log q(h|v) - \log p(h|v) \rangle_{q(h|v)} \geq 0$$

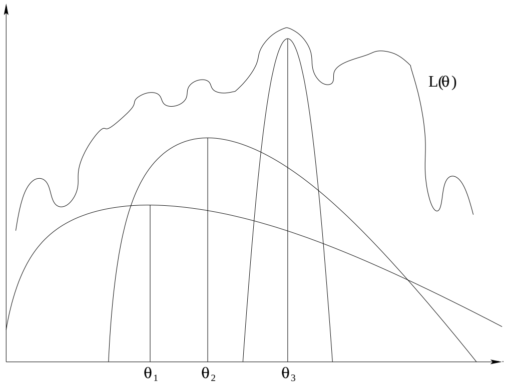


Figure 14.4: The idea in variational learning is to bound a possibly complex likelihood $L(\Theta)$ by a simpler function, for which the maximum is easy to find, here say finding Θ_1. Subsequently, a new lower bound on the likelihood is fitted, using the previous best value for the optimal Θ, and a new optimal bound value Θ_2 is found. This is repeated until convergence, each time pushing up the lower bound, and therefore hopefully improving our estimate of where the maximum of the likelihood is. This iterative procedure can often lead to the rapid finding of (locally) optimal values for the parameters.

Hence, using Bayes rule,

$$\langle \log q(h|v)\rangle_{q(h|v)} - \langle \log p(h,v)\rangle_{q(h|v)} + \log p(v) \geq 0$$

Rearranging, we therefore have the bound[1]

$$\log p(v) \geq \underbrace{- \langle \log q(h|v)\rangle_{q(h|v)}}_{Entropy} + \underbrace{\langle \log p(h,v)\rangle_{q(h|v)}}_{Energy}$$

Summing over the training data, we get the bound on the marginal likelihood

$$\log p(V|\Theta) \geq \underbrace{-\sum_{\mu=1}^{P} \langle \log q^\mu(h|v)\rangle_{q^\mu(h|v)}}_{Entropy} + \underbrace{\sum_{\mu=1}^{P} \langle \log p(h^\mu, v^\mu|\Theta)\rangle_{q^\mu(h|v)}}_{Energy}$$

This bound is exact (that is, it is equal to the log marginal likelihood) when we set $q^\mu(h|v) = p(h^\mu|v^\mu)$. Recalling that our aim is to find an algorithm that will adjust any parameters of p to maximize the likelihood, a reasonable thing to do would be a relaxed version, namely to maximize a lower bound on the likelihood. That is, to iteratively adjust the parameters Θ to push up the lower bound on the (marginal) likelihood, and in so doing hopefully push up the true (marginal) likelihood.

Variational Learning

Since the parameter Θ only occurs in the Energy term, this suggests that we can iteratively firstly set the optimal parameters Θ by optimising the Energy term

[1] This is analogous to the Mean Field bound on the partition function in statistical physics, and motivates the terminology 'energy' and 'entropy'.

(for fixed $q^{\mu}(h|v)$). And then, we can optimise (push up the lower bound) by finding a better set of fixed $q^{\mu}(h|v)$, by optimising with respect to the variational distributions $q^{\mu}(h|v)$:

1. Expectation (E) step : Choose a set of distributions

   $$q^{\mu}(h|v), \mu = 1 \dots P$$

   from a chosen class of distributions, for which each $q^{\mu}(h|v)$ minimises the KL divergence $KL(q^{\mu}(h|v), p(h^{\mu}|v^{\mu}))$.

2. Maximisation (M) step : Set

   $$\Theta \leftarrow \arg\max_{\Theta} \sum_{\mu=1}^{P} \left\langle \log p(h^{\mu}, v^{\mu}|\Theta) \right\rangle_{q^{\mu}(h|v)}$$

Iterate (1,2) until parameter convergence. Steps (1) and (2) are guaranteed to increase the lower bound on the likelihood.

The EM algorithm cannot decrease the likelihood

Whilst, by definition, the EM algorithm cannot decrease the bound on the likelihood, an important question is whether or not the iterations cannot decrease the likelihood itself.

Another way to rephrase our bound on the likelihood $\log p(v|\boldsymbol{\theta}') \geq LB(\boldsymbol{\theta}'|\boldsymbol{\theta})$ is as

$$\log p(v|\boldsymbol{\theta}') = LB(\boldsymbol{\theta}'|\boldsymbol{\theta}) + KL(p(h|v, \boldsymbol{\theta})|p(h|v, \boldsymbol{\theta}'))$$

That is, the KL divergence is simply the difference between the lower bound and the true likelihood. Similarly, we may write

$$\log p(v|\boldsymbol{\theta}) = LB(\boldsymbol{\theta}|\boldsymbol{\theta}) + \underbrace{KL(p(h|v, \boldsymbol{\theta})|p(h|v, \boldsymbol{\theta}))}_{zero}$$

Hence

$$\log p(v|\boldsymbol{\theta}') - \log p(v|\boldsymbol{\theta}) = \underbrace{LB(\boldsymbol{\theta}'|\boldsymbol{\theta}) - LB(\boldsymbol{\theta}|\boldsymbol{\theta})}_{\geq 0} + \underbrace{KL(p(h|v, \boldsymbol{\theta})|p(h|v, \boldsymbol{\theta}'))}_{\geq 0}$$

The first assertion is true since, by definition, we search for a $\boldsymbol{\theta}'$ which has a higher value for the bound than our starting value $\boldsymbol{\theta}$. The second assertion is trivially true by the property of the KL divergence. Hence we reach the important conclusion that the EM (or GEM/variational implementation), not only essentially increases the lower bound on the likelihood, but also increases the likelihood itself (or, at least, the EM cannot decrease these quantities).

EM Algorithm

Clearly, if we do not restrict the class of distributions that the q can take, the optimal choice is

$$q^{\mu}(h|v) = p(h^{\mu}|v^{\mu})$$

Using these $q's$, corresponds to the standard "EM" algorithm.

Intractable Energy

The algorithm assumes that we can calculate $\left\langle \log p(h^{\mu}, v^{\mu}|\Theta) \right\rangle_{q^{\mu}(h|v)}$. However, in

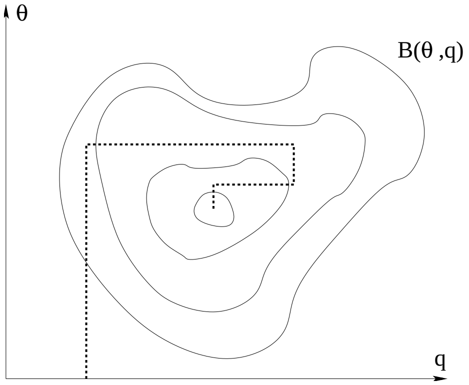


Figure 14.5: The EM approach is an axis aligned way to find a maximum of the lower bound $B(\theta, q)$. This proceeds by, for fixed q, finding the best parameters θ (the M-step), and then for fixed θ, finding the best distributions q (the E-step). Of course, any other optimisation procedure is valid, and indeed may result in faster convergence than this simple axis aligned approach. However, an advantage of the EM style is that is leads to a simple-to-implement-and-interpret algorithm.

general, it may be that we can only carry out the average over q for a very restricted class of distributions. For example, factorised distributions $q^\mu(h|v) = \prod_j q^\mu(h_j|v)$. Hence, in practice, we often choose a simpler class of distributions, $\mathcal{Q}$, e.g $\mathcal{Q}$ = factorised $q^\mu(h|v) = \prod_i q^\mu(h_i|v)$, which may make the averaging required for the energy simpler.

Determining the best distribution in the class

Imagine we parameterise our distribution class $\mathcal{Q}$ using a parameter θ_Q. We can find the best distribution in class $\mathcal{Q}$ by minimising the KL divergence between $q^\mu(h|v, \theta_Q)$ and $p(h^\mu|v^\mu, \Theta)$ numerically using a non-linear optimisation routine. Alternatively, one can assume a certain structured form for the q distribution, and learn the optimal factors of the distribution by free form functional calculus.

Using a class of simpler q distributions like this corresponds to a Generalised EM algorithm (GEM).

Application to Belief Networks

The previous variational learning theory is very general. To make things more concrete, we apply the previous theory to learning the CPTs in a BN in which certain variables are hidden. We first apply it to a very simple network.

$$p(a, c, s) = p(c|a, s)p(a)p(s)$$

Imagine, as in table fig(14.1), we have a set of data, but that we do not know the states of variable a. That is,

S	C
1	1
0	0
1	1
1	0
1	1
0	0
0	1

Firstly, let's assume that we have chosen some values for the distributions $q^\mu(a|c,s)$, e.g. $q^1(a=1|c=1,s=1)=0.6, q^2(a=1|c=0,s=0)=0.3, q^3(a=1|c^3=1,s=1)=0.7, q^4(a=1|c=0,s=1)=0.1\ldots$. Now we write down the Energy term:

$$E=\sum_{\mu=1}^{7}\left\langle \log p(c^\mu|a^\mu,s^\mu)+\log p(a^\mu)+\log p(s^\mu)\right\rangle_{q^\mu(a|c,s)}$$

$$E=\sum_{\mu=1}^{7}\left\{\left\langle \log p(c^\mu|a^\mu,s^\mu)\right\rangle_{q^\mu(a|c,s)}+\left\langle \log p(a^\mu)\right\rangle_{q^\mu(a|c,s)}+\log p(s^\mu)\right\}$$

Remember that our goal is to learn the CPTs $p(c|a,s)$ and $p(a)$ and $p(s)$.

Pleasingly, the final term is simply the log likelihood of the variable s, and $p(s)$ appears explicitly only in this term. Hence, the usual maximum likelihood rule applies, and $p(s=1)$ is simply given by the relative number of times that $s=1$ occurs in the database (hence $p(s=1)=4/7$, $p(s=0)=3/7$).

The parameter $p(a=1)$ occurs in the terms

$$\sum_\mu\left\{q^\mu(a=0|c,s)\log p(a=0)+q^\mu(a=1|c,s)\log p(a=1)\right\}$$

which, using the normalisation constraint is

$$\log p(a=0)\sum_\mu q^\mu(a=0|c,s)+\log(1-p(a=0))\sum_\mu q^\mu(a=1|c,s)$$

Differentiating with respect to $p(a=0)$ we get

$$p(a=0)=\frac{\sum_\mu q^\mu(a=0|c,s)}{\sum_{\mu=1} q^\mu(a=0|c,s)+\sum_\mu q^\mu(a=1|c,s)}$$

That is, whereas in the standard ML estimate, we would have the real counts of the data in the above formula, here they have been replaced with our guessed values $q^\mu(a=0|c,s)$ and $q^\mu(a=1|c,s)$.

A similar story holds for the more complex case of say $p(c=1|a=0,s=1)$. The contribution of this term to the Energy is

$$\sum_{\mu:c^\mu=1,s^\mu=1} q^\mu(a=0|c=1,s=1)\log p(c=1|a=0,s=1)$$
$$+\sum_{\mu:c^\mu=0,s^\mu=1} q^\mu(a=0|c=0,s=1))\log(1-p(c=1|a=0,s=1))$$

which is

$$\log p(c=1|a=0,s=1)\sum_{\mu:c^\mu=1,s^\mu=1} q^\mu(a=0|c=1,s=1)$$
$$+\log(1-p(c=1|a=0,s=1))\sum_{\mu:c^\mu=0,s^\mu=1} q^\mu(a=0|c=1,s=1)$$

Optimising with respect to $p(c=1|a=0,s=1)$ gives

$$p(c=1|a=0,s=1)=$$
$$\frac{\sum_{\mu:c^\mu=1,s^\mu=1} q^\mu(a=0|c=1,s=1)}{\sum_{\mu:c^\mu=1,s^\mu=1} q^\mu(a=0|c=1,s=1)+\sum_{\mu:c^\mu=0,s^\mu=1} q^\mu(a=0|c=0,s=1)}$$

Again, this has an intuitive relationship to ML for the complete data case, in which the missing data has been filled in by the assumed distributions q.

What about the parameters $q^\mu(a|c,s)$? If we use the standard EM algorithm, we should set these to

$$q^\mu(a|c,s) = p(a^\mu|c^\mu, s^\mu) \propto p(a^\mu, c^\mu, s^\mu)$$

$$q^\mu(a|c,s) \propto p(c^\mu|a^\mu, s^\mu)p(a^\mu)p(s^\mu)$$

where the current set of values for the p's have been assumed from the previous calculation. These two stages are then iterated : in the next step, we use these new values of $q^\mu(a|c,s)$ to calculate the next $p's$ etc. These equations will converge to a local optimum of the bound.

More general Belief Networks

The form of the energy term for belief networks is

$$\sum_\mu \langle \log p(h^\mu, v^\mu)\rangle_{q^\mu(h|v)} = \sum_\mu \sum_i \langle \log p(x_i^\mu|\mathrm{pa}\,(x_i^\mu))\rangle_{q^\mu(h|v)}$$

where each x_i is either clamped into a visible state, or is a hidden unit. Note that $p(x_i^\mu|\mathrm{pa}\,(x_i^\mu))$ is only a function of the variables $x_i \cup \mathrm{pa}\,(x_i)$, the family of node x_i and that, in general, some of these may be hidden. The hidden nodes of this family are $g_i^\mu \equiv x_i \cup \mathrm{pa}\,(x_i) \setminus v^\mu$. Since the term $p(x_i^\mu|\mathrm{pa}\,(x_i^\mu))$ therefore only depends on g_i^μ, we only require to average with respect to $q^\mu(g_i^\mu|v)$. If we use the optimal choice (EM setting), $q^\mu(g_i^\mu|v) = p(g_i^\mu|v^\mu)$, it is clear that this marginal is easy to calculate for any (poly)tree, since the marginal can be calculated by the JTA, and that therefore this term can be computed efficiently.

To be more specific, consider a simple five variable distribution with discrete variabl-es,

$$p(x_1, x_2, x_3, x_4, x_5) = p(x_1|x_2)p(x_2|x_3)p(x_3|x_4)p(x_4|x_5)p(x_5),$$

in which the variables x_2 and x_4 are consistently hidden in the training data, and training data for x_1, x_3, x_5 are always present. In this case, the contributions to the energy have the form

$$\sum_\mu \langle \log p(x_1^\mu|x_2)p(x_2|x_3^\mu)p(x_3^\mu|x_4)p(x_4|x_5^\mu)p(x_5^\mu\rangle_{q^\mu(x_2,x_4|x_1,x_3,x_5)}$$

Which may be written as

$$\begin{aligned}
&\sum_\mu \langle \log p(x_1^\mu|x_2)\rangle_{q^\mu(x_2,x_4|x_1,x_3,x_5)} \\
&+ \sum_\mu \langle \log p(x_2|x_3^\mu)\rangle_{q^\mu(x_2,x_4|x_1,x_3,x_5)} \\
&+ \sum_\mu \langle \log p(x_3^\mu|x_4)\rangle_{q^\mu(x_2,x_4|x_1,x_3,x_5)} \\
&+ \sum_\mu \langle \log p(x_4|x_5^\mu)\rangle_{q^\mu(x_2,x_4|x_1,x_3,x_5)} \\
&\qquad\qquad\qquad\qquad + \sum_\mu \log p(x_5^\mu) \qquad (14.2.1)
\end{aligned}$$

A useful property can now be exploited, namely that each term depends on only those hidden variables in the family that that term represents. Thus we may write

$$\begin{aligned}&\sum_\mu \langle \log p(x_1^\mu|x_2)\rangle_{q^\mu(x_2|x_1,x_3,x_5)}\\&+\sum_\mu \langle \log p(x_2|x_3^\mu)\rangle_{q^\mu(x_2|x_1,x_3,x_5)}\\&+\sum_\mu \langle \log p(x_3^\mu|x_4)\rangle_{q^\mu(x_4|x_1,x_3,x_5)}\\&+\sum_\mu \langle \log p(x_4|x_5^\mu)\rangle_{q^\mu(x_4|x_1,x_3,x_5)}\\&\qquad\qquad\qquad+\sum_\mu \log p(x_5^\mu)\end{aligned} \qquad (14.2.2)$$

It is clear that the final term causes us no difficulties, and this table can be set using the standard ML framework. Let us consider therefore a more difficult table, namely $p(x_1|x_2)$. When will the table entry $p(x_1 = i|x_2 = j)$ occur in the energy? This happens whenever x_1^μ is in state i. Since there is a summation over all the states of variables x_2 (due to the average), there is also a single time when variable x_2 is in state j. Hence the contribution to the energy from terms of the form $p(x_1 = i|x_2 = j)$ is

$$\sum_\mu I[x_1^\mu = i] q^\mu(x_2 = j|x_1, x_3, x_5) \log p(x_1 = i|x_2 = j)$$

where the indicator function $I[x_1^\mu = i]$ equals 1 if x_1^μ is in state i and is zero otherwise. To ensure normalisation of the table, we add a Lagrange term:

$$\sum_\mu I[x_1^\mu = i] q^\mu(x_2 = j|x_1, x_3, x_5) \log p(x_1 = i|x_2 = j) + \lambda \left\{ 1 - \sum_k p(x_1 = k|x_2 = j) \right\}$$

Differentiating with respect to $p(x_1 = i|x_2 = j)$ we get

$$\sum_\mu I[x_1^\mu = i] \frac{q^\mu(x_2 = j|x_1, x_3, x_5)}{p(x_1 = i|x_2 = j)} = \lambda$$

or

$$p(x_1 = i|x_2 = j) \propto \sum_\mu I[x_1^\mu = i] q^\mu(x_2 = j|x_1, x_3, x_5).$$

Hence

$$p(x_1 = i|x_2 = j) = \frac{\sum_\mu I[x_1^\mu = i] q^\mu(x_2 = j|x_1, x_3, x_5)}{\sum_{\mu,k} I[x_1^\mu = k] q^\mu(x_2 = j|x_1, x_3, x_5)}$$

Using the EM algorithm, we would use $q^\mu(x_2 = j|x_1, x_3, x_5) = p(x_2 = j|x_1^\mu, x_3^\mu, x_5^\mu)$. Note that this optimal distribution is easy to find for any polytree since this just corresponds to the marginal on the family, given some nodes in the graph are clamped in their evidential state. Hence, for EM, an update for the table would be

$$p^{new}(x_1 = i|x_2 = j) = \frac{\sum_\mu I[x_1^\mu = i] p^{old}(x_2 = j|x_1^\mu, x_3^\mu, x_5^\mu)}{\sum_{\mu,k} I[x_1^\mu = k] p^{old}(x_2 = j|x_1^\mu, x_3^\mu, x_5^\mu)} \qquad (14.2.3)$$

Similar expressions can be derived for the other tables. The important thing to note is that we only ever need local marginals for the variables in a family. These are always easy to obtain in polytrees (assuming that the number of states in a family is not too large), since this corresponds to inference in a tree conditioned on some evidence. Hence all updates in the EM algorithm are computable.

What about the table $p(x_2 = i|x_3 = j)$?

To ensure normalisation of the table, we add a Lagrange term:

$$\sum_\mu I[x_3^\mu = j] q^\mu(x_2 = i|x_1, x_3, x_5) \log p(x_2 = i|x_3 = j) + \lambda \left\{ 1 - \sum_k p(x_2 = k|x_3 = j) \right\}$$

As before, differentiating, and using the EM settings, we have

$$p^{new}(x_2 = i|x_3 = j) = \frac{\sum_\mu I[x_3^\mu = j] p^{old}(x_2 = i|x_1^\mu, x_3^\mu, x_5^\mu)}{\sum_{\mu,k} I[x_3^\mu = j] p^{old}(x_2 = k|x_1^\mu, x_3^\mu, x_5^\mu)} \tag{14.2.4}$$

There is a simple intuitive pattern to equation (14.2.3) and equation (14.2.4) :

If there were no hidden data, equation (14.2.3) would read

$$p^{new}(x_1 = i|x_2 = j) \propto \sum_\mu I[x_1^\mu = i] I[x_2^\mu = j]$$

and equation (14.2.4) would be

$$p^{new}(x_2 = i|x_3 = j) \propto \sum_\mu I[x_3^\mu = j] I[x_2^\mu = i]$$

All that we do, therefore, in the general EM case, is to replace those deterministic functions such as $I[x_2^\mu = i]$ by their missing variable equivalents $p^{old}(x_2 = i|x_1^\mu, x_3^\mu, x_5^\mu)$.

14.3 Optimising the Likelihood by Gradient methods

For latent variable models, sometimes the EM algorithm is very slow to converge.

According to [26], when the missing information is small compared to the complete information, EM exhibits approximate Newton behavior and enjoys fast, typically superlinear convergence in the neighborhood of the optimum point. If the fraction of missing information approaches unity, EM will exhibit extremely slow convergence.

An alternative is to compute the gradient of the likelihood directly.

$$L(\theta) = \log p(v|\theta)$$

Then

$$\partial_\theta L(\theta) = \frac{1}{p(v|\theta)} \partial_\theta p(v|\theta)$$

$$\partial_\theta L(\theta) = \frac{1}{p(v|\theta)} \partial_\theta \int_h p(v, h|\theta)$$

At this point, it may seem that computing the derivative is difficult. However, we may observe

$$\partial_\theta L(\theta) = \frac{\int_h p(v,h|\theta)}{p(v|\theta)} \partial_\theta \log p(v,h|\theta)$$

$$\partial_\theta L(\theta) = \int_h p(h|v,\theta) \partial_\theta \log p(v,h|\theta)$$

The rhs is just the average of the derivative of the complete likelihood. This is closely related to the EM algorithm, though note that the average is performed with respect the current distribution parameters θ and not θ^{old} as in the EM case. Used in this way, computing the derivatives of latent variable models is relatively straightforward. These derivatives may then be used as part of a standard optimisation routine such as conjugate gradients[26].

14.4 Iterated Proportional Fitting

An interesting questions is how to fit undirected models efficiently using Maximum Likelihood. Conditional Random Fields are a good example. (Wiegerinck-Heskes paper[27]. Also the work by Tony Jebara[28] on the reversed Jensen for Exponential Family models.)

The basic idea is as follows:

Consider an undirected distribution

$$p(v) = \frac{1}{Z} \prod_c \phi_c(v)$$

where $Z = \sum_v \prod_c \phi_c(v)$ ensures normalisation.

Given a set of data, $v^\mu, \mu = 1, \ldots, P$, how do we learn the ML parameters of the ϕ_c? Assuming iid data, the log likelihood is

$$L = \sum_\mu \sum_c \log \phi_c(v^\mu) - P \log Z$$

What makes this problem awkward is that the parameters also occur in Z, and hence the objective function does not split into a set of isolated parameter terms.

An upper bound on Z

Consider the bound, for positive x.

$$\log x \leq x - 1 \Rightarrow -\log x \geq 1 - x$$

Hence

$$-\log \frac{Z}{Z'} \geq 1 - \frac{Z}{Z'} \Rightarrow -\log Z \geq -\log Z' + 1 - \frac{Z}{Z'}$$

Let's call the parameters θ. Then we can write the bound (for a single datapoint, $P = 1$) as

$$L(\theta) \geq \underbrace{E(\theta) - \log Z(\theta^{old}) + 1 - \frac{Z(\theta)}{Z(\theta^{old})}}_{LB(\theta,\theta^{old})}$$

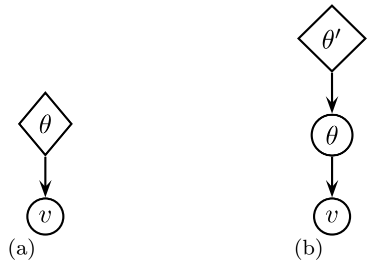


Figure 14.6: (a) Standard ML learning. (b) ML-II learning.

where $E(\theta)$ represents $\sum_c \log \phi_c(v^\mu)$ Hence

$$L(\theta) - L(\theta^{old}) \geq LB(\theta, \theta^{old}) - L(\theta^{old})$$

Using, the property that $L(\theta^{old}) = LB(\theta^{old}, \theta^{old}))$, we have

$$L(\theta) - L(\theta^{old}) \geq LB(\theta, \theta^{old}) - LB(\theta^{old}, \theta^{old})$$

Hence, provided we can find a θ that increases the lower bound on the likelihood, we are guaranteed to increase the likelihood itself. This is similar to the guarantees provided by the EM algorithm. The generalisation to multiple datapoints $P > 1$ just follows by summing the above over the datapoints.

The IPF procedure then follows iteratively maximising wrt θ. The potential advantage of this method over gradient based procedures is apparent if the optimum of $LB(\theta, \theta^{old}$ with respect to θ can be achieved in closed form. Otherwise, there may be little advantage[29].

14.5 Bayesian Methods and ML-II

Consider a parameterised distribution $p(v|\theta)$, and that we wish to set the parameters θ given some data. The model $p(v|\theta)$ is depicted in fig(14.6a), where the diamond indicates that no distribution is specified for that variable. For a single observed datapoint v, then setting θ by ML would correspond to finding the parameter θ that maximises $p(v|\theta)$.

In some cases, though, we may have an idea about which parameters θ are more appropriate. We can express this prior preference using a distribution $p(\theta)$. If the prior were fully specified, then there is nothing to 'learn' since $p(\theta|v)$ is now fully known. However, in many cases in practice, we are unsure of the exact parameter settings of the prior, and hence specify a parametersised prior using *hyperparameters* θ', using a distribution $p(\theta|\theta')$. This is depicted in fig(14.6b). The learning corresponds to finding the optimal θ' that maximises the likelihood $p(v|\theta') = \int_\theta p(v|\theta)p(\theta|\theta')$. This is known as an ML-II procedure since it corresponds to maximum likelihood, but at the higher, hyperparameter level. See [30] and [31]. This is a form of approximate Bayesian analysis since, although θ' is set using maximum likelihood, after training, we have a distribution over parameters, $p(\theta|v, \theta')$.

14.6 Problems

Exercise 44 (Printer Nightmare) *Cheapco is, quite honestly, a pain in the neck. Not only did they buy a dodgy old laser printer from StopPress and use* *it mercilessly, but try to get away with using substandard components and material. Unfortunately for StopPress, they have a contract to maintain Cheapco's old warhorse, and end up frequently sending the mechanic out to repair the printer. After the 10^{th} visit, they decide to make a model of Cheapco's printer, so that they will have a reasonable idea of the fault based only on the information that Cheapco's secretary tells them on the phone. In that way, StopPress hopes to be able to send out to Cheapco only a junior repair mechanic.*

Based on the manufacturer's information, StopPress has a good idea of the dependencies in the printer, and what is likely to directly affect other printer components. However, the way that Cheapco abuse their printer is a mystery, so that the exact probabilistic relationships between the faults and problems is idiosyncratic to Cheapco. However, StopPress has the following table of faults for each of the 10 visits. (Each column represents a visit, the transpose of the normal format).

fuse assembly malfunction	0	0	0	1	0	0	0	0	0	0
drum unit	0	0	0	0	1	0	0	1	0	0
toner out	1	1	0	0	0	1	0	1	0	0
poor paper quality	1	0	1	0	1	0	1	0	1	1
worn roller	0	0	0	0	0	0	1	0	0	0
burning smell	0	0	0	1	0	0	0	0	0	0
poor print quality	1	1	1	0	1	1	0	1	0	0
wrinkled pages	0	0	1	0	0	0	0	0	1	0
multiple pages fed	0	0	1	0	0	0	1	0	1	0
paper jam	0	0	1	1	0	0	1	1	1	1

Based on their knowledge of the printer, they build a graphical model.

1. *Load printer.bbnet*

   Using the above training data, complete the Conditional Probability Tables for the model. The CPT for "poor print quality" has already been completed. All other tables have been set initially to uniform values, and need to be altered to match the above table of training data.

   You will find that at times, there is insufficient data to complete a CPT – use your imagination to create a reasonable value for the missing CPT entry.

2. *Experiment with this network by clamping the bottom "problem" nodes.*

3. *How could the junior engineer use the network to help detect faults?*

Exercise 45 *Consider the graphical model $X_1 \to X_2$. Let $\theta_{k|j} = P(X_2 = s_k|X_1 = s_j)$ denote the probability that X_2 is in state k given that X_1 is in state j. Clearly $\sum_k \theta_{k|j} = 1$ for all j. We are given n observations of the states of X_1 and X_2. Let the number of transitions from $X_1 = s_j$ to $X_2 = s_k$ be denoted n_{kj}. Write down the likelihood of the data, and show that the maximum likelihood estimator $\hat{\theta}_{k|j} = n_{kj}/\sum_l n_{lj}$.*

Consider the case where X_0 is also a parent of X_2 (and not of X_1). Write down the ML estimator for $\theta_{k|ij} = P(X_2 = s_k|X_0 = s_i, X_1 = s_j)$ given count data $\{n_{kij}\}$.

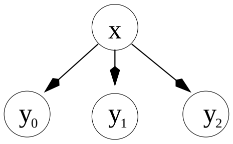
Exercise 46 *You have a machine that measures property x, the "orangeness" of liquids. You wish to discriminate between $\mathcal{C}_1$ = "IrnBru" and $\mathcal{C}_2$ = "Orangina". It is known that*

$$p(x|\mathcal{C}_1) = \begin{cases} 10 & 1.0 \leq x \leq 1.1 \\ 0 & \text{otherwise} \end{cases}$$

$$p(x|\mathcal{C}_2) = \begin{cases} 200(x-1) & 1.0 \leq x \leq 1.1 \\ 0 & \text{otherwise} \end{cases}$$

The prior probabilities $P(\mathcal{C}_1) = 0.6$ and $P(\mathcal{C}_2) = 0.4$ are also known from experience. Calculate the optimal Bayes' classifier and $P(\text{error})$.

Exercise 47 *If n observations $y_0, \ldots, y_{n-1}$ were noisy iid measurements of an underlying variable x, then the graphical model (for $n = 3$) would be*



The notation $P(x) \sim N(\mu, \sigma^2)$ is shorthand for the Gaussian distribution $P(x) = e^{-(x-\mu)^2/2\sigma^2}/\sqrt{2\pi\sigma^2}$. Assume that $P(x) \sim N(0, \sigma_0^2)$ and $P(y_i|x) \sim N(x, \sigma^2)$ for $i = 0, \ldots, n-1$. Show that $P(x|y_0, \ldots, y_{n-1})$ is Gaussian with mean

$$\mu = \frac{n\sigma_0^2}{n\sigma_0^2 + \sigma^2}\overline{y}$$

where $\overline{y} = (y_0 + y_1 + \ldots + y_{n-1})/n$ and variance σ_n^2 such that

$$\frac{1}{\sigma_n^2} = \frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}.$$

Exercise 48 (Bayesian analysis) . *Consider the beta distribution $p(\theta) = c(\alpha, \beta)\theta^{\alpha-1}(1-\theta)^{\beta-1}$, where $c(\alpha, \beta)$ is a normalizing constant. The mean of this distribution is $E[\theta] = \alpha/(\alpha + \beta)$. For $\alpha, \beta > 1$ the distribution is unimodal (i.e. it has only one maximum). Find the value θ^* where this maximum is attained, and compare it to the mean. For what values of α and β to the mean and θ^* coincide?*

Exercise 49 *Consider the multivariate Gaussian distribution $p(x) \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ on the vector x with components $x_1, \ldots, x_n$:*

$$p(x) = \frac{1}{\sqrt{|2\pi\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$$

Calculate $p(x_i|x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$.

Exercise 50 *Suppose that instead of using the Bayes' decision rule to choose class k if $P(\mathcal{C}_k|x) > P(\mathcal{C}_j|x)$ for all $j \neq k$, we use a randomized decision rule, choosing class j with probability $Q(\mathcal{C}_j|x)$. Calculate the error for this decision rule, and show that the error is minimized by using Bayes' decision rule.*

Exercise 51 (Bayesian analysis) . *Consider the problem of the Bayesian analysis of θ, the probability that a certain coin will come up heads. You specify the prior distribution as $Beta(\alpha_h, \alpha_t)$, and then repeatedly toss the coin. After each toss you update the posterior distribution for θ. Write a short matlab program that does the following:*

- *Takes the true value of θ*

- *Takes values for α_h and α_t.*
- *Generates a sequence of (pseudo) random draws from the coin [Hint: use* `rand` *and test to which side of the true θ this value falls.* `rand` *generates pseudo random numbers in $(0,1)$.]*
- *Plots the posterior distribution for θ sequentially, i.e. as each new draw comes in. [Hint: for drawing the distribution you do not need to calculate the Γ function normalization terms.]*

As the number of observations becomes large, explain the behaviour you observe.

Exercise 52 *The Gaussian distribution in one dimension is defined as*

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

and satisfies $\int_{-\infty}^{\infty} p(x)dx = 1$.

Show that $\int_{-\infty}^{\infty} xp(x)dx = \mu$.

Show that $\int_{-\infty}^{\infty} (x-\mu)^2 p(x)dx = \sigma^2$.

Exercise 53 *Consider data $x^i, i = 1, \ldots, P$. Show that the Maximum Likelihood estimator of μ is $\hat{\mu} = \frac{1}{P}\sum_{i=1}^{P} x^i$ and that the ML estimate of σ^2 is $\hat{\sigma^2} = \frac{1}{P}\sum_{i=1}^{P}(x^i - \mu)^2$*

Exercise 54 *A training set consists of one dimensional examples from two classes. The training examples from class 1 are*

$$0.5, 0.1, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.35, 0.25$$

and from class 2 are

$$0.9, 0.8, 0.75, 1.0$$

Fit a (one dimensional) Gaussian using Maximum Likelihood to each of these two classes. Also estimate the class probabilities p_1 and p_2 using Maximum Likelihood. What is the probability that the test point $x = 0.6$ belongs to class 1?

Exercise 55 *Given the distributions $p(x|class1) = N(\mu_1, \sigma_1^2)$ and $p(x|class2) = N(\mu_2, \sigma_2^2)$, with corresponding prior occurrence of classes p_1 and p_2 ($p_1 + p_2 = 1$), calculate the decision boundary explicitly as a function of $\mu_1, \mu_2, \sigma_1^2, \sigma_2^2, p_1, p_2$. How many solutions are there to the decision boundary, and are they all reasonable?*

14.7 Solutions

III. Probabilistic models in Machine Learning

15 Introduction to Bayesian Methods

Introduction

Regarding the general problem of fitting models to data, we are rarely certain about either our data measurements (they may be inherently 'noisy') or model beliefs. It is natural to use probabilities to account for these uncertainties. How can we combine our data observations with these modelling uncertainties in a consistent and meaningful manner? The Bayesian approach provides a consistent framework for formulating a response to these difficulties, and is noteworthy for its conceptual elegance[32, 30, 33, 34]. Indeed, throughout these chapters, we have been using the Bayesian framework, since this is simply inherent in the correct use of probabilities in graphical models and, in particular, inference in graphical models. However, here we investigate a little more Bayes' rule and it's applications in some very simple graphical models.

As a reminder to the reader, the fundamental probabilistic relationship required for inference is the celebrated Bayes' rule which, for general events A,B,C is

$$p(A|B,C) = \frac{p(B|A,C)p(A|C)}{p(B|C)} \tag{15.0.1}$$

In modelling data, it is convenient to think of different levels of uncertainty in formulating a model. At the lowest level, we may assume that we have the correct model, but are uncertain as to the parameter settings θ for this model. This assumption details how observed data is generated, $p\left(\text{data}|\theta, \text{model}\right)$. The task of inference at this level is to calculate the posterior distribution of the model parameter. Using Bayes' rule, this is

$$p(\theta|\text{data}, \text{model}) = \frac{p\left(\text{data}|\theta, \text{model}\right) p\left(\theta|\text{model}\right)}{p\left(\text{data}|\text{model}\right)} \tag{15.0.2}$$

Thus, if we wish to infer model parameters from data we need two assumptions: (1) How the observed data is generated under the assumed model, the *likelihood* $p\left(\text{data}|\theta, \text{model}\right)$ and (2) Beliefs about which parameter values are appropriate, before the data has been observed, the *prior* $p(\theta|\text{model})$. (The denominator in equation (15.0.2) is the normalising constant for the posterior and plays a role in uncertainty at the higher, model level). That these two assumptions are required is an inescapable consequence of Bayes' rule, and forces the Bayesian to lay bare all necessary assumptions underlying the model.

Coin Tossing Example

Let θ be the probability that a coin will land up heads. An experiment yields the data, $D = \{h, h, t, h, t, h, \ldots\}$, which contains H heads and T tails in $H + T$ flips of the coin. What can we infer about θ from this data? Assuming that each coin is flipped independently, the likelihood of the observed data is

$$p\left(D|\theta, \text{model}\right) = \theta^H \left(1 - \theta\right)^T \tag{15.0.3}$$

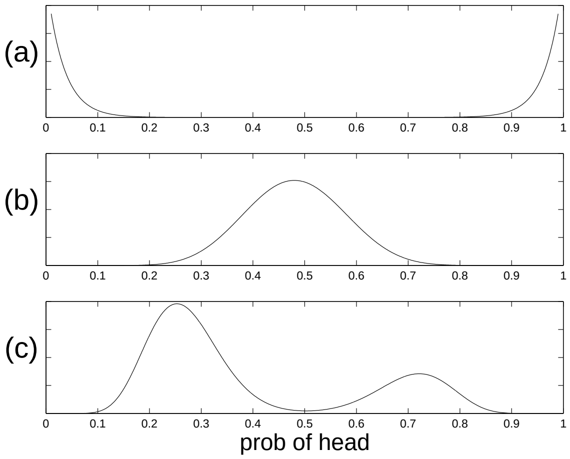


Figure 15.1: Coin Tossing: (a) The prior: this indicates our belief that the coin is heavily biased. (b) The likelihood after 13 Tails and 12 Heads are recorded, $\theta^{ML} = 0.48$. (c) The posterior: the data has moderated the strong prior beliefs resulting in a posterior less certain that the coin is biased. $\theta^{MAP} = 0.25$, $\bar{\theta} = 0.39$

A standard approach in the statistical sciences is to estimate θ by maximising the likelihood, $\theta^{ML} = \arg\max_\theta p(D|\theta, \text{model})$. Strictly speaking, this approach is non-Bayesian since it does not require the specification of a prior and, consequently, theories which deal with uncertainty in ML estimators are primarily concerned with the data likelihood, and not directly posterior parameter uncertainty. In the Bayesian approach, however, we need to be explicit about our prior beliefs $p(\theta|\text{model})$. These are updated by the observed data to yield the posterior distribution

$$p\left(\theta|D, \text{model}\right) \propto \theta^H \left(1-\theta\right)^T p\left(\theta|\text{model}\right) \tag{15.0.4}$$

The Bayesian approach is more flexible than maximum likelihood since it allows (indeed, *instructs*) the user to calculate the effect that the data has in modifying prior assumptions about which parameter values are appropriate. For example, if we believe that the coin is heavily biased, we may express this using the prior distribution in fig(15.1a). The likelihood as a function of θ is plotted in fig(15.1b) for data containing 13 Tails and 12 Heads. The resulting posterior fig(15.1c) is bi-modal, but less extreme than the prior. It is often convenient to summarise the posterior by either the maximum a posteriori (MAP) value, or the mean, $\bar{\theta} = \int \theta p(\theta|D) d\theta$. Such a summary is not strictly required by the Bayesian framework, and the best choice of how to summarise the posterior depends on other loss criteria[30].

Model Comparison and Hierarchical Models

The above showed how we can use the Bayesian framework to assess which parameters of a model are *a posteriori* appropriate, given the data at hand. We can carry out a similar procedure at a higher, model level to asses which models are more appropriate fits to the data. In general, the model posterior is given by

$$p(M|D) = \underbrace{p(D|M)}_{\text{Model Likelihood}} \quad \underbrace{p(M)}_{\text{Model Prior}} \; /p(D) \tag{15.0.5}$$

If the model is parameterised by some unknown variable θ, we need to integrate this out to calculate the model likelihood,

$$p(D|M) = \int p(D|\theta, M)p(\theta|M)d\theta \tag{15.0.6}$$

Comparing two competing model hypotheses M_1 and M_2 is straightforward

$$\frac{p(M_1|D)}{p(M_2|D)} = \underbrace{\frac{p(D|M_1)}{p(D|M_2)}}_{\text{Bayes Factor}} \frac{p(M_1)}{p(M_2)} \tag{15.0.7}$$

In the coin example, we can use this to compare the biased coin hypothesis (model M_1 with prior given in fig(15.1a)) with a less unbiased hypothesis formed by using a Gaussian prior $p(\theta|M_2)$ with mean 0.5 and variance 0.1^2 (model M_2). This gives a Bayes factor $p(D|M_1)/p(D|M_2) \approx 0.00018$. If we have no prior preference for either model M_1 or M_2, the data more strongly favours model M_2, as intuition would suggest. If we desired, we could continue in this way, forming a hierarchy of models, each less constrained than the submodels it contains.

Bayes automatically penalises overcomplex models

For simplicity, consider first two models $M1$ and $M2$ whose parameter spaces are of the same dimension, and for which the prior is flat. The model likelihood $p(D|M1) = \int_\theta p(D|\theta, M1)p(\theta|M1)$ is therefore essentially dominated by only the high likelihood region. If model $M2$ has roughly the same likelihood values where it fits well, but has a higher volume of them, then the likelihood for model $M2$ is higher than for model $M1$, see fig(15.2). This also explains why a model which has a higher dimension parameter space will usually be rejected in favour of a model which fits equally well with a smaller parameter space, since the prior in the latter case is a unit mass spread out in a smaller number of dimensions, which will therefore have a higher weight.

Non Bayesian and Minimal Empirical Risk criteria

The philosophy in the Bayesian approach is that parameter (or model) uncertainty is reduced in the presence of observed data. However, except in pathological cases (such as an infinite amount of training data) there still remains uncertainty in the parameters. Whilst the Bayesian principal is well-established, not everyone is convinced. A popular, non-bayesian method for model or parameter determiniation runs somewhat as follows. We split the available training data into two sets, a training set and a validation set. Model $M1$ and model $M2$ are both trained on the training data, giving a single 'optimal' parameter for each model. Then each model with its optimal parameter is then tested on the validation set. That model which has the better performance on the validation set is then preferred. In this sense, the uncertainty is not in the parameter space (since only a single optimal parameter is retained). Rather, the uncertainty is in the predictive performance of each model.

Need to talk about classical hypothesis testing....... The predictive performance is (often, but not necessarily) assumed to be Gaussian. Then we would perhaps

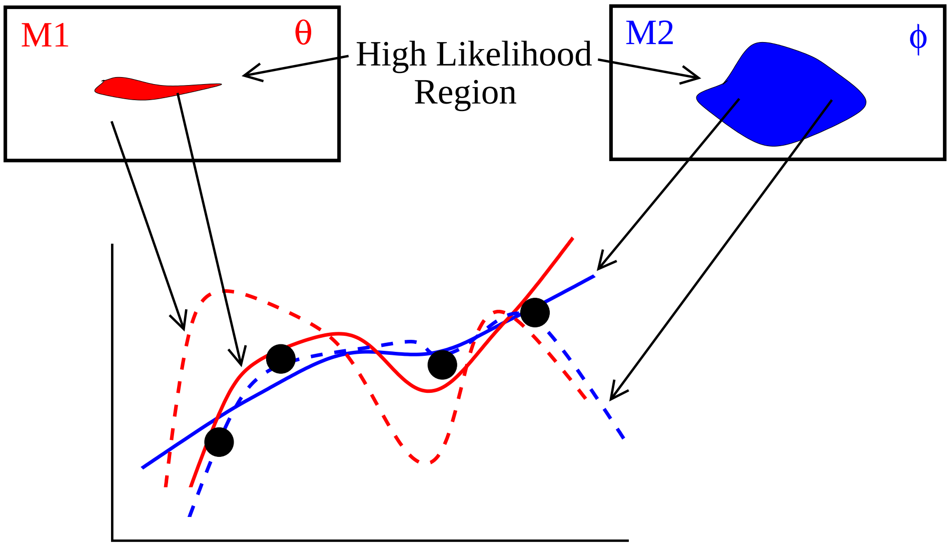


Figure 15.2: The points represent data for which we wish to find a function that goes through the datapoints well. We consider two model classes $M1$ and $M2$ – each with their own parameter spaces. For example, $M1$ might represent polynomicals of order 20, and $M2$ polynomials of order 10. In $M1$, there is only a small region of parameter space for which the function fits well, for example, the solid red curve. If we move slightly away from this region, and use the red-dashed function, due to the complex nature of the model, by 'definition' this means that it will fit many other kinds of data, and hence be sensitive in this way. On the otherhand, for model $M2$, there is a large area of the parameter space for which the functions fit well. Since $p(D|M) = \int_\theta p(D|\theta, M)p(\theta|M)$, this means that the 'evidence' for how well the model fits is roughly the volume of the space for which the likelihood is very high. For two models for which the likelihood is equally high, since the prior $p(\theta|M)$ is a unit mass spread out over the parameter space, then the model for which the likelihood covers a higher amount of the space will be preferred.

observe a certain performance, and judge whether or not this is more typical of model 1 or 2....blah...

Need to explain the differences in these approaches, (see also the Laplace to supernova paper).

A detailed example : Bayesian Error Analysis

We consider here how to assess if two classifiers, based on a set of test error results, are performing equally well. This question is often considered in the realm of sampling theory, based on classical hypothesis testing. Here we present a simple Bayesian treatment that is quite general, and also is able to deal with the (practically common) case where the errors that two classifiers make are *dependent*. This is an introduction to Bayesian statistics applied to the analysis of experimental results. In particular, the situation that we are interested in is how to tell if two machine learning classifiers are performing the same[1]. This is a standard problem in assessing the results of a machine learning experiment[2]. Let's say classifier A makes 20 errors, and 35 correct classifications, whereas classifier B makes 23 errors and 32 correct classifications – is classifier A really better than classifier B? Of course, intuitively, the uncertainty stems from the small amount of results. If we had rather that classifier A makes 20000 errors and 35000 correct classifications, classifier B makes 23000 errors and 32000 correct classifications, intuitively, we would be much more confident than in the previous scenario that classifier A is better than classifier B.

Initially, we will try to answer here the above question by appealing to a basic Bayesian analysis. In doing this, we assume that the errors/labels that two classifiers make do not depend on each other. Later we'll relax this to show how one can make a test to see if they are dependent.

This small note is essentially a classic exercise in Bayesian statistics. The interested reader may consult [35] (chapter 37 is most closely related to this area, although the whole of part IV is relevant) and references therein for an introduction to the general area.

Error Analysis

Consider a situation where two classifiers A and B have been tested on some data, so that we have, for each example in the test set, an error pair

$$(e_a(\mu), e_b(\mu)), \mu = 1, \dots P$$

where P is the number of test data points, and $e_a \in \{1, \dots Q\}$ (and similarly for e_b). That is, there are Q possible types of error that can occur. This is useful in text classification, where TruePositive, FalseNegative, TrueNegative and FalsePositive might form four kinds of 'errors'. For notational simplicity we also call a TruePositive an 'error'. It might be more appropriate to use a term such as 'outcome label', although this should also not be confused with the class label of

[1] The theory is readily extensible to multiple classifiers, and is left as an exercise for the interested reader.

[2] Ideally, a true Bayesian will use a Bayesian Classifier, for which there will always, in principle, be a direct way to estimate the suitability of the model in explaining the experimental data. We consider here the less fortunate situation where two non-Bayesian classifiers have been used, and only their test performances are available for evaluating the classifiers.

the classifier – it is their evaluation against the truth that we are interested in.

Let's call $e_a = \{e_a(\mu), \mu = 1, \ldots, P\}$, the sample set A, and similarly for e_b.

We are here interested in the question :

'How much evidence is there supporting that the two classifiers are performing differently?'

Mathematically, our major *assumption* here is that this is the same question as :

'How much evidence is there in favour of two sample sets being from different multinomial distributions?'

The main question that we address here is to test whether or not two classifiers are essentially performing the same. To do this, we have two hypotheses :

1. H_{indep} : The sample sets are from different distribution.
2. H_{same} : The sample sets are from the same distribution.

We need then to formally mathematically state what these two hypotheses mean. In both cases, however, we will make the *independence of trials assumption*

$$p(e_a, e_b) = \prod_\mu p(e_a(\mu), e_b(\mu)).$$

H_{indep} : Independence of classifiers

For H_{indep}, we assume

$$p(e_a(\mu), e_b(\mu)|H_{indep}) = p(e_a(\mu)|H_{indep})p(e_b(\mu)|H_{indep})$$

Note that we don't *need* to make this independence-of-errors assumption, since it is often quite reasonable to assume that both classifiers will tend to perform well on the same 'easy' example, and perhaps poorly on the same 'difficult' example, see fig(15.3). We'll consider how to implement this case in a later section.

Since each classifier can make one of Q types of errors, we need to specify what the probability of making such an error could be. For classifier A, we write

$$\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_Q), \qquad \sum_q \alpha_q = 1$$

and similarly for $\boldsymbol{\beta}$. (These are the values of the probability tables for generating errors).

Hence, under the independence assumption a probability for generating an error on the μ^{th} example,

$$p(e_a(\mu)|\boldsymbol{\alpha}, H_{indep})$$

and likewise for classifier B.

The data likelihood

Since the data is assumed generated by a multinomial distribution, the likelihood of generating the training data is

$$p(e_a|\boldsymbol{\alpha}) = \prod_{q=1}^{Q} \alpha_q^{c_q^a}$$

where c_q^a is the number of times that classifier A makes error q[3]. A similar expression holds for classifier B.

Dirichlet Prior

Since we are dealing with multinomial distributions, it is convenient to use the Dirichlet prior, which is conjugate to the multinomial distribution:

$$p(\boldsymbol{\alpha}) = \frac{1}{Z(u)} \prod_q \alpha_q^{u_q - 1}$$

where

$$Z(u) = \frac{\prod_{q=1}^{Q} \Gamma(u_q)}{\Gamma\left(\sum_{q=1}^{Q} u_q\right)}$$

The prior parameter u controls how strongly the mass of the distribution is pushed to the corners of the simplex. Setting $u_q = 1$ for all q corresponds to a uniform prior. The uniform prior assumption is reasonable, although there may be situations where it would preferable to use non-uniform priors[4].

Posterior

With a Dirichlet prior and a multinomial likelihood term, the posterior is another Dirichlet distribution (dropping the a index, since this result is general),

$$p(\boldsymbol{\alpha}|e) = \frac{1}{Z(u+c)} \prod_{q=1}^{Q} \alpha_q^{c_q + u_q - 1} \tag{15.0.8}$$

where c are the counts of the errors.

[3] The kinds of errors are assumed mutually exclusive and exhaustive. Clearly, the exhaustive condition means that there are dependencies in the errors produced at one time – this is taken care of by the constraint that the probabilities sum to one. Also, a potential source of confusion is whether or not we view the dataset errors as a sample from a distribution with a fixed total number of errors, say 50 TruePositve, 76 FalseNegative, etc, which would add combinatorial prefactors to the data likelihoods. Clearly, also in this case, if we know the total number of datapoints, then the errors are also not independent. Essentially, in the context here, using such a system is incorrect, since we only have a *single* dataset, and our interest is in the likelihood of generating this single dataset, and not in the likelihood of generating error counts.

[4] We could also use $p(\theta)$ to specify a distribution of priors $p(u|\theta)$, over which one can then integrate.

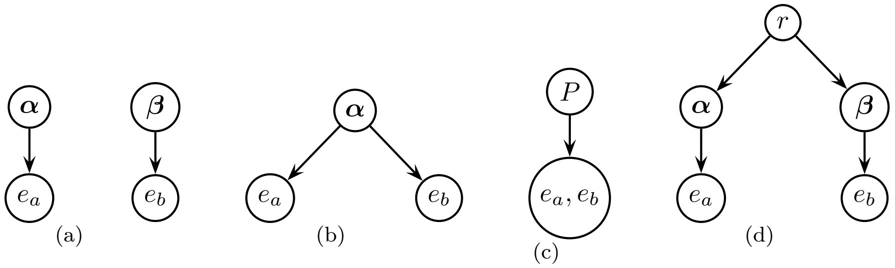


Figure 15.3: (a) H_{indep} : Corresponds to the errors for the two classifiers being independently generated. (b) H_{same}: both errors are generated from the same distribution. (c) H_{dep} : the errors are dependent ('correlated'). (d) $H_{related}$: In this case the distributions $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ which generate e_a and e_b are related in some way – for example they may be constrained to be similar through the variable r. This case is not considered in the text.

H_{indep} : Model Likelihood

In the Bayesian framework, we want to find how likely it is that a model/hypothesis is responsible for generating the data:

$$p(H_{indep}|e_a, e_b) = p(e_a, e_b|H_{indep})p(H_{indep})/p(e_a, e_b)$$

where $p(H_{indep})$ is our prior belief that H_{indep} is the correct hypothesis. Note that the normalising constant $p(e_a, e_b)$ does not depend on the hypothesis. Then

$$\begin{aligned}&p(e_a, e_b)p(H_{indep}|e_a, e_b) \qquad (15.0.9)\\&= \int p(e_a, e_b|\boldsymbol{\alpha}, \boldsymbol{\beta}, H_{indep})p(\boldsymbol{\alpha}, \boldsymbol{\beta}|H_{indep})p(H_{indep})d\boldsymbol{\alpha}d\boldsymbol{\beta}\\&= p(H_{indep})\int p(e_a|\boldsymbol{\alpha}, H_{indep})p(\boldsymbol{\alpha}|H_{indep})d\boldsymbol{\alpha}\int p(e_b|\boldsymbol{\beta}, H_{indep})p(\boldsymbol{\beta}|H_{indep})d\boldsymbol{\beta}\end{aligned}$$

where we assumed (pretty innocuously) $p(\boldsymbol{\alpha}, \boldsymbol{\beta}|H_{indep}) = p(\boldsymbol{\alpha}|H_{indep})p(\boldsymbol{\beta}|H_{indep})$. Let's calculate

$$\int p(e_a|\boldsymbol{\alpha}, H_{indep})p(\boldsymbol{\alpha}|H_{indep})d\boldsymbol{\alpha} = \frac{1}{Z(u)}\prod_q \alpha_q^{c_q^a+u_q-1} = \frac{Z(u+c^a)}{Z(u)}$$

Hence

$$p(H_{indep}|e_a, e_b) = p(H_{indep})\frac{Z(u+c^a)}{Z(u)}\frac{Z(u+c^b)}{Z(u)}$$

H_{same} : Model Likelihood

In H_{same}, the hypothesis is that the errors for the two classifiers are generated from the *same* multinomial distribution. Hence

$$\begin{aligned}&p(e_a, e_b)p(H_{same}|e_a, e_b) \qquad (15.0.10)\\&= p(H_{same})\int p(e_a|\boldsymbol{\alpha}, H_{same})p(e_b|\boldsymbol{\alpha}, H_{same})p(\boldsymbol{\alpha}|H_{same})d\boldsymbol{\alpha}\\&= p(H_{same})\frac{Z(u+c^a+c^b)}{Z(u)}\end{aligned}$$

Bayes Factor

If we assume that we have no prior preference for either hypothesis ($p(H_{indep}) = p(H_{same})$), then

$$\frac{p(H_{indep}|e_a, e_b)}{p(H_{same}|e_a, e_b)} = \frac{Z(u + c^a)Z(u + c^b)}{Z(u)Z(u + c^a + c^b)}$$

This is the evidence to suggest that the data were generated by two different multinomial distributions. In other words, this is the evidence in favour of the two classifiers being different.

Examples

In the experiments that I demonstrate here and elsewhere, I'll assume that there are three kinds of 'errors', $Q = 3$.

- We have the two error counts $c^a = [39, 26, 35]$ and $c^b = [63, 12, 25]$

  Then, the above Bayes factor is 20.7 – strong evidence in favour of the two classifiers being different. (This is consistent with the model I used to generate the data – they were indeed from different multinomial distributions).

- Alternatively, if we have the two error counts $c^a = [52, 20, 28]$ and $c^b = [44, 14, 42]$

  Then, the above Bayes factor is 0.38 – weak evidence against the two classifiers being different. (This is consistent with the model I used to generate the data – they were indeed from the same multinomial distributions)

- As a final example, consider counts $c^a = [459, 191, 350]$ and $c^b = [465, 206, 329]$. This gives a Bayes factor of 0.008 – strong evidence that the two classifiers are statistically the same (Indeed, the errors were in this case generated by the same multinomial).

These results show that the Bayesian analysis performs in a way that is consistent with the intuition that the more test data we have, the more confident we are in our statements about which is the better model.

Dependent Error Analysis

Here we consider the (perhaps more common) case that errors are *dependent*. For example, it is often the case that if classifier A works well, then classifier B will also work well. Similarly, if one classifier performs poorly, then often the other will too. Here, we assume that dependencies exist, but we make no preferences for one to another (of course, such preferences would be straightforward to include if desired). (There may be some interest in situations where if classifier A performs poorly, then classifier B is likely to perform well). Thus we want to consider the Hypothesis

H_{dep} : the errors that the two classifiers make are dependent.

For convenience, let's write

$$e = (e_a, e_b)$$

Mathematically, we need to specify a distribution:

$$p(e_a(\mu), e_b(\mu)|H_{dep}) = p(e_a(\mu), e_b(\mu)|P, H_{dep})$$

here P is a $Q \times Q$ matrix of probabilities:

$$[P]_{ij} = p(e_a = i, e_b = j)$$

namely that the ij element of P is the probability that A makes error i, and B makes error j.

Then, as before

$$\begin{aligned}\frac{1}{p(H_{dep})} p(e)p(H_{dep}|e) &= p(e|H_{dep}) \\ &= \int p(e, P|H_{dep})dP \\ &= \int p(e|P, H_{dep})p(P|H_{dep})dP\end{aligned}$$

Assuming a Dirichlet prior on P, with parameters U, we have

$$p(e)p(H_{dep}|e) = p(H_{dep})\frac{Z(vec\,(U + C))}{Z(vec(U))}$$

where $vec(D)$ simply forms a vector by concatenating the rows of the matrix D. Here C is the count matrix, with $[C]_{ij}$ equal to the number of times that joint error $(e_a = i, e_b = j)$ occurred in the P datapoints. As before, we can then use this in a Bayes factor calculation. For the uniform prior, $[U]_{ij} = 1, \forall i, j$.

Testing for dependencies in the classifiers

Imagine that we wish to test whether or not the errors of the classifiers are dependent H_{dep}, against the hypothesis that they are independent H_{indep}.

$$\frac{p(H_{indep})}{p(H_{dep})} = \frac{Z(u + c^a)}{Z(u)}\frac{Z(u + c^b)}{Z(u)}\frac{Z(vec(U))}{Z(vec\,(U + C))}$$

Examples

- Consider the error count matrix C

  $$\begin{pmatrix} 98 & 7 & 93 \\ 168 & 13 & 163 \\ 245 & 12 & 201 \end{pmatrix}$$

  so that $c^a = [511, 32, 457]$, and $c^b = [198, 344, 458]$. Then

  $$\frac{p(H_{indep})}{p(H_{dep})} = 3020$$

  – very strong evidence that the classifiers perform independently (indeed, this is consistent with the way I generated the data).

- Consider the error count matrix C

  $$\begin{pmatrix} 82 & 120 & 83 \\ 107 & 162 & 4 \\ 170 & 203 & 70 \end{pmatrix}$$

  so that $c^a = [359, 485, 156]$, and $c^b = [284, 273, 443]$. Then

  $$\frac{p(H_{indep})}{p(H_{dep})} = 2 \times 10^{-18}$$

  – extremely strong evidence that the classifiers perform dependently (again, this is consistent with the way I generated the data).

Dependent vs the Same

Perhaps the most useful test that can be done practically is between the H_{dep} versus H_{same}. This is because, in practice, it is reasonable to believe that dependencies are quite likely in the errors that classifiers make (both classifiers will do well on 'easy' test examples, and badly on 'difficult' examples). In this sense, it is natural to believe that dependencies will most likely exist in practice. The relevant question is : are these dependencies strong enough to make us believe in fact that the errors are coming from the *same* process? In this sense, we want to test

$$\frac{p(H_{same})}{p(H_{dep})} = \frac{Z(u + c^a + c^b)}{Z(u)} \frac{Z(vec(U))}{Z(vec\,(U + C))}$$

- Consider an experiment which gives the test error count matrix C

  $$\begin{pmatrix} 105 & 42 & 172 \\ 42 & 45 & 29 \\ 192 & 203 & 170 \end{pmatrix}$$

  so that $c^a = [339, 290, 371]$, and $c^b = [319, 116, 565]$. Then

  $$\frac{p(H_{same})}{p(H_{dep})} = 4.5 \times 10^{-38}$$

  – extremely strong evidence that the classifiers are performing differently (this is consistent with the way I generated this data set).

- Consider an experiment which gives the test error count matrix C

  $$\begin{pmatrix} 15 & 8 & 10 \\ 5 & 4 & 8 \\ 13 & 12 & 25 \end{pmatrix}$$

  so that $c^a = [33, 24, 43]$, and $c^b = [33, 17, 50]$. Then

  $$\frac{p(H_{same})}{p(H_{dep})} = 42$$

  – strong evidence that the classifiers are performing the same (this is consistent with the way I generated this data set).

H_{dep} :Looking at the Posterior

Personally, I think that in machine learning, it is most natural to consider H_{dep}, due to the typical situation that when classifier A works well on example μ, then so will classifier B. Conversely, when one works badly on an example, then so will the other. In this case, we may simply want to work with H_{dep}, and try to assess this posterior directly.

Consider the following simple example where $Q = 2$. In this case P is a 2×2 matrix, with elements P_{ij} representing the probability that classifier A generates an error of type i, and classifier B makes an error on type j on the same example.

...to be continued. I want to write down the posterior mean, and standard error bars around the posterior P_{ij}. This involves marginals of Dirichlet. It would also be good to try to answer if A makes more errors than B – this involves integration over a restricted space.....

15.1 Problems

Exercise 56 *blah*

15.2 Solutions

56

16 Bayesian Regression

Bayesian Regression

Regression refers to inferring an unknown input-output mapping on the basis of observed data $D = \{(x^\mu, t^\mu), \mu = 1, \ldots P\}$, where (x^μ, t^μ) represents an input-output pair. For example, fit a function to the crosses in fig(16.1a). Since there is the possibility that each observed output t^μ has been corrupted by noise, we would like to recover the underlying clean input-output function. We assume that each (clean) output is generated from the model $f\left(x; w\right)$ where the parameters w of the function f are unknown and that the observed outputs t^μ are generated by the addition of noise η to the clean model output,

$$t = f\left(x; w\right) + \eta \tag{16.0.1}$$

If the noise is Gaussian distributed, $\eta \sim N(0, \sigma^2)$, the model M generates an output t for input x with probability

$$p(t|w, x, M) = \exp\left\{-\frac{1}{2\sigma^2}\left(t - f(x; w)\right)^2\right\} / \sqrt{2\pi\sigma^2} \tag{16.0.2}$$

If we assume that each data input-output pair is generated identically and independently from the others, the data likelihood is

$$p(D|w, M) = \prod_{\mu=1}^{P} p(t^\mu|w, x^\mu, M) \tag{16.0.3}$$

(Strictly speaking, we should write $p(t^1, \ldots, t^P|w, x^1, \ldots, x^P, M)$ on the left hand side of the above equation. However, since we assume that the training inputs are fixed and non-noisy, it is convenient and conventional to write $p(D|w, M)$). The posterior distribution $p(w|D, M) \propto p(D|w, M)p(w|M)$ is

$$\log p(w|D, M) = -\frac{\beta}{2}\sum_{\mu}\left(t^\mu - f(x^\mu; w)\right)^2 + \log p\left(w|M\right) + \frac{P}{2}\log\beta + const. \tag{16.0.4}$$

where $\beta = 1/\sigma^2$. Note the similarity between equation (16.0.4) and the sum square regularised training error used in standard approaches to fitting functions to data, for example using neural networks [36]. In the Bayesian framework, we can motivate the choice of a sum square error measure as equivalent to the assumption of additive Gaussian noise. Typically, we wish to encourage smoother functions so that the phenomenon of overfitting is avoided. One approach to solving this problem is to use a regulariser penalty term to the training error. In the Bayesian framework, we use a prior to achieve a similar effect. In principle, however, the Bayesian should make use of the full posterior distribution, and not just a single weight value. In standard neural network training, it is good practice to use committees of networks, rather than relying on the prediction of a single network[36]. In the Bayesian framework, the posterior automatically specifies a committee (indeed, a distribution) of networks, and the importance attached to each committee members prediction is simply the posterior probability of that network weight.

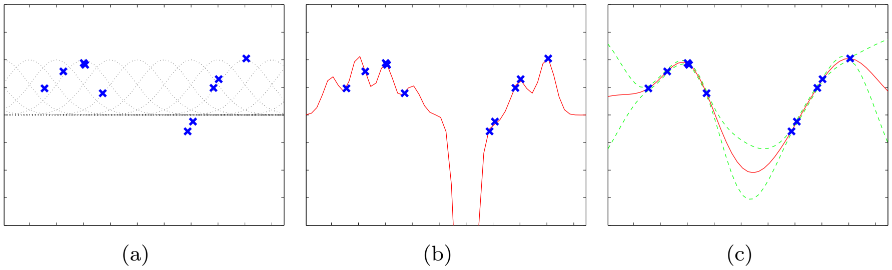


Figure 16.1: Along the horizontal axis we plot the input x and along the vertical axis the output t. (a) The raw input-output training data. (b) Prediction using regularised training and fixed hyperparameters. (c) Prediction with error bars, using ML-II optimised hyperparameters.

RBFs and Generalised Linear Models

Generalised linear models have the form

$$f(x;w) = \sum_i w_i \phi_i(x) \equiv w^T \Phi(x) \tag{16.0.5}$$

Such models have a linear parameter dependence, but nevertheless represent a non-linear input-output mapping if the basis functions $\phi(x), i = 1, \ldots, k$ are non-linear. Radial basis functions are an example of such a network[36]. A popular choice is to use Gaussian basis functions $\phi_i(x) = \exp(-\left(x - \boldsymbol{\mu}^i\right)^2 / (2\lambda^2))$. In this discussion, we will assume that the centres $\boldsymbol{\mu}^i$ are fixed, but that the width of the basis functions λ is a hyperparameter that can be adapted. Since the output is linearly dependent on w, we can discourage extreme output values by penalising large weight values. A sensible weight prior is thus

$$\log p(w|\alpha) = -\frac{\alpha}{2} w^T w + \frac{k}{2} \log \alpha + const. \tag{16.0.6}$$

Under the Gaussian noise assumption, the posterior distribution is

$$\log p(w|\Gamma, D) = -\frac{\beta}{2} \sum_{\mu=1}^{P} (t^\mu - w^T \Phi(x))^2 - \frac{\alpha}{2} w^T w + const. \tag{16.0.7}$$

where Γ represents the hyperparameter set $\{\alpha, \beta, \lambda\}$. (We drop the fixed model dependency wherever convenient). The weight posterior is therefore a Gaussian, $p(w|\Gamma, D) = N(\bar{w}, S)$ where

$$S = \left(\alpha I + \beta \sum_{\mu=1}^{P} \Phi\left(x^\mu\right) \Phi^T\left(x^\mu\right) \right)^{-1} \qquad \bar{w} = \beta \Sigma \sum_{\mu=1}^{P} t^\mu \Phi\left(x^\mu\right) \tag{16.0.8}$$

The mean predictor is straightforward to calculate,

$$\bar{f}(x) \equiv \int f(x;w) p(w|D, \Gamma) dw = \bar{w}^T \Phi\left(x\right).$$

Similarly, error bars are straightforward, $\text{var}(f(x)) = \Phi(x)^T S \Phi(x)$ (predictive standard errors are given by $\sqrt{\text{var}(f) + \sigma^2}$). In fig(16.1b), we show the mean prediction on the data in fig(16.1a) using 15 Gaussian basis functions with width $\lambda = 0.03$ spread out evenly over the input space. We set the other hyperparameters to be $\beta = 100$ and $\alpha = 1$. The prediction severely overfits the data, a result of a poor choice of hyperparameters.

Determining Hyperparameters: ML-II

How would the mean predictor be calculated if we were to include the hyperparameters Γ as part of a hierarchical model? Formally, this becomes

$$\bar{f}(x) = \int f(x;w)p(w,\Gamma|D)dwd\Gamma = \int \left\{ \int f(x;w)p(w|\Gamma,D)dw \right\} p(\Gamma|D)d\Gamma \tag{16.0.9}$$

The term in curly brackets is the mean predictor for fixed hyperparameters. We therefore weight each mean predictor by the posterior probability of the hyperparameter $p(\Gamma|D)$. Equation (16.0.9) shows how to combine different models in an ensemble – each model prediction is weighted by the posterior probability of the model. There are other non-Bayesian approaches to model combination in which the determination of the combination coefficients is motivated heuristically.

Provided the hyperparameters are well determined by the data, we may instead approximate the above hyperparameter integral by finding the MAP hyperparameters $\Gamma^* = \arg\max_\Gamma p(\Gamma|D)$. Since $p(\Gamma|D) = p(D|\Gamma)p(\Gamma)/p(D)$, if the prior belief about the hyperparameters is weak ($p(\Gamma) \approx \mathit{const.}$), we can estimate the optimal hyperparameters by optimising the hyperparameter likelihood

$$p(D|\Gamma) = \int p(D|\Gamma,w)p(w|\Gamma)dw \tag{16.0.10}$$

This approach to setting hyperparameters is called 'ML-II' [36, 30] and assumes that we can calculate the integral in equation (16.0.10). In the case of GLMs, this involves only Gaussian integration, giving

$$2\log p(D|\Gamma) = -\beta \sum_{\mu=1}^{P} (t^\mu)^2 + d^T S^{-1} d - \log|S| + k\log\alpha + P\log\beta + \mathit{const.} \tag{16.0.11}$$

where $d = \beta \sum_\mu \Phi(x^\mu)t^\mu$. Using the hyperparameters α, β, λ that optimise the above expression gives the results in fig(16.1c) where we plot both the mean predictions and standard predictive error bars. This solution is more acceptable than the previous one in which the hyperparameters were not optimised, and demonstrates that overfitting is avoided automatically. A non-Bayesian approach to model fitting based on minimsing a regularised training error would typically use a procedure such as cross validation to determine the regularisation parameters (hyperparameters). Such approaches require the use of validation data[36]. An advantage of the Bayesian approach is that hyperparameters can be set without the need for validation data, and thus all the data can be used directly for training.

The Kernel Trick

(See also the section on logistic regression). We can write the solution to w in the form

$$w = \sum_\mu \alpha_\mu \boldsymbol{\phi}(x^\mu)$$

And hence the scalar product $w^T\boldsymbol{\phi}(x)$ is of the form

$$\sum_\mu \alpha_\mu \underbrace{\boldsymbol{\phi}^T(x)\boldsymbol{\phi}(x^\mu)}_{K(x,x^\mu)}$$

This means we can just use the kernels K throughout, and use the α_μ as the parameters. It's an analogous treatment as for classification.... One point to bear in mind though is that the predictions usually will decay to zero away from the data (this depends on the choice of the kernel, but is usually the case). This means that we will predict very confidently that the regression should be zero, far from the training data[1]. This is not really what we want – we want to be highly *uncertain* away from the training data. This isn't a problem if we use finite basis functions $\boldsymbol{\phi}$ which are non-local, for example they grow to infinity at infinity. To be continued...relationships to Gaussian Processes.

Relation to Gaussian Processes

The use of GLMs can be difficult in cases where the input dimension is high since the number of basis functions required to cover the input space fairly well grows exponentially with the input dimension – the so called 'curse of dimensionality'[36]. If we specify n points of interest $x^i, i \in 1, \ldots n$ in the input space, the GLM specifies an n-dimensional Gaussian distribution on the function values $f_1, \ldots, f_n$ with mean $\bar{f}_i = \bar{w}^T \Phi\left(x^i\right)$ and covariance matrix with elements $c_{ij} = c(x^i, x^j) = \Phi\left(x^i\right)^T \Sigma \Phi\left(x^j\right)$. The idea behind a GP is that we can free ourselves from the restriction to choosing a covariance function $c(x^i, x^j)$ of the form provided by the GLM prior – any valid covariance function can be used instead. Similarly, we are free to choose the mean function $\bar{f}_i = m(x^i)$. A common choice for the covariance function is $c(x^i, x^j) = \exp\left(-|x^i - x^j|^2\right)$. The motivation is that the function space distribution will have the property that for inputs x^i and x^j which are close together, the outputs $f(x^i)$ and $f(x^j)$ will be highly correlated, ensuring smoothness. This is one way of obviating the curse of dimensionality since the matrix dimensions depend on the number of training points, and not on the number of basis functions used. However, for problems with a large number of training points, computational difficulties can arise, and approximations again need to be considered.

16.1 Problems

Exercise 57 *The question relates to Bayesian regression.*

- *Show that for*

$$f = w^T x$$

and $p(w) \sim \mathcal{N}(0, \Sigma)$, that $p(f|x)$ is Gaussian distributed. Furthermore, find the mean and covariance of this Gaussian.

- *Consider a target point t which is related to the function f by additive noise σ^2. What is $p(f|t,x)$? Hint : use $p(f|t,x) \propto p(t|f,x)p(f|x)$.*

16.2 Solutions

57

[1] For classification, this isn't a problem since the argument of the sigmoid function goes to zero, which means that there is complete uncertainty in the class prediction.

17 Logistic Regression

17.1 Introduction

We've talked about using Generative Models to do classification. Now we look at a discriminative approach.

A common application of machine learning is to classify a novel instance x as belonging to a particular class. Here we concentrate on only two class problems. Explicitly, we are given some training data, $D = \{(x^\mu, t^\mu), \mu = 1 \dots P\}$, where the targets $c \in \{0, 1\}$. An example is given is given in fig(17.1) in which the training inputs x are two dimensional real values, and the associated target values are plotted.

We need to make an assignment for a novel point x to one of the two classes. More generally, we can assign the probability that a novel input x belongs to class 1

$$p(c = 1|x) = f(x; w) \tag{17.1.1}$$

where f is some function parameterised by w. Since the function $f(x)$ represents a probability, $f(x)$ must be bounded between 0 and 1.

In previous chapters we have used class conditional density estimation and Bayes rule to form a classifier $p(c|x) \propto p(x|c)p(c)$. Here, we take the direct approach and postulate a model explicitly for $p(c|x)$. There are advantages and disadvantages in both of these approaches – my personal favourite is to try the indirect approach more often than the direct approach.

Logistic Sigmoid Function

One of the simplest choices of function is the sigmoid function, $f(x) = 1/(1 + \exp(x))$, which is plotted in fig(17.2). What about the argument of the function f? Logistic regression corresponds to the choice

$$p(c = 1|x) = \sigma(b + x^T w) \tag{17.1.2}$$

where b is a constant scalar, and w is a constant vector. When the argument of the sigmoid function $b + x^T w$ is above zero, the probability that the input point x belongs to class 1 is above 0.5. The greater the argument value is, the higher is the probability that x is in class 1 (according to our logistic regresssion model). Similarly, the more negative is the argument, the more likely it is that x belongs to class 0.

Linear (Hyperplane) Decision Boundary

The hyperplane $b + x^T w = 0$ forms the decision boundary (where $p(c = 1|x) = 0.5$) – on the one side, examples are classified as 1's, and on the other, 0's. The "bias" parameter b simply shifts the decision boundary by a constant amount. The orientation of the decision boundary is determined by w – indeed, w represents the

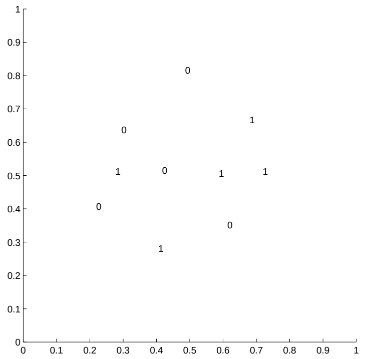


Figure 17.1:

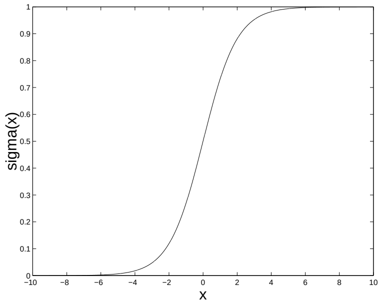


Figure 17.2: The logistic sigmoid function $\sigma(x) = 1/(1 + e^{-x})$.

normal to the hyperplane. To understand this, consider a new point $x^* = x + w^\perp$, where $w^\perp$ is a vector perpendicular to w ($w^T w^\perp = 0$). Then

$$b + w^T x^* = b + w^T \left(x + w^\perp \right) = b + w^T x + w^T w^\perp = b + w^T x = 0 \qquad (17.1.3)$$

Classification confidence

Thus if x is on the decision boundary, so is x plus any vector perpendicular to w. In n dimensions, the space of vectors that are perpendicular to w occupy an $n-1$ dimensional linear subspace, in otherwords an $n-1$ dimensional hyperplane. For example, if the data is two dimensional, the decision boundary is a one dimensional hyperplane, a line. This situation is depicted in fig(17.3). If all the training data for class 1 lie on one side of the line, and for class 0 on the other, the data is said to be *linearly separable*. We plot $\sigma(b + x^T w)$ for different values of w in fig(17.4) and fig(17.5). The decision boundary is at $\sigma(x) = 0.5$. Note how the classification becomes more confident as the size of the weight vector components increases – that is, as we move only a short distance away from the decision boundary, we predict very confidently the class of x if the weights are large.

The Perceptron

As we have defined it sofar, x is assigned to class 1 with some probability. It is not certainly in class 1 unless $p(c = 1|x) = 1$, which cannot happen unless the weights tend to infinity.

The perceptron is a historical simpler model in which x is assigned to class 1 with complete certainty if $b + w^T x \geq 0$, and to class 0 otherwise. Alternatively, we can define a new rule :

$$p(c = 1|x) = \theta(b + x^T w) \qquad (17.1.4)$$

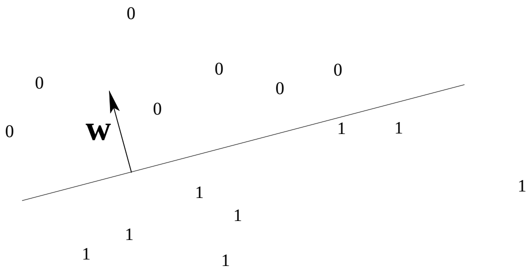


Figure 17.3: The decision boundary $p(c = 1|x) = 0.5$ (solid line). For two dimensional data, the decision boundary is a line. If all the training data for class 1 lie on one side of the line, and for class 0 on the other, the data is said to be *linearly separable.*

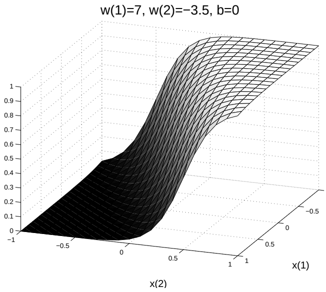


Figure 17.4: The logistic sigmoid function $\sigma(x) = 1/(1 + e^{-x})$, with $x = w^T x + b$.

where the "theta" function is defined as $\theta(x) = 1$ if $x \geq 0$, and $\theta(x) = 0$ if $x < 0$. Since the perceptron is just a special case (the deterministic limit) of logistic regression, we develop here training algorithms for the more general case.

17.1.1 Training

Given a data set D, how can we adjust/"learn" the weights to obtain a good classification? Probabilistically, if we assume that each data point has been drawn independently from the same distribution that generates the data (the standard

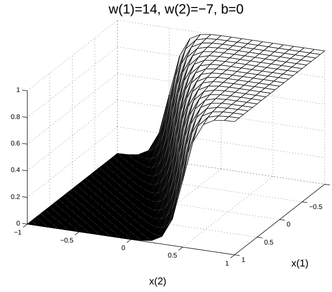


Figure 17.5: The logistic sigmoid function $\sigma(x) = 1/(1 + e^{-x})$, with $x = w^T x + b$.

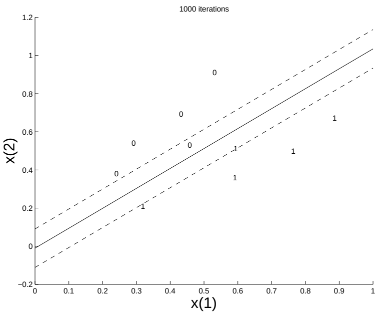


Figure 17.6: The decision boundary $p(c = 1|x) = 0.5$ (solid line) and confidence boundaries $p(c = 1|x) = 0.9$ and $p(c = 1|x) = 0.1$.

i.i.d assumption), the likelihood of the observed data is[1]

$$p(D) = \prod_{\mu=1}^{P} p(c^\mu|x^\mu) = \prod_{\mu=1}^{P} (p(c = 1|x^\mu))^{c^\mu} (1 - p(c = 1|x^\mu))^{1-c^\mu} \qquad (17.1.5)$$

Thus the log likelihood is

$$L = \sum_{\mu=1}^{P} c^\mu \log p(c = 1|x^\mu) + (1 - c^\mu) \log (1 - p(c = 1|x^\mu)) \qquad (17.1.6)$$

Using our assumed logistic regression model, this becomes

$$L(w, b) = \sum_{\mu=1}^{P} c^\mu \log \sigma(b + w^T x^\mu) + (1 - c^\mu) \log \left(1 - \sigma(b + w^T x^\mu)\right) \qquad (17.1.7)$$

17.1.2 Gradient Ascent

We wish to maximise the likelihood of the observed data. To do this, we can make use of gradient information of the likelihood, and then ascend the likelihood.

[1] Note that this is not quite the same strategy that we used in density estimation. There we made, for each class, a model of how x is distributed. That is, given the class c, make a model of x, $p(x|c)$. We saw that, using Bayes rule, we can use $p(x|c)$ to make class predictions $p(c|x)$. Here, however, we assume that, given x, we wish to make a model of the class probability, $p(c|x)$ directly. This does not require us to use Bayes rule to make a class prediction. Which approach is best depends on the problem, but my personal feeling is that density estimation $p(x|c)$ is worth considering first.

The gradient is given by (using $\sigma'(x) = \sigma(x)(1 - \sigma(x))$)

$$\nabla_w L = \sum_{\mu=1}^{P} (c^\mu - \sigma(x^\mu; w))x^\mu \tag{17.1.8}$$

and the derivative with respect to the biases is

$$\frac{dL}{db} = \sum_{\mu=1}^{P} (c^\mu - \sigma(x^\mu; w)) \tag{17.1.9}$$

Gradient ascent would then give

$$w^{new} = w + \eta \nabla_w L \tag{17.1.10}$$

$$b^{new} = b + \eta dL/db \tag{17.1.11}$$

where η, the *learning rate* is a small scalar chosen small enough to ensure convergence of the method (a reasonable guess is to use $\eta = 0.1$). The application of the above rule will lead to a gradual increase in the log likelihood.

Batch version Writing the above result out in full gives explicitly

$$w^{new} = w + \eta \sum_{\mu=1}^{P} (c^\mu - \sigma(x^\mu; w))x^\mu \tag{17.1.12}$$

$$b^{new} = b + \eta \sum_{\mu=1}^{P} (c^\mu - \sigma(x^\mu; w)) \tag{17.1.13}$$

This is called a "batch" update since the parameters w and b are updated only after passing through the whole (batch) of training data – see the MATLAB code below which implements the batch version (note that this is not written optimally to improve readability). We use a stopping criterion so that if the gradient of the objective function (the log likelihood) becomes quite small, we are close to the optimum (where the gradient will be zero), and we stop updating the weights.

Online version An alternative that is often preferred to Batch updating, is to update the parameters after each training example has been considered:

$$w^{new} = w + \frac{\eta}{P}(c^\mu - \sigma(x^\mu; w))x^\mu \tag{17.1.14}$$

$$b^{new} = b + \frac{\eta}{P}(c^\mu - \sigma(x^\mu; w)) \tag{17.1.15}$$

These rules introduce a natural source of stochastic (random) type behaviour in the updates, and can be useful in avoiding local minima. However, as we shall see below, the error surface for logistic regression is bowl shaped, and hence there are no local minima. However, it is useful to bear in mind the online procedure for other optimisation problems with local minima.

```
% Learning Logistic Linear Regression Using Gradient Ascent (BATCH VERSION)

n0 = 16; x0 = randn(2,n0) + repmat([1 -1]',1,n0); % training data for class 0
n1 = 11; x1 = randn(2,n1) + repmat([-1 1]',1,n1); % training data for class 1

eta = 0.1; % learning rate
w = [0 0]'; b = 0; % initial guess about the parameters
it = 0; itmax = 1000;  % maximum number of iterations
gb = 1; gw = zeros(size(w)); % set gradients initally to ensure at least update

while sum(abs(gw)) + abs(gb) > 0.1  % continue whilst gradient is large

  it = it + 1;  % increment the number of updates carried out
  gb = 0; gw = 0*gw; % reset gradients to zero

  for d = 1:n1 % cycle through the class 1 data
    c = 1 - 1/(1+exp(-(b+w'*x1(:,d))));
    gb = gb + c;
    gw = gw + c*x1(:,d);
  end

  for d = 1:n0 % cycle through the class 0 data
    c = 0 - 1/(1+exp(-(b+w'*x0(:,d))));
    gb = gb + c;
    gw = gw + c*x0(:,d);
  end

  w = w + eta*gw; % update the weight vector
  b = b + eta*gb; % update the bias scalar

  if it > itmax; break; end
end

% calculate the probabilities p(c=1|x) for the training data :
disp('p(c=1|x) for class 1 training data : ');
1./(1+exp(-(repmat(b,1,n1)+w'*x1)))

disp('p(c=1|x) for class 0 training data : ');
1./(1+exp(-(repmat(b,1,n0)+w'*x0)))
```

One important point about the training is that, provided the data is linearly separable, the weights will continue to increase, and the classifications will become extreme. This may be an undesirable situation in case some of the training data has been mislabelled, or a test point needs to be classified – it is rare that we could be absolutely sure that a test point belongs to a particular class. For non-linearly separable data, the predictions will be less certain, as reflected in a broad confidence interval – see fig(17.7).

The error surface is bowl-shaped

The Hessian of the log likelihood is

$$H_{ij} \equiv \frac{\partial^2 H}{\partial w_i w_j} = -\sum_{\mu} x_i^{\mu} x_j^{\mu} \sigma^{\mu}(1-\sigma^{\mu}) \qquad (17.1.16)$$

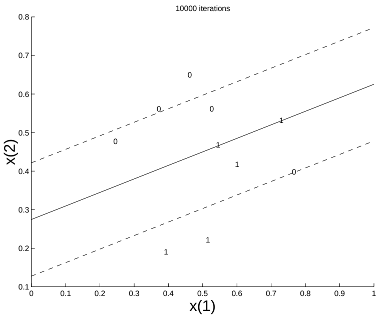


Figure 17.7: The decision boundary $p(c = 1|x) = 0.5$ (solid line) and confidence boundaries $p(c = 1|x) = 0.9$ and $p(c = 1|x) = 0.1$ for non-linearly separable data. Note how the confidence interval remains broad.

This is negative definite since

$$\sum_{ij} w_i H_{ij} w_j = -\sum_{i,j,\mu} w_i x_i^\mu w_j x_j^\mu \sigma^\mu (1-\sigma^\mu) = -\left(\sum_{i,\mu} w_i x_i^\mu\right)^2 \sigma^\mu (1-\sigma^\mu) \quad (17.1.17)$$

This means that the error surface has a bowl shape, and gradient ascent is guaranteed to find the best solution, provided that the learning rate η is small enough.

Perceptron Convergence Theorem

One can show that, provided that the data is linearly separable, the above procedure used in an online fashion for the perceptron (replacing $\sigma(x)$ with $\theta(x)$) converges in a finite number of steps. The details of this proof are not important for this course, but the interested reader may consult *Neural Networks for Pattern Recognition*, by Chris Bishop. Note that the online version will not converge if the data is not linearly separable. The batch version will converge (provided that the learning rate η is small) since the error surface is bowl shaped.

17.1.3 Avoiding Overconfident Classification

We saw that in the case that data is linearly separable, the weights will tend to increase indefinitely (unless we use some stopping criterion). One way to avoid this is to penalise weights that get too large. This can be done by adding a penalty term to the objective function $L(\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is a vector of all the parameters, $\boldsymbol{\theta} = (w, b)$,

$$L'(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) - \alpha \boldsymbol{\theta}^T \boldsymbol{\theta}. \quad (17.1.18)$$

The scalar constant $\alpha > 0$ encourages smaller values of $\boldsymbol{\theta}$ (remember that we wish to maximise the log likelihood). How do we choose an appropriate value for α? We shall return to this issue in a later chapter on generalisation.

17.1.4 Logistic Regression and PCA ?

In previous chapters, we have looked at first using PCA to reduce the dimension of the data, so that a high dimensional datapoint x is represented by a lower dimensional vector y.

If $e^1, \dots, e^m$ are the eigenvectors with largest eigenvalues of the covariance matrix of the high-dimensional data, then the PCA representation is

$$y_i = (e^i)^T (x - c) = (e^i)^T x + a_i \quad (17.1.19)$$

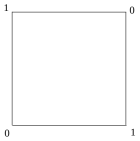


Figure 17.8: The XOR problem. This is not linearly separable.

where c is the mean of the data, and a_i is a constant for each datapoint. Using vector notation, we can write

$$y = E^T x + a \tag{17.1.20}$$

where E is the matrix who's i^{th} column is the eigenvector e^i. If we were to use logistic regression on the y, the argument of the sigmoid $\sigma(h)$ would be

$$h = w^T y + b = w^T(E^T x + a) + b \tag{17.1.21}$$

$$= (Ew)^T x + b + w^T a = \tilde{w}^T x + \tilde{b} \tag{17.1.22}$$

Hence, there is nothing to be gained by first using PCA to reduce the dimension of the data. Mathematically, PCA is a linear projection of the data. The argument of the logistic function is also a linear function of the data, and a linear function combined with another is simply another linear function.

However, there is a subtle point here. If we use PCA first, then use logistic regression afterwards, although overall, this is still representable as a logistic regression problem, the problem is *constrained* since we have forced linear regression to work in the subspace spanned by the PCA vectors. Consider 100 training vectors randomly positioned in a 1000 dimensional space each with a random class 0 or 1. With very high probability, these 100 vectors will be linearly separable. Now project these vectors onto a 10 dimensional space: with very high probability, 100 vectors plotted in a 10 dimensional space will *not* be linearly separable. Hence, arguably, we should not use PCA first since we could potentially transform a linearly separable problem into a non-linearly separable problem.

The XOR problem

Consider the following four training points and class labels
$\{([0,0],0),([0,1],1),([1,0],1),([1,1],0)\}$.

This data represents a basic logic function, the XOR function, and is plotted in fig(17.8). This function is clearly not representable by a linear decision boundary, an observation much used in the 1960's to discredit work using perceptrons. To overcome this, we clearly need to look at methods with more complex, non-linear decision boundaries – indeed, we encountered a quadratic decision boundary in a previous chapter. Historically, another approach was used to increase the complexity of the decision boundary, and this helped spawn the area of neural networks, to which we will return in a later chapter.

17.1.5 An Example : Classifying Handwritten Digits

If we apply logistic regression to our often used handwritten digits example, in which there are 300 ones, and 300 sevens in the training data, and the same number in the test data, the training data is found to be linearly separable. This may surprise you, but consider that there are 784 dimensions, and only 600 training points. The stopping criterion used was the same as in the example MATLAB code in this chapter. Using the linear decision boundary, the number of errors made on the 600 test points is 12.

17.2 The Kernel Trick

The simple logistic method of doing discriminative classification is very popular. In particular, one of the major benefits is the speed and simplicity of the training algorithm. However, one major drawback is the simplicity of the decision surface – a hyperplane. One way to improve the method is to consider mapping the inputs x in a non-linear way to $\boldsymbol{\psi}(x)$:

$$p(c = 1|x) = \sigma\left(w^T \boldsymbol{\psi}(x)\right)$$

(adding a bias is a trivial extra modification). Note that $\boldsymbol{\psi}(x)$ does not have to be of the same dimension as w. For example, the one-dimensional input x could get mapped to a two dimensional vector $(x^2, \sin(x))$. (Lower dimensional mappings are also possible, but less popular since this can make it more difficult to find a simple classifier). The usual motivation for this is that mapping into a high dimensional space makes it easier to find a separating hyperplane in the high dimensional space (remember that any set of points that are independent can be linearly separated provided we have as many dimensions as datapoints – this motivates the use of non-linear mappings since the related high-dimensional datapoints will then usually be independent).

If we wish to use the ML criterion, we can use exactly the same algorithm as in the standard case, except wherever there was a x before, this gets replaced with $\boldsymbol{\psi}(x)$.

Is there another way to do training? Informally, if we assume that we begin with the zero vector, and make updates according to the gradient ascent rule, then the updates are of the form

$$w^{new} = w + \sum_{\mu} \gamma(w)\boldsymbol{\psi}(x^{\mu})$$

where γ is some scalar function. The point is that, by iterating the above equation, any solution will therefore be of the form of a linear combination of the points $\boldsymbol{\psi}^{\mu}$, where for simplicity, we write $\boldsymbol{\psi}^{\mu} \equiv \boldsymbol{\psi}(x^{\mu})$. Hence, we may assume a solution

$$w = \sum_{\mu} \alpha_{\mu} \boldsymbol{\psi}^{\mu}$$

and try to find a solution in terms of the vector of parameters α_{μ}. This is potentially advantageous since there may be less training points than dimensions of $\boldsymbol{\psi}$. The classifier depends only on scalar products

$$w^T \boldsymbol{\psi}(x) = \sum_{\mu} \alpha_{\mu} \boldsymbol{\psi}(x^{\mu})^T \boldsymbol{\psi}(x)$$

Hence, the only role that $\boldsymbol{\psi}$ plays is in the form of a scalar product:

$$K(x, x') = \boldsymbol{\psi}(x)^T \boldsymbol{\psi}(x')$$

Since the right is a scalar product, it defines a positive definite (kernel) function (see section (E)). These are symmetric functions for which, roughly speaking, the corresponding matrix defined on a set of points $x^i, i = 1, \ldots, N$ is positive define. Indeed, Mercer's Theorem states that a function defines a positive definite kernel function if and only if it has such an inner product representation. What this means is that we are then free to define a function which is positive definite kernel function, since this is the only thing that the classifier depends on. (This is well established in classical statistics, and forms the basis of Gaussian Processes – see later chapter). Hence, we can define

$$p(c = 1|x) = \sigma\left(\sum_{\mu} \alpha_{\mu} K(x, x^{\mu})\right)$$

For convenience, we can write the above as

$$p(c = 1|x) = \sigma\left(\boldsymbol{\alpha}^T k(x)\right)$$

where the P dimensional vector $k(x)$ has elements $[k(x)]_{\mu} = K(x, x^{\mu})$. Then the above is of exactly the same form as the original specification of logistic regression, namely as a function of a linear combination of vectors. Hence the same training algorithm to maximise the likelihood can be employed.

For example

$$K(x, x') = e^{-\lambda\left(x-x'\right)^2}$$

defines a positive definite kernel (see problems).

It should be clear from that above that essentially *any* method which depends on a scalar product $w^T \boldsymbol{\psi}(x)$, or indeed, possibly a set of such scalars, can be kernelised. Recently, a small industry producing kernelised methods has been in action, based on the same basic idea.

Support Vector Machines

The realisation that the higher the dimension of the space is, the easier it is to find a hyperplane that linearly separates the data, forms the basis for the Support Vector Machine method. The main idea (contrary to PCA) is to map each vector in a much *higher* dimensional space, where the data can then be linearly separated. Training points which do not affect the decision boundary can then be discarded. We will not go into the details of how to do this in this course, but the interested reader can consult `http://www.support-vector.net`. Related methods currently produce the best performance for classifying handwritten digits – better than average human performance. Essentially, however, the distinguishing feature of the SVM approach is not in the idea of a high-dimensional projection, but rather in the manner of finding the hyperplane. The idea is to find the hyperplane such that the distance between the hyperplane and (only) those points which determine the placement of the plane, should be maximal. This is a quadratic programming problem. In this case, usually only a small set of the training data effects the decision boundary. However, this method is not probabilistic, and no satisfactory manner of formulating the SVM directly as a probabilistic model has been achieved (although there have been numerous approaches, all of which contain a fudge somewhere). More later......

A similar method which retains the benefits of a probabilistic analysis is the Relevance Vector Machine.

Are Kernels really necessary?

In the above, we saw how to define classifier

$$p(c=1|x) = \sigma\left(\sum_{\mu} \alpha_\mu K(x, x^\mu)\right)$$

In the case that K is a Kernel, we can interpret this as essentially fitting a hyperplane through a set of points, where the points are the data training points projected into a (usually) higher dimensional space. However, if one does not require this condition, we can define a more general classifier

$$p(c=1|x) = \sigma\left(\sum_{i} \alpha_i K_i(x)\right)$$

where the $K_i(x), i = 1, \ldots, F$ are a fixed set of functions mapping the vector x to a scalar. For example, if we set $K_i(x) = \tanh(x^T w^i)$, and treat also w^i as a parameter, the solution will not be representable in a Kernel way. In these more general settings, training is more complex, since the error surface cannot be guaranteed to be convex, and simple gradient ascent methods (indeed, any optimisation method) will potentially get trapped in a local optimum. In this sense, Kernels are useful since they mean we can avoid training difficulties.

As an aside, consider if we set $K_i(x) = \tanh(x^T w^i)$, for *fixed* w^i, and treat only the α_i as adjustable parameters, then the solution *is* representable as a Kernel (since the argument of the sigmoid is representable as a scalar product between a parameter vector and a fixed vector function of x).

17.3 Mixture Models

The big advantage of the 'kernelised' verions of logistic regression is that it is probabilistic. This means that we can do a Bayesian style analysis for training (more later). Also, we can do things such as mixtures, and mixtures of experts.

17.3.1 Mixtures

How can we increase the power of the above methods? One way to do this is to write

$$p(c=1|x) = \sum_{h=1}^{H} p(c=1, h|x) = \sum_{h=1}^{H} p(c=1|h, x)p(h|x)$$

Usually, the hidden variable h is taken to be discrete. Here, then $p(c=1|h,x)$ is one of a set of H classifiers.

In a standard *mixture model*, we assume independence, $p(h|x) = p(h)$.

17.3.2 Mixture of Experts

In a *mixture of experts* (cite Jordan) model, we assume that $p(h|x)$ has some parametric form, for example using a softmax function

$$p(h|x) = \frac{e^{(w^h)^T x}}{\sum_{h'} e^{(w^{h'})^T x}}$$

In both cases, the natural way to train them is to use the variational EM approach, since the h is a hidden variable.

Note that these methods are completely general, and not specific to logistic regression. To do... example of mixture of experts applied to handwritten digits.

17.3.3 A 'Bayesian' approach to setting the regularisation parameter

In a previous section, we mentioned a problem with the ML criterion (remember that this is just a heuristic). How might we correct for this problem? Well, let's go back to basics, and see what we can do. From Bayes' rule, we know that we are ultimately interested in $p(w|D)$, and that this is proportional to $p(D|w)p(w)$. It's clear that the assumed flat prior on $p(w)$ (which is related to the ML heuristic) is letting us down, since very large values of w will give rise to over-confident classifications. From the previous discussions, we know that we can kernelise the logistic method to consider in more generality,

$$p(c=1|x) = \sigma\left(\sum_{\mu'=1}^{P} w_{\mu'} K(x, x^{\mu'}) + b\right)$$

(In the following, I'll set b to zero, just for notational clarity).

Note: a potential confusion here with notation. I'm now using w where previously I used α. Now α refers to the precision.

How can we prevent the classifications becoming too severe? If the Kernel values themselves are bounded (for the squared exponential kernel, this is clearly the case), then putting a soft constraint on the size of the components w_μ will discourage overly confident classifications.

A convenient prior $p(w)$ one is to use a Gaussian constraint:

$$p(w|\alpha) = \frac{\alpha^{P/2}}{(2\pi)^{P/2}} e^{-\alpha w^T w/2}$$

where α is the inverse variance (also called the *precision*) of the Gaussian distribution. (Remember that here the dimension of w is equal to the number of training points).

More formally, we could put another distribution on $p(\alpha)$, say a Gamma distribution (see section (C)), as part of an hierarchical prior.

$$p(w) = \int_\alpha p(w|\alpha)p(\alpha) = \int e^{-\frac{\alpha}{2} w^T w} \alpha^{\gamma-1} e^{-\alpha/\beta} d\alpha$$

It's clear that the RHS is another Gamma distribution (the Gaussian and Gamma distributions are conjugate). Indeed (exercise) the reader can easily show that the distribution of w is a t-distribution.

Here we'll generally keep life simple, and assume the above 'flat' prior on α. We have therefore a GM of the form

$$p(w, \alpha|D) = \frac{1}{Z} p(w|\alpha)p(\alpha) \prod_{\mu=1}^{P} p(c^\mu|x^\mu, w)$$

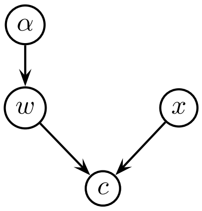


Figure 17.9: Graphical representation of logistic regression.

where the constant Z ensures normalisation,

$$Z = \int_{\alpha,w} p(w|\alpha)p(\alpha)\prod_{\mu=1}^{P} p(c^\mu|x^\mu, w).$$

The above suggests the following strategies:

Full Bayesian Ultimately, we'll be interested in using the classifier in novel situations. In this sense, we would like to compute

$$p(c = 1|x, D) = \int_{w,\alpha} p(c = 1|x, w)p(w, \alpha|D)$$

However, the P dimensional integrals over w cannot be analytically calculated. This means that approximations need to be considered. One approach would be to draw samples from $p(w, \alpha|D)$ (see section (26)). Whilst this is relatively straightforward, due to the relative benign nature of the posterior in this particular case, we may as well exploit the relative simplicity of the posterior to make an analytic based approximation.

Variational Method We could fit a variational distribution to the posterior. Fairly straightforward. Blah.... Not really necessary in this case since the posterior is simple.

Laplace Approximation If we have enough training datapoints, we might expect the posterior distribution to become reasonably well peaked around a most-probably value of w. This is even more the case for α, since this is just a one dimensional variable – almost certainly, given a reasonable amount of training data, the posterior distribution $p(\alpha|D)$ will be very sharply peaked.

In the following, we'll take the path of least resistance, and use the simplest approximation, the Laplace method. This is justifiable in this case since the posterior $p(w|\alpha, D)$ is unimodal. Historically, this corresponds to one of the earliest applications of approximate Bayesian methods in machine learning[35].

17.3.4 Evidence Procedure

What about the setting of $p(\alpha)$? If we assume a flat prior on α, this will effectively mean that we favour smaller values of the variance, and hence small values of the weights. In this case, finding the α that maximises $p(\alpha|D)$ is called ML-II estimation (we don't use ML at the first level to determine w, but rather use ML at the second, hyperparameter level).

Alternatively, we notice that we can integrate out analytically (usually) over the one-dimensional α to obtain

$$p(w|D) = \frac{1}{Z} p(w) \prod_{\mu=1}^{P} p(c^\mu|x^\mu, w)$$

where $p(w) = \int p(w|\alpha)p(\alpha)d\alpha$ and

$$Z = \int_w p(w) \prod_{\mu=1}^{P} p(c^\mu|x^\mu, w)$$

The main difficulty in both approaches above is that we cannot analytically integrate over the w since the distribution, and we are forced to make an approximation. The best way to make an approximation has been a topic of some intense debate. Should we integrate out the hyperparameter distribution first, and then attempt to approximate the posterior distribution $p(w|D)$, or approximate the joint distribution $p(w, \alpha|D)$? Since we have a good idea that $p(\alpha|D)$ will be sharply peaked, and $p(w|\alpha, D)$ is unimodal, the argument goes that it makes sense to make the simple unimodal Laplace approximation on the simple $p(w|\alpha, D)$, rather than the more complex $p(w|D)$.

$$p(\alpha|D) \propto p(\alpha) \int_w p(w|\alpha) \prod_\mu \sigma\left((2c^\mu - 1)(w^T k^\mu)\right)$$

where $[k^\mu]_i \equiv K(x^\mu, x^i)$. A simple Laplace approximation section (F) gives

$$\log p(\alpha|D) \approx \log p(\alpha) - E(w^*) - \frac{1}{2}\log\det 2\pi H + \frac{P}{2}\log\alpha + const.$$

and

$$E(w) = \frac{\alpha}{2} w^T w - \sum_{\mu=1}^{P} \log\sigma\left(w^T h^\mu\right)$$

$h^\mu = (2c^\mu - 1)k^\mu$. The Laplace approximation states that we need to find the minimum of $E(w)$. Differentiating, we get

$$\nabla E = \alpha w - \sum_\mu (1 - \sigma^\mu) h^\mu$$

where $\sigma^\mu \equiv \sigma\left(w^T h^\mu\right)$. We could then use a simple gradient descent algorithm. However, since the surface is convex, and the Hessian is simple to calculate,

$$H(w) = \alpha I + \underbrace{\sum_{\mu=1}^{P} \sigma^\mu(1 - \sigma^\mu) h^\mu \left(h^\mu\right)^T}_{J}$$

we may as well use a Newton update:

$$w^{new} = w - H^{-1}\left(\nabla E\right) \tag{17.3.1}$$

Once this iteration has converged to a value w^*, we are in a position to approximate the likelihood. Since, ultimately, we want to calculate

$$\frac{dL}{d\alpha} = \frac{\partial L}{\partial\alpha} + \underbrace{\frac{\partial L}{\partial w^*}}_{=0} \frac{\partial w^*}{\partial\alpha}$$

to optimise L with respect to α, we only need consider the terms with an explicit α dependence,

$$L(\alpha) \approx -\frac{\alpha}{2}(w^*)^T w^* - \frac{1}{2}\log\det(\alpha I + J) + \frac{P}{2}\log\alpha + const.$$

Differentiating wrt α, using $\partial \log\det(M) = \mathrm{trace}\left(M^{-1}\partial M\right)$, and setting to zero, we can make a fixedpoint iteration

$$\alpha^{new} = \frac{P}{(w^*)^T w^* + \mathrm{trace}\left((\alpha I + J)^{-1}\right)} \tag{17.3.2}$$

We have then the celebrated 'evidence procedure'[35]:

1. Initialise w and α to (sensible!) values.
2. Find w^*. This is achieved here by iterating equation (17.3.1) to convergence.
3. Update α according to equation (17.3.2). (Sometimes slightly different fixed point updates are used.)
4. Iterate steps 2 and 3 until convergence of α.

To make predictions on novel inputs x, we can do the following

$$p(c = 1|x, D) = \int p(c = 1|x, w)p(w|D)dw = \int \sigma(h)\, p(h|x, D)dh$$

where $p(h|x, D)$ is the distribution of the quantity $x^T w$. Under the Laplace approximation, w is Gaussian,

$$p(w^*|D) = N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

where $\boldsymbol{\mu} = w^*$, and $\boldsymbol{\Sigma} = (H(w^*))^{-1}$. This means that h (which is linearly related to w) is also Gaussian distributed

$$p(h|x, D) = N(x^T w^*, x^T \boldsymbol{\Sigma} x)$$

Then, to make predictions, we can do the following

$$p(c = 1|x, D) = \int \sigma(h)\, p(h|x, D)dh$$

The simple one-dimensional integral over h is carried out numerically.

Need to extend the evidence stuff to determining the hyperparameters λ. This is all related to GP classification.

Relevance Vector Machine

One might wonder if all datapoints are equally important in determining the solution of a classification problem. Intuitively, it would be reasonable to think that many points don't really affect the solution, and could be effectively removed, or pruned from the dataset. We can achieve this by repeating the previous evidence framework analysis, but this time, rather than having a global α for the whole weight vector, rather we have a penalty

$$p(w|\boldsymbol{\alpha}) \propto e^{-\sum_{\imath=1}^{P} \alpha_i w_i^2/2}$$

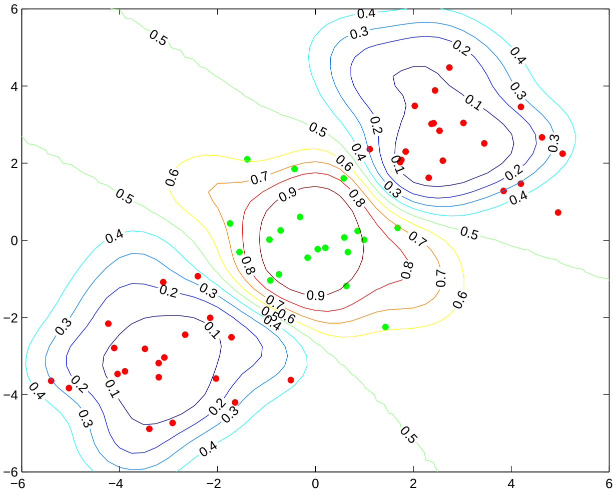


Figure 17.10: An example using Logistic Regression with the squared exponential kernel, $e^{-(x-x')^2}$. The green points are training data from class 1, and the red points are training data from class 0. The contours represent the probability of being in class 1. The optimal value of α found by the evidence procedure in this case is 0.45.

The reader may verify that the only alterations in the previous evidence procedure are simply

$$[\nabla E]_i = \alpha_i w_i - \sum_{\mu}(1-\sigma^{\mu})h_i^{\mu}$$

$$H(w) = diag(\boldsymbol{\alpha}) + J$$

These are used in the Newton update formula as before. The implicit equation for the α's is given by

$$\alpha_i = \frac{1}{w_i^2 + \Sigma_{ii}}$$

where $\boldsymbol{\Sigma} = (H(w))^{-1}$. Running this procedure, one typically finds that many of the α's tend to infinity, and may be effectively pruned from the dataset. Those remaining tend to be rather in the centres of mass of a bunch of datapoints of the same class. Contrast this with the situation in SVMs, where the retained datapoints tend to be on the decision *boundaries*. In that sense, the RVM and SVM have very different characteristics. The number of training points retained by the RVM tends to be very small – smaller indeed that the number retained in the SVM framework. However, the RVM is a little more computationally expensive than SVMs, but otherwise retains the advantages inherited from a probabilistic framework[37]. Naturally enough, one can extend this idea of sparseness to many other probabilistic models, and is a special case of the automatic relevance determination (ARD) method introduced by MacKay and Neal[35]. Finding such sparse representations has obvious applications in compression. A hot research issue to speed up training of RVMs.

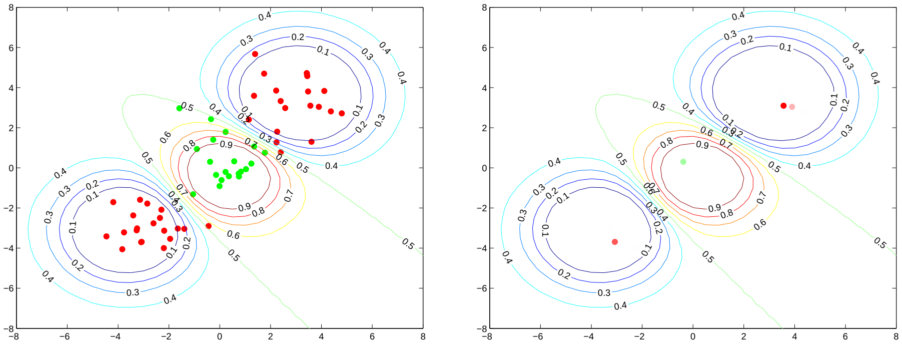


Figure 17.11: An example using RVM classification with the squared exponential kernel, $e^{-(x-x')^2}$. The green points are training data from class 1, and the red points are training data from class 0. The contours represent the probability of being in class 1. On the left are plotted the training points. On the right we plot the training points weighted by their relevance value $1/\alpha_\mu$. Nearly all the points have a value so small that they effectively vanish.

17.4 Problems

Exercise 58 *Show that*

$$K(x,x') = e^{-\lambda\left(x-x'\right)^2}$$

defines a positive definite kernel.

Hint: one simple way to show this is to consider expanding the exponent, and then to consider the properties of the power series expansion of the exponential function).

17.5 Solutions

58

18 Naive Bayes

18.1 Why Naive Bayes?

Naive Bayes is one of the simplest density estimation methods from which we can form one of the standard classification methods in machine learning. Its fame is partly due to the following properties:

- Very easy to program and intuitive
- Fast to train and to use as a classifier
- Very easy to deal with missing attributes
- Very popular in fields such as computational linguistics/NLP

Despite the simplicity of Naive Bayes, there are some pitfalls that need to be avoided, as we will describe. The pitfalls usually made are due to a poor understanding of the central assumption behind Naive Bayes, namely conditional independence.

18.2 Understanding Conditional Independence

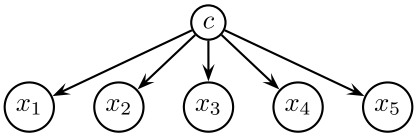
EasySell.com considers that its customers conveniently fall into two groups – the ‘young’ or ‘old’. Based on only this information, they build general customer profiles for product preferences. Easysell.com *assumes* that, given the knowledge that a customer is either ‘young’ or ‘old’, this is *sufficient* to determine whether or not a customer will like a product, *independent* of their likes or dislikes for any other products. Thus, given that a customer is ‘young’, she has a 95% chance to like Radio1, a 5% chance to like Radio2, a 2% chance to like Radio3 and a 20% chance to like Radio4. Similarly, they model that an ‘old’ customer has a 3% chance to like Radio1, an 82% chance to like Radio2, a 34% chance to like Radio3 and a 92% chance to like Radio4. Mathematically, we would write

$$p(R1, R2, R3, R4|age) = p(R1|age)p(R2|age)p(R3|age)p(R4|age)$$

where each of the variables $R1, R2, R3, R4$ can take the values either ‘like’ or ‘dislike’, and the ‘age’ variable can take the value either ‘young’ or ‘old’. Thus the information about the age of the customer is so powerful that this determines the individual product preferences without needing to know anything else. This kind of assumption is indeed rather ‘naive’, but can lead to surprisingly good results.

In this chapter, we will take the conditioning variable to represent the class of the datapoint x. Coupled then with a suitable choice for the conditional distribution $p(x_i|c)$, we can then use Bayes rule to form a classifier. We can generalise the situation of two variables to a conditional independence assumption for a set of variables $x_1, \ldots, x_N$, conditional on another variable c:

$$p(x|c) = \prod_{i=1}^{N} p(x_i|c) \qquad (18.2.1)$$



See fig(18.2) for the graphical model. In this chapter, we will consider two cases of different conditional distributions, one appropriate for discrete data and the other for continuous data. Furthermore, we will demonstrate how to learn any free parameters of these models.

18.3 Are they Scottish?

Consider the following vector of attributes:

$$\text{(likes shortbread, likes lager, drinks whiskey,eats porridge,} \\ \text{watched England play football)}^T \tag{18.3.1}$$

A vector $x = (1, 0, 1, 1, 0)^T$ would describe that a person likes shortbread, does not like lager, drinks whiskey, eats porridge, and has not watched England play football. Together with each vector x^μ, there is a class label describing the nationality of the person: Scottish, or English. We wish to classify a new vector $x = (1, 0, 1, 1, 0)^T$ as either Scottish(S) or English(E). We can use Bayes rule to calculate the probability that x is Scottish or English:

$$p(S|x) = \frac{p(x|S)p(S)}{p(x)}$$

$$p(E|x) = \frac{p(x|E)p(E)}{p(x)}$$

Since we must have $p(S|x) + p(E|x) = 1$, we could also write

$$p(S|x) = \frac{p(x|S)p(S)}{p(x|S)p(S) + p(x|E)p(E)}$$

It is straightforward to show that the “prior” class probability $p(S)$ is simply given by the fraction of people in the database that are Scottish, and similarly $p(E)$ is given as the fraction of people in the database that are English. What about $p(x|S)$? This is where our density model for x comes in. In the previous chapter, we looked at a using a Gaussian distribution. Here we will make a different, very strong *conditional independence* assumption:

$$p(x|S) = p(x_1|S)p(x_2|S) \dots p(x_5|S)$$

What this assumption means is that knowing whether or not someone is Scottish, we don’t need to know anything else to calculate the probability of their likes and dislikes.

Matlab code to implement Naive Bayes on a small dataset is written below, where each row of the datasets represents a (row) vector of attributes of the form equation (18.3.1).

```
% Naive Bayes using Bernoulli Distribution

xE=[0 1 1 1 0 0;  % english
    0 0 1 1 1 0;
    1 1 0 0 0 0;
    1 1 0 0 0 1;
    1 0 1 0 1 0];

xS=[1 1 1 1 1 1 1; % scottish
    0 1 1 1 1 0 0;
    0 0 1 0 0 1 1;
    1 0 1 1 1 1 0;
    1 1 0 0 1 0 0];

pE = size(xE,2)/(size(xE,2) + size(xS,2)); pS =1-pE; % ML class priors pE = p(c=E), pS=p(c=S)

mE = mean(xE')'; % ML estimates of p(x=1|c=E)
mS = mean(xS')'; % ML estimates of p(x=1|c=S)

x=[1 0 1 1 0]'; % test point

npE = pE*prod(mE.^x.*(1-mE).^(1-x)); % p(x,c=E)
npS = pS*prod(mS.^x.*(1-mS).^(1-x)); % p(x,c=S)

pxE = npE/(npE+npS) % probability that x is english
```

Based on the training data in the code above, we have the following : $p(x_1 = 1|E) = 1/2$,$p(x_2 = 1|E) = 1/2$,$p(x_3 = 1|E) = 1/3$,$p(x_4 = 1|E) = 1/2$,$p(x_5 = 1|E) = 1/2$, $p(x_1 = 1|S) = 1$,$p(x_2 = 1|S) = 4/7$,$p(x_3 = 1|S) = 3/7$,$p(x_4 = 1|S) = 5/7$,$p(x_5 = 1|S) = 3/7$ and the prior probabilities are $p(S) = 7/13$ and $p(E) = 6/13$.

For $x^* = (1, 0, 1, 1, 0)^T$, we get

$$p(S|x*) = \frac{1 \times \frac{3}{7} \times \frac{3}{7} \times \frac{5}{7} \times \frac{4}{7} \times \frac{7}{13}}{1 \times \frac{3}{7} \times \frac{3}{7} \times \frac{5}{7} \times \frac{4}{7} \times \frac{7}{13} + \frac{1}{2} \times \frac{1}{2} \times \frac{1}{3} \times \frac{1}{2} \times \frac{1}{2} \times \frac{6}{13}} \tag{18.3.2}$$

which is 0.8076. Since this is greater than 0.5, we would classify this person as being Scottish.

18.3.1 Further Issues

Consider trying to classify the vector $x = (0, 1, 1, 1, 1)^T$. In the training data, all Scottish people say they like shortbread. This means that $p(x, S) = 0$, and hence that $p(S|x) = 0$. This demonstrates a difficulty with sparse data – very extreme class probabilities can be made. One way to ameliorate this situation is to smooth the probabilities in some way, for example by adding a certain small number M to the frequency counts of each class. This ensures that there are no zero probabilities in the model:

$$p(x_i = 1|c) = \frac{\text{number of times } x_i = 1 \text{ for class c} + M}{\text{number of times } x_i = 1 \text{ for class c} + M + \text{number of times } x_i = 0 \text{ for class c} + M} \tag{18.3.3}$$

Continuous Data

Fitting continuous data is also straightforward using Naive Bayes. For example, if we were to model each attributes distribution as a Gaussian, $p(x_i|c) = N(\mu_i, \sigma_i)$, this would be exactly equivalent to using a conditional Gaussian density estimator with a diagonal covariance matrix.

18.3.2 Text Classification

Naive Bayes has been often applied to classify documents in classes. We will outline here how this is done. Refer to a computational linguistics course for details of how exactly to do this.

Bag of words

Consider a set of documents about politics, and a set about sport. We search through all documents to find the, say 100 most commonly occuring words. Each document is then represented by a 100 dimensional vector representing the number of times that each of the words occurs in that document – the so called 'bag of words' representation (this is clearly a very crude assumption since it does not take into account the order of the words). We then fit a Naive Bayes model by fitting a distribution of the number of occurrences of each word for all the documents of, first sport, and then politics.

The reason Naive Bayes may be able to classify documents reasonably well in this way is that the conditional independence assumption is not so silly : if we know people are talking about politics, this perhaps is almost sufficient information to specify what kinds of other words they will be using – we don't need to know anything else. (Of course, if you want ultimately a more powerful text classifier, you need to relax this assumption).

18.4 Pitfalls with Naive Bayes

So far we have described how to implement Naive Bayes for the case of binary attributes and also for the case of Gaussian continuous attributes. However, very often, the software that people seem to commonly use requires that the data is in the form of binary attributes. It is in the transformation of non-binary data to a binary form that a common mistake occurs.

Consider the following attribute : age. In a survey, a person's age is marked down using the variable $a \in 1, 2, 3$. $a = 1$ means the person is between 0 and 10 years old, $a = 2$ means the person is between 10 and 20 years old, $a = 3$ means the person is older than 20. Perhaps there would be other attributes for the data, so that each data entry is a vector of two variables $(a, b)^T$.

1-of-M encoding

One way to transform the variable a into a binary representation would be to use three binary variables (a_1, a_2, a_3). Thus, $(1, 0, 0)$ represents $a = 1$, $(0, 1, 0)$ represents $a = 2$ and $(0, 0, 1)$ represents $a = 3$. This is called $1 - of - M$ coding since only 1 of the binary variables is active in encoding the M states. The problem here is that this encoding, by construction, means that the variables a_1, a_2, a_3 are *dependent* – for example, if we know that $a_1 = 1$, we know that $a_2 = 0$ and $a_3 = 0$. Regardless of any possible conditioning, these variables will always remain completely dependent, contrary to the assumption of Naive Bayes. This mistake, however, is widespread – please help preserve a little of my sanity by not making the same error. The correct approach is to simply use variables with many states

– the multinomial rather than binomial distribution. This is straightforward and left as an exercise for the interested reader.

18.5 Estimation using Maximum Likelihood : Bernoulli Process

Here we formally derive how to learn the parameters in a Naive Bayes model from data. The results are intuitive, and indeed, we have already made use of them in the previous sections. Additionally, some light can be cast on the nature of the decision boundary (at least for the case of binary attributes).

Consider a dataset $X = \{x^\mu, \mu = 1, \dots, P\}$ of binary attributes. That is $x_i^\mu \in \{0, 1\}$. Each datapoint x^μ has an associated class label c^μ. Based upon the class label, we can split the inputs into those that belong to each class : $X^c = \{x|x \text{ is in class } c\}$. We will consider here only the case of two classes (this is called a Bernoulli process – the case of more classes is also straightforward and called the multinomial process). Let the number of datapoints from class $c = 0$ be n_0 and the number from class $c = 1$ be n_1.

For each class of the two classes, we then need to estimate the values $p(x_i = 1|c) \equiv \theta_i^c$. (The other probability, $p(x_i = 0|c)$ is simply given from the normalisation requirement, $p(x_i = 0|c) = 1 - p(x_i = 1|c) = 1 - \theta_i^c$). Using the standard assumption that the data is generated identically and independently, the likelihood of the model generating the dataset X^c (the data X belonging to class c) is

$$p(X^c) = \prod_{\mu \text{ from class c}} p(x^\mu|c) \tag{18.5.1}$$

Using our conditional independence assumption

$$p(x|c) = \prod_i p(x_i|c) = \prod_i (\theta_i^c)^{x_i} (1 - \theta_i^c)^{1-x_i} \tag{18.5.2}$$

(remember that in each term in the above expression, x_i is either 0 or 1 and hence, for each i term in the product, only one of the two factors will contribute, contributing a factor θ_i^c if $x_i = 1$ and $1 - \theta_i^c$ if $x_i = 0$). Putting this all together, we can find the log likelihood

$$L(\boldsymbol{\theta}^c) = \sum_{i,\mu} x_i^\mu \log \theta_i^c + (1 - x_i^\mu) \log(1 - \theta_i^c) \tag{18.5.3}$$

Optimising with respect to $\theta_i^c \equiv p(x_i = 1|c)$ (differentiate with respect to θ_i^c and equate to zero) gives

$$p(x_i = 1|c) = \frac{\text{number of times } x_i = 1 \text{ for class c}}{(\text{number of times } x_i = 1 \text{ for class c}) + (\text{number of times } x_i = 0 \text{ for class c})} \tag{18.5.4}$$

A similar Maximum Likelihood argument gives the intuitive result:

$$p(c) = \frac{\text{number of times class c occurs}}{\text{total number of data points}} \tag{18.5.5}$$

18.5.1 Classification Boundary

If we just wish to find the most likely class for a new point x, we can compare the log probabilities, classifying x^* as class 1 if

$$\log p(c = 1|x^*) > \log p(c = 0|x^*) \tag{18.5.6}$$

Using the definition of the classifier, this is equivalent to (since the normalisation constant $-\log p(x^*)$ can be dropped from both sides)

$$\sum_i \log p(x_i^*|c = 1) + \log p(c = 1) > \sum_i \log p(x_i^*|c = 0) + \log p(c = 0)$$

Using the binary encoding $x_i \in \{0, 1\}$, we classify x^* as class 1 if

$$\sum_i \left\{x_i^* \log \theta_i^1 + (1 - x_i^*) \log(1 - \theta_i^1)\right\} + \log p(c = 1) > \sum_i \left\{x_i^* \log \theta_i^0 + (1 - x_i^*) \log(1 - \theta_i^0)\right\} + \log p(c = 0)$$

This decision rule can be expressed in the form: classify x^* as class 1 if $\sum_i w_i x_i^* + a > 0$ for some suitable choice of weights w_i and constant a (the reader is invited to find the explicit values of these weights). The interpretation of this is that w specifies a hyperplane in the x space and x^* is classified as a 1 if it lies on one side of the hyperplane. We shall talk about other such "linear" classifiers in a later chapter.

18.6 Naive Bayes : The multinomial case

Consider a (visible) variable x_i that can be in a discrete state $s \in \{1, \dots S\}$ (the generalisation to having a different number of states for a different i is straightforward). Consider the fitting a model to the data from class c. Under the naive Bayes assumption, a discrete valued vector x will have probability

$$p(x|c) = \prod_i p(x_i|c)$$

subject to the normalisation constraint

$$\sum_s p(x_i = s|c) = 1$$

For a set of data vectors x^μ,$\mu = 1, \dots P$, belonging to class c assuming iid, the likelihood of the data from class c is

$$\prod_{\mu=1}^{P} p(x^\mu|c^\mu) = \prod_{\mu=1}^{P} \prod_{i=1}^{N} \prod_{s=1}^{S} \prod_{c=1}^{C} p(x_i = s|c)^{I[x_i^\mu = s]I[c^\mu = c]}$$

or

$$L = \sum_{\mu=1}^{P} \sum_{i=1}^{N} \sum_{s=1}^{S} \sum_{c=1}^{C} I[x_i^\mu = s]I[c^\mu = c] \log p(x_i = s|c)$$

The parameters are $p(x_i = s|c)$. If we optimize this with respect to these parameters, using a lagrange multiplier to ensure normalisation (one for each of the outputs i):

$$L = \sum_{\mu=1}^{P} \sum_{i=1}^{N} \sum_{s=1}^{S} \sum_{c=1}^{C} I[x_i^\mu = s]I[c^\mu = c] \log p(x_i = s|c) + \sum_{c=1}^{C} \sum_{i=1}^{N} \lambda_i^c \sum_{s=1}^{S} p(x_i = s|c)$$

Differentiating this with respect to $p(x_i = s|c)$, we get

$$\sum_{\mu=1}^{P} \frac{I[x_i^\mu = s]I[c^\mu = c]}{p(x_i = s|c)} = \lambda_i^c$$

Hence, by normalisation,

$$p(x_i = s|c) = \frac{\sum_\mu I[x_i^\mu = s]I[c^\mu = c]}{\sum_{s'} \sum_{\mu'} I[x_i^{\mu'} = s']I[c^{\mu'} = c]}$$

In words, this means simply that the optimal ML setting for the parameter $p(x_i = s|c)$ is simply (for the class c data), the relative number of times that attribute i is in state s. (Analogous to the binary example before).

18.6.1 Dirichlet Prior

The previous ML derivation suffers if there are no cases in which a variable is in state s in the training data. In that case, the probabilities become certainties, and classification can be overconfident. A simple approach around this is to consider putting priors on the probabilities $p(x_i = s|c)$. A natural prior to use is a Dirichlet distribution (see appendix). First, though, let's see how we would use, in general, a prior distribution in the classification of a novel point x^*.

let D denote the training data $(x^\mu, c^\mu), \mu = 1, \ldots, P$. We will have, for each i, a distribution $p(x_i|c)$. Since each x_i can take one of S states (the derivation below is general enough to include the case that each i can have a different number of states), we need to specify a probability for each of these states. This is described by the S dimensional vector $\boldsymbol{\alpha}^i(c)$, so that $p(x_i = s|c) \equiv \alpha_s^i(c)$.

We would like to calculate

$$p(c|x^*, D) = \frac{p(x^*, c, D)}{p(x^*, D)} \propto \int_\alpha p(x^*, c, D, \alpha) \propto \prod_i \int_{\boldsymbol{\alpha}^i(c)} p(x_i^*|\boldsymbol{\alpha}^i(c))p(\boldsymbol{\alpha}(c)|D)$$

Let's look at the posterior distribution

$$p(\boldsymbol{\alpha}(c)|D) \propto p(c)p(D|\boldsymbol{\alpha}(c))p(\boldsymbol{\alpha}(c)) \tag{18.6.1}$$

$$= \prod_i \left(\left(\prod_\mu p(x_i^\mu|\boldsymbol{\alpha}^i(c^\mu)) \right) p(\boldsymbol{\alpha}^i(c)) \right) \tag{18.6.2}$$

It's clear therefore that the posterior factorises

$$p(\boldsymbol{\alpha}(c)|D) = \prod_i p(\boldsymbol{\alpha}^i(c)|D)$$

where

$$p(\boldsymbol{\alpha}^i(c)|D) \propto p(\boldsymbol{\alpha}^i(c)) \prod_{\mu:c^\mu=c} p(x_i^\mu|\boldsymbol{\alpha}^i(c))$$

When the prior is a Dirichlet distribution,

$$p(\boldsymbol{\alpha}^i(c)) = \text{Dirichlet}(\boldsymbol{\alpha}^i(c)|u^i(c))$$

the posterior is also a Dirichlet distribution (since the Dirichlet distribution is conjugate to the multinomial distribution),

$$p(\boldsymbol{\alpha}^i(c)|D) = \text{Dirichlet}(\boldsymbol{\alpha}^i(c)|\hat{u}^i(c))$$

where the vector $\hat{u}^i(c)$ has components

$$[\hat{u}^i(c)]_s = u^i_s(c) + \sum_{\mu:c^\mu=c} I[x^\mu_i = s]$$

Here the parameter $u^i(c)$ describes the form of the prior. If we take this to be the unit vector, the distribution on the (infinite set of) distributions $p(\boldsymbol{\alpha}^i(c))$ is flat. This is not an unreasonable assumption to make. This then becomes

$$p(c|x^*, D) \propto p(c) \prod_i \frac{Z(u^{*i}(c))}{Z(\hat{u}^i(c))}$$

where

$$u^{*i}_s(c) = \hat{u}^i_s(c) + I[x^*_i = s]$$

and $Z(u)$ is the normalisation constant of the distribution Dirichlet$(\boldsymbol{\alpha}|u)$ Repeating the previous analysis on the 'Are they Scottish?' data, the probability under a uniform Dirichlet prior for all the tables, gives a value of 0.236 for the probability that $(1, 0, 1, 1, 0)$ is Scottish, compared with a value of 0.192 under the standard Naive Bayes assumption.

An advantage of this Dirichlet prior framework is that it works also when there are zero counts in the data.

CHECK: is there a simpler way to write the above ratio using the relationships of the $Gamma(x + 1) = xGamma(x)$? Maybe this gives an equivalent form to smoothing the counts?

18.7 Problems

Exercise 59 *A local supermarket specializing in breakfast cereals decides to analyze the buying patterns of its customers.*

They make a small survey asking 6 randomly chosen people which of the breakfast cereals (Cornflakes, Frosties, Sugar Puffs, Branflakes) they like, and also asking for their age (older or younger than 60 years). Each respondent provides a vector with entries 1 or 0 corresponding to whether they like or dislike the cereal. Thus a respondent with (1101) *would like Cornflakes, Frosties and Branflakes, but not Sugar Puffs.*

The older than 60 years respondents provide the following data :

$$(1000), (1001), (1111), (0001)$$

For the younger than 60 years old respondents, the data is

$$(0110), (1110)$$

A novel customer comes into the supermarket and says she only likes Frosties and Sugar Puffs. Using Naive Bayes trained with maximum likelihood, what is the probability that she is younger than 60?

Exercise 60 *A psychologist does a small survey on 'happiness'.*

Each respondent provides a vector with entries 1 or 0 corresponding to whether answer 'yes' to a question or 'no', respectively. The question vector has attributes

$$x = (rich, married, healthy)$$

In addition, each respondent gives a value $c = 1$ if they are content with their lifestyle, and $c = 0$ if they are not.

Thus, a response $(1, 0, 1)$ would indicate that the respondent was 'rich', 'unmarried', 'healthy'.

The following responses were obtained from people who claimed also to be 'content' :

$$(1, 1, 1), (0, 0, 1), (1, 1, 0), (1, 0, 1)$$

For the 'not content' respondents, the data is

$$(0, 0, 0), (1, 0, 0), (0, 0, 1), (0, 1, 0), (0, 0, 0)$$

Using Naive Bayes on this data, what is the probability that a person who is 'not rich', 'married' and 'healthy' is 'content'?

What is the probability that a person who is 'not rich' and 'married' is 'content'? (That is, we do not know whether or not they are 'healthy').

Consider the following vector of attributes :

$x_1 = 1$ *if customer is younger than 20* ; $x_1 = 0$ *otherwise*

$x_2 = 1$ *if customer is between 20 and 30 years old* ; $x_2 = 0$ *otherwise*

$x_3 = 1$ *if customer is older than 30* ; $x_3 = 0$ *otherwise*

$x_4 = 1$ *if customer walks to work* ; $x_4 = 0$ *otherwise*

Each vector of attributes has an associated class label "rich" or "poor".

Point out any potential difficulties with using your previously described approach to training using Naive Bayes. Hence describe how to extend your previous Naive Bayes method to deal with this dataset. Describe in detail how maximum likelihood could be used to train this model.

Exercise 61 *Whizzco decide to make a text classifier. To begin with they attempt to classify documents as either sport or politics. They decide to represent each document as a (row) vector of attributes describing the presence or absence of words.*

$$x = (\text{goal}, \text{football}, \text{golf}, \text{defence}, \text{offence}, \text{wicket}, \text{office}, \text{strategy}) \tag{18.7.1}$$

Training data from sport documents and from politics documents is represented below in MATLAB using a matrix in which each row represents a (row) vector of the 8 attributes.

```
xP=[1 0 1 1 1 0 1 1;  % Politics
    0 0 0 1 0 0 1 1;
    1 0 0 1 1 0 1 0;
    0 1 0 0 1 1 0 1;
    0 0 0 1 1 0 1 1;
    0 0 0 1 1 0 0 1]

xS=[1 1 0 0 0 0 0 0; % Sport
    0 0 1 0 0 0 0 0;
    1 1 0 1 0 0 0 0;
    1 1 0 1 0 0 0 1;
    1 1 0 1 1 0 0 0;
    0 0 0 1 0 1 0 0;
    1 1 1 1 1 0 1 0]
```

Using a Naive Bayes classifier, what is the probability that the document $x = (1, 0, 0, 1, 1, 1, 1, 0)$ is about politics?

Exercise 62 *A Naive Bayes Classifier for binary attributes $x_i \in \{0, 1\}$ is parameterised by $\theta_i^1 = p(x_i = 1|class = 1)$, $\theta_i^0 = p(x_i = 1|class = 0)$, and $p_1 = p(class = 1)$ and $p_0 = p(class = 0)$. Show that the decision boundary to classify a datapoint x can be written as $\boldsymbol{w^T x + b > 0}$, and state explicitly w and b as a function of $\boldsymbol{\theta}^1, \boldsymbol{\theta}^0, p_1, p_0$.*

Exercise 63 *The logistic sigmoid function is defined as $\sigma(x) = e^x/(1+e^x)$. What is the inverse function, $\sigma^{-1}(x)$?*

Exercise 64 *Given a dataset $\{(x^\mu, c^\mu), \mu = 1, \ldots, P\}$, where $c^\mu \in \{0, 1\}$, logistic regression uses the model $\boldsymbol{p(c = 1|x) = \sigma(w^T x + b)}$. Assuming that the data is drawn independently and identically, show that the derivative of the log likelihood L of the data is*

$$\nabla_w L = \sum_{\mu=1}^{P} \left(c^\mu - \sigma\left(w^T x^\mu + b\right)\right) x^\mu$$

Exercise 65 *Consider a dataset $\{(x^\mu, c^\mu), \mu = 1, \ldots, P\}$, where $c^\mu \in \{0, 1\}$, and x is a N dimensional vector.*

- *Show that if the training data is linearly separable with the hyperplane $\boldsymbol{w^T x + b}$, the data is also separable with the hyperplane $\tilde{w}^T x + \tilde{b}$, where $\tilde{w} = \lambda w$, $\tilde{b} = \lambda b$ for any scalar $\lambda > 0$.*
- *What consequence does the above result have for maximum likelihood training of linearly separable data?*

Exercise 66 *Consider a dataset $\{(x^\mu, c^\mu), \mu = 1, \ldots, P\}$, where $c^\mu \in \{0, 1\}$, and x is a P dimensional vector. (Hence we have P datapoints in a P dimensional space). If we are to find a hyperplane (parameterised by (w, b)) that linearly separates this data we need, for each datapoint x^μ,*

$$\boldsymbol{w^T x^\mu + b = \epsilon^\mu}$$

where $\epsilon^\mu > 0$ for $c^\mu = 1$ and $\epsilon^\mu < 0$ for $c^\mu = 0$.

- *Show that, provided that the data $\{x^\mu, \mu = 1, \ldots, P\}$ are linearly independent, a solution (w, b) always exists for any chosen values ϵ^μ.*
- *Discuss what bearing this has on the fact that the 600 handwritten digit training points are linearly separable in a 784 dimensional space.*
- *(Difficult) Comment on the relation between maximum likelihood training and the algorithm suggested above.*

Exercise 67 *Given training data $D = \{(x^\mu, y^\mu), \mu = 1, \ldots, P\}$, you decide to fit a regression model $y = mx + c$ to this data. Derive an expression for m and c in terms of D using the minimum sum squared error criterion.*

Exercise 68 *The regularised training error for a linear parameter regression model* $\boldsymbol{y = w^T \phi}$ *is*

$$E_{regtrain}(w, \lambda) = \sum_{\mu=1}^{P} (y^\mu - w^T \boldsymbol{\phi}^\mu)^2 + \lambda w^T w$$

Derive an expression for w that minimises $E_{regtrain}$ for fixed λ.

Exercise 69 *The MATLAB code below implements a Radial Basis Function LPM on some artificial training data:*

```
x = 0:0.05:1; y = sin(20*x); % training data
alpha = 0.1;  % basis function width
m = -1.5:0.1:1.5; % basis function centres

phi = phi_rbfn(x,alpha,m);
w=(phi*phi'+10^(-7)*eye(size(phi,1)))\sum(repmat(y,size(phi,1),1).*phi,2);

ytrain = w'*phi; xpred = -0.5:0.01:1.5;
ypred=w'*phi_rbfn(xpred,alpha,m); plot(x,y,'rx'); hold on;
plot(xpred,sin(20*xpred)); plot(xpred,ypred,'--');
```

Write a routine `function phi=phi_rbfn(x,alpha,m)` *to complete the above code where $\phi_i(x) = \exp(-0.5(x - m_i)^2/\alpha^2)$, and run the completed code. Comment on your predictions.*

Exercise 70 *Given training data $D = \{(x^\mu, c^\mu), \mu = 1, \ldots, P\}$, $c^\mu \in \{0, 1\}$, you decide to make a classifier*

$$p(c = 1|x) = \sigma\left(w^T \boldsymbol{\phi}(x)\right)$$

where $\sigma(x) = e^x/(1 + e^x)$ and $\boldsymbol{\phi}(x)$ is a chosen vector of basis functions.

- *Calculate the log likelihood $L(w)$ of the training data.*
- *Calculate the derivative $\nabla_w L$ and suggest a training algorithm to find the maximum likelihood solution for w.*
- *Comment on the relationship between this model and logistic regression.*
- *Comment on the decision boundary of this model.*

- *If we have P training points and $dim(x) = P$, can we always separate the training data perfectly using this model?*

Exercise 71 *Given training data $D = \{(x^\mu, c^\mu), \mu = 1, \ldots, P\}$, $c^\mu \in \{0, 1\}$, and x are two dimensional, you decide to make a classifier using a neural network with a single hidden layer, with two hidden units, each with transfer function $g(x) = \exp(-0.5x^2)$. The output transfer function is the logistic sigmoid, $\sigma(x) = e^x/(1 + e^x)$, so that the network is,*

$$p(c = 1|x) = \sigma(b_0 + v_1 g(w_1^T x + b_1) + v_2 g(w_2^T x + b_2))$$

- *Write down the log likelihood, based on the usual i.i.d assumption for this model.*
- *Calculate the derivatives of the log likelihood as a function of the network parameters, $w_1, w_2, b_1, b_2, v, b0$*
- *Comment on the relationship between this model and logistic regression.*
- *Comment on the decision boundary of this model.*

Exercise 72 *An example classification application is to classify Reuters news stories into categories (e.g. sport, politics, finance etc). This is an example of a multi-class problem. One approach to solving this is to use the so-called 'softmax' output which, for C classes takes the general form:*

$$p(class = i|x) = \frac{e^{f(x,i)}}{\sum_{j=1}^{C} e^{f(x,j)}}$$

Comment on the relation of this general approach to logistic regression.

To use the above 'softmax' approach we need to convert text into a vector representation. This is usually done using the "bag of words" representation of the document where the order of the words is thrown away, so that simple counts of each word present in the document are used. There are some standard preprocessing techniques that are also used:

- *Word stems are used rather than words;*
- *Words that occur infrequently in the training data are discarded;*
- *"Stop words" (like "and", "or" etc) are not counted;*
- *To deal with documents of different length, each document feature vector is normalized to have unit length.*

Consider how long the feature vector may be and discuss why these preprocessing techniques may be beneficial. Are there situations where you think the "bag of words" representation would cause problems?

Outline how you would proceed to formulate and train an appropriate model.

18.8 Solutions

59 Looking at the data, the estimates using maximum likelihood are

$$p(C = 1|Young) = 0.5, p(F = 1|Young) = 1, p(SP = 1|Young) = 1, p(B = 1|Young) = 0$$

and

$$p(C = 1|Old) = 0.75, p(F = 1|Old) = 0.25, p(Sp = 1|Old) = 0.25, p(B = 1|Old) = 0.75$$

and $p(Young) = 2/6$ and $p(Old) = 4/6$. Plugging this into Bayes formula, we get

$$p(Young|C = 0, F = 1, SP = 1, B = 0) \propto 0.5 * 1 * 1 * 1/6$$

$$p(Old|C = 0, F = 1, SP = 1, B = 0) \propto 0.25 * 0.25 * 0.25 * 0.25 * 4/6$$

Using the fact that these probabilities sum to 1, this gives $p(Young|C = 0, F = 1, SP = 1, B = 0) = 64/65$

19 Mixture Models : Discrete Hidden Variables

19.1 Mixture Models

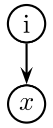


Figure 19.1: A mixture model has a trivial graphical representation as a DAG with a single hidden node, which can be in one of H states, $i = 1 \dots H$.

A common assumption is that data lies in 'clusters'. For example, if we have examples of handwritten digits, then the training vectors which represent the digit '1' will typically be much closer to each other (in the Euclidean sense) than those vectors which represent the digit '8'. This suggests that, in this case, it is natural to expect the presence of two main clusters in the total training data – that is, when all the data, regardless of whether it is classed as a '1' or an '8' is examined together.

In a probabilistic sense, a cluster equates to a mixture component, so that the data is represented by a summation (mixture) of models, each model being responsible for an individual 'cluster' :

$$p(x|\Theta) = \sum_{i=1}^{H} p\left(x|\theta_i, i\right) p\left(i\right) \tag{19.1.1}$$

Where i represents mixture component i and $\Theta = \{\theta_1 \dots \theta_h\}$ are the parameters that determine the mixture distributions. The general form of a mixture model is given in fig(19.1).

Whilst there are some cases, such as the Gaussian Mixture Model discussed below, where there is a clear visual interpretation of the meaning of 'cluster', the reader should bear in mind that the intuitive meaning of 'cluster' is based on datapoints being 'close' in some sense to each other. The tremendous advantage of the realisation that mixture models generalise the idea of modelling clusters, is that we are freed from the conceptual constraint of distances in some Euclidean type sense. Instead, two datapoints become 'close' if they are both likely with respect to the model for that 'cluster'. Hence, we can immediately start to 'cluster' all kinds of data – music, shopping purchases etc – things which do not necessarily have a natural 'distance' measure.

Training Mixture Models

Although it is perfectly feasible to find appropriate MAP parameters by standard optimization techniques, the variational learning approach is usually far superior.

According to the general theory, we need to consider the energy term:

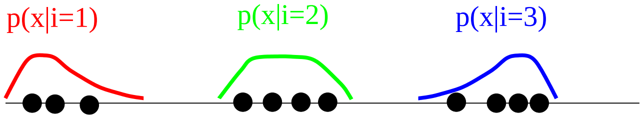


Figure 19.2: It is clear that the black dots, which represent the one dimensional data values are naturally clustered into three groups. Hence, a reasonable model of this data would be $p(x) = p(x|1)p(1)+p(x|2)p(2)+p(x|3)p(3) = \sum_{i=1}^{3} p(x|i)p(i)$ where $p(x|i)$ is the model for the data in cluster i, and $\sum_i p(i) = 1$.

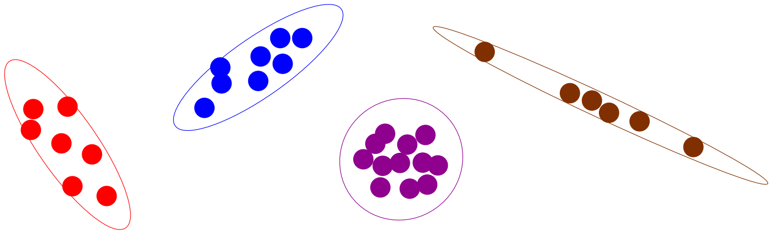


Figure 19.3: Gaussian Mixture Models place blobs of probability mass in the space. Here we have 4 mixture components in a 2 dimensional space. Each mixture has a different covariance matrix and mean.

$$\sum_{i,\mu} q^{\mu}(i) \log \left(p\left(x^{\mu}|\theta_i, i\right) p\left(i\right)\right) \tag{19.1.2}$$

and maximise this with respect to the parameters $\theta_i, i = 1, \ldots, H$, and $p(i), i = 1, \ldots, H$.

Considering the dependence of the above function on $p(i)$, and including a normalisation Lagrangian, we obtain

$$\sum_{i,\mu} q^{\mu}(i) \log p\left(i\right) + \lambda \left(1 - \sum_i p(i)\right)$$

Differentiating, and ensuring normalisation gives

$$p^{new}(i) \propto \sum_{\mu} q^{\mu}(i) \tag{19.1.3}$$

The parameters $\theta_i, i = 1, \ldots, H$ are determined by

$$\underset{\theta_i, i=1,\ldots,H}{\operatorname{argmax}} \sum_{i,\mu} q^{\mu}(i) \log p\left(x^{\mu}|\theta_i, i\right) \tag{19.1.4}$$

Provided that the parameters are not shared by the mixture components, we have simply, for each θ_i

$$\theta_i^{new} = \underset{\theta_i}{\operatorname{argmax}} \sum_{\mu} q^{\mu}(i) \log p\left(x^{\mu}|\theta_i, i\right)$$

The choice for the variational distributions is user dependent. The optimal EM setting is

$$q^{\mu}(i) = p(i|x^{\mu}, \theta_i) \propto p\left(x^{\mu}|\theta_i, i\right) p\left(i\right) \tag{19.1.5}$$

Equations (19.1.3,19.1.4,19.1.5) are repeated until convergence. The initialisation of the parameters θ_i and mixture probabilities can severely affect the quality of the solution found. If random initialisations are used, it is recommended to record the value of the likelihood itself, to see which converged parameters have the higher likelihood. In the case that the likelihood is difficult to compute exactly, and the variational method is used to form a bound,

$$B(\mathbf{\Theta}) = \sum_{\mu} \left\{ -\sum_i q^{\mu}(i) \log q^{\mu}(i) + \sum_i q^{\mu}(i) \log \left(p(x^{\mu}|\theta_i, i) p(i) \right) \right\}$$

then the bound serves as a way to assess which converged parameters are to be preferred.

19.2 Gaussian Mixture Models

One commonly used form for the mixture components is to use Gaussian (normal) distributions.

An n dimensional Gaussian distribution is specified as

$$p\left(x|m, S\right) = \frac{1}{\sqrt{\det 2\pi S}} \exp\left\{ -\frac{1}{2} \left(x - m\right)^T S^{-1} \left(x - m\right) \right\} \qquad (19.2.1)$$

where m is the mean and S is the covariance matrix. If we have a separate Gaussian for each component, with no shared parameters, then $\theta_i = \{m_i, S_i\}$. It is customary to choose a number of mixture components that is smaller than the number of datapoints.

Infinite Troubles

A difficulty arises here with the application of ML to training Gaussian mixture models. By taking the limit of an infinitely narrow covariance matrix, the contribution to the likelihood from that datapoint becomes infinite. This says that, according to the pure ML criterion, optimally, we should place infinitely narrow Gaussians on some datapoints, and not worry about the rest. This is clearly an undesirable solution to the problem, and arises because in this case, the ML solution does not constrain the parameters in a sensible way. This is a classic example of when ML can lead to poor results. Exercise for the reader : why does this problem not occur when we only fit one Gaussian (to more than one datapoint).

All computational methods which aim to fit mixtures of Gaussians using ML therefore either succeed by getting trapped in serendipitous local maxima, or by the ad hoc addition of "extra constraints" on the width of the Gaussians. A more reasonable approach is to incorporate such necessary assumptions on the widths of the Gaussians in the prior beliefs. This has the advantage of transparency and clarity in the line of thought. In this sense, therefore, we can use the MAP approach in preference to the ML solution. In practice, however, it is more commonplace to use a simple criterion which prevents the eigenvalues of the covariance matrices from becoming too small. A Bayesian solution to this problem is possible and requires a prior on covariance matrices. The natural prior in this case is the so-called Wishart Distribution, which we shall discuss later.

Finding the optimal m_i

In the case of using Gaussian mixture components,

$$\log p\left(x^\mu|\theta_i, i\right) = -\frac{1}{2}\left(x^\mu - m_i\right)^T S_i^{-1}\left(x^\mu - m_i\right) - \frac{1}{2}\log\det 2\pi S_i \qquad (19.2.2)$$

We can then readily optimize the associated energy term

$$\sum_\mu q^\mu(i)\left\{-\frac{1}{2}\left(x^\mu - m_i\right)^T S_i^{-1}\left(x^\mu - m_i\right) - \frac{1}{2}\log\det 2\pi S_i\right\} \qquad (19.2.3)$$

with respect to m_i to obtain

$$m_i = \frac{\sum_\mu q^\mu(i)x^\mu}{\sum_\mu q^\mu(i)} \qquad (19.2.4)$$

Finding the optimal S_i

Optimising equation (19.2.3) with respect to S_i is slightly more difficult.

Using $\text{trace}\left(\log A\right) = \log\det A$, and $\partial\log A = A^{-1}\partial A$, where ∂ represents a differentiation operator, one can show that $\partial\log\det A = \text{trace}\left(A^{-1}\partial A\right)$. Also, since $A^{-1}A = I$, this gives $\partial A^{-1} = A^{-1}\partial A A^{-1}$. Using these results we get, for the derivative wrt S_i,

$$\sum_\mu q^\mu(i)\left\{\frac{1}{2}(\Delta x^\mu)^T S_i^{-1}\partial S_i S_i^{-1}\Delta x^\mu - \frac{1}{2}\text{trace}\left(S_i^{-1}\partial S_i\right)\right\} \qquad (19.2.5)$$

where $\Delta x^\mu \equiv x^\mu - m_i$. Using $a^T A a \equiv \text{trace}\left(A a a^T\right)$, we obtain

$$\text{trace}\left(S_i^{-1}\partial S_i S_i^{-1}\sum_\mu q^\mu(i)\left(\Delta x^\mu\left(\Delta x^\mu\right)^T - S_i\right)\right) \qquad (19.2.6)$$

This derivative is clearly zero if

$$S_i = \frac{\sum_\mu q^\mu(i)\left(x^\mu - m_i\right)\left(x^\mu - m_i\right)^T}{\sum_\mu q^\mu(i)} \qquad (19.2.7)$$

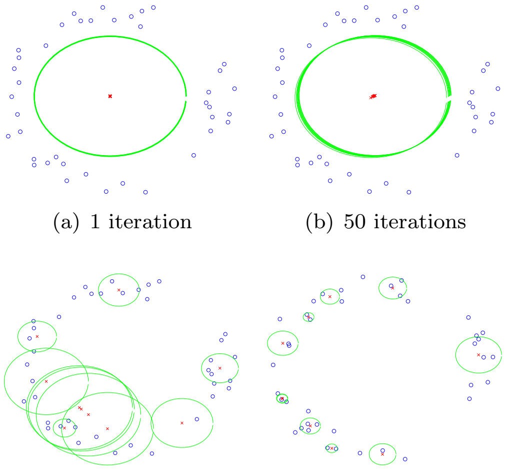
To get the mixing coefficients, use a Lagrange multiplier to give

$$p(i) = \frac{\sum_\mu q^\mu(i)}{\sum_{i,\mu} q^\mu(i)} = \frac{1}{P}\sum_\mu q^\mu(i) \qquad (19.2.8)$$

where P is the number of training examples. The EM choice for the variational distributions is

$$q^\mu(i) \propto p(i)e^{-\frac{1}{2}(x^\mu - m_i)^T S_i^{-1}(x^\mu - m_i)} \qquad (19.2.9)$$

The above equations (19.2.4,19.2.7,19.2.8,19.2.9) are iterated until convergence. A useful intialisation strategy is to set the covariances to be diagonal, with large variance – this gives the components a chance to 'sense' where data lies. An illustration of the performance of the algorithm is given in fig(19.4).



(a) 1 iteration

(b) 50 iterations

(c) 125 iterations

(d) 150 iterations

Figure 19.4: Training a mixture of 10 Gaussians (a) If we start with large variances for the Gaussians, even after one iteration, the Gaussians are centred close to the mean of the data. (b) The Gaussians begin to separate (c) One by one, the Gaussians move towards appropriate parts of the data (d) The final converged solution. Here the Gaussians were constrained so that the variances could not go below 0.01.

Symmetry Breaking

An interesting observation about the performance of EM applied to mixture models is that, initially, there appears as if little is happening, as each model jostles with the others to try to explain the data. Eventually, some almost seemingly random effect causes one model to break away from the jostling and explain data close to that model. The origin of this jostling is an inherent symmetry in the solution to this problem: it makes no difference to the likelihood if we relabel what the components are called. This permutation symmetry causes the initial confusion amongst the models as to who is going to explain which parts of the data. Eventually, this symmetry is broken, and a local solution is found. This can severely handicap the performance of EM when there are a large number of models in the mixture. An heuristic is to therefore begin with a small number of models, say two, for which symmetry breaking is less problematic. Once a local broken solution has been found, then more models are included into the mixture, initialised close to the currently found broken solutions. In this way, a hierarchical breaking scheme is envisaged. Clearly, there is potentially some bias introduced in this scheme as to the kinds of solutions found – however, this may be a small price to pay in the light of waiting for a very long time as the models jostle unnecessarily. Another popular method for initialisation is to center the means to those found by the K-means algorithm – however, this itself requires a heuristic initialisation.

Parzen Estimator

The Parzen estimator is one of the simplest density estimators. The idea is simply to put a Gaussian at each datapoint x^k, $k = 1 \dots P$. Usually, this Gaussian is chosen to be isotropic – that is, with covariance matrix $\mathbf{\Sigma} = \sigma \mathbf{I}$, where σ is some pre-chosen value. In general, therefore, the density estimator is

$$p(x) = \frac{1}{P} \sum_{\mu=1}^{P} \frac{1}{\sqrt{\det 2\pi \mathbf{\Sigma}}} e^{-\frac{1}{2}(x-x^\mu)\mathbf{\Sigma}^{-1}(x-x^\mu)} \tag{19.2.10}$$

Whilst an intuitively reasonable thing to do, if one is working with large datasets in high dimensional spaces, one needs to store all the datapoints in order to calculate the density, which can be prohibitive.

Unless the widths of the Gaussians are chosen to be broad, only a small region of the space is covered by each Gaussian bump. In high dimensional spaces therefore, the Parzen estimator will only have appreciable density very close to the data or, if the Gaussians are broad, the density will be underestimated close to the data.

```
% demo for fitting mixture of isotropic Gaussians

%make an annulus of data :
l = 0.2; r1 = 0.5; for r = 1:50
  rad = r1 + rand*l;  theta = rand*2*pi;  X(1,r) = rad*cos(theta); X(2,r) = rad*sin(theta);
end

h = 5;         % number of mixtures
d = size(X,1); % dimension of the space
n = size(X,2); % number of training patterns
Smin = 0.001;  % minimum variance of Gaussians

r = randperm(n); M = X(:,r(1:h)); % initialise the centres to random datapoints
S = 100*ones(1,h); % initialise the variances to be large
P = ones(1,h)./h;  % intialise the component probilities to be uniform

for its = 1:150 % number of iterations

  for i = 1:h
    for k = 1:n
      v = X(:,k) - M(:,i);
      Q(k,i) = exp(-0.5*(v'*v)/S(i)).*P(i)./sqrt((S(i))^d);
    end
  end
  su = sum(Q,2);
  for k =1:n
    Q(k,:) = Q(k,:)./su(k); % responsibilities p(i|x^n)
  end

  for i = 1:h % now get the new parameters for each component
    N(i) = sum(Q(:,i));
    Mnew(:,i) = X*Q(:,i)./N(i);
    Snew(i) = (1/d)*sum( (X - repmat(Mnew(:,i),1,n)).^2 )*Q(:,i)./N(i);
    if Snew(i) < Smin % don't decrease the variance below Smin
      Snew(i) = Smin;
    end
  end

  Pnew = N; Pnew = Pnew./sum(Pnew);
  S = Snew; M = Mnew; P = Pnew; % update the parameters
end
```

19.3 K Means

A non-probabilistic limit of fitting Gaussian mixtures to data is given by the K means algorithm, in which we simply represent an original set of P datapoints by K points.

19.3.1 The algorithm

1. Initialise the centres $\boldsymbol{\mu}_i$ to K randomly chosen datapoints.
2. For each cluster mean, j, find all the x for which cluster j is the nearest cluster. Call this set of points S_j. Let N_j be the number of datapoints in set S_j.
3. Assign

$$\boldsymbol{\mu}_j = \frac{1}{N_j} \sum_{x \in S_j} x \tag{19.3.1}$$

We then iterate steps 2 and 3 above until some convergence criterion.

The code below implements this algorithm.

```
% demo for K Means
x = [randn(2,50) 5+randn(2,50) (repmat([-4 4]',1,50)+randn(2,50))]; % 150 2-dim datapoints

K = 3; % number of clusters
r = randperm(size(x,2));
m(:,1:K) = x(:,r(1:K)); % initialise the clusters to K randomly chosen datapoints
mold =m;

for its = 1: 100 % maximum number of iterations

  for p = 1:size(x,2)   % calculate the distances (this could be vectorised)
    for k = 1:K
      v = x(:,p) - m(:,k); d(k,p) = v'*v;
    end
  end

  [a,b]=min(d);   % find the nearest centres

  for k = 1:K
    if length(find(b==k))>0
      m(:,k) = mean(x(:,find(b==k))')';
    end
  end

  if mean(sum( (m-mold).^2)) < 0.001; break; end; mold =m; % termination criterion
end

cla; plot(x(1,:),x(2,:),'.'); hold on;
plot(m(1,:),m(2,:),'rx','markersize',15);
```

An example is given in fig(19.5) in which we represented 150 datapoints using 3 clusters.

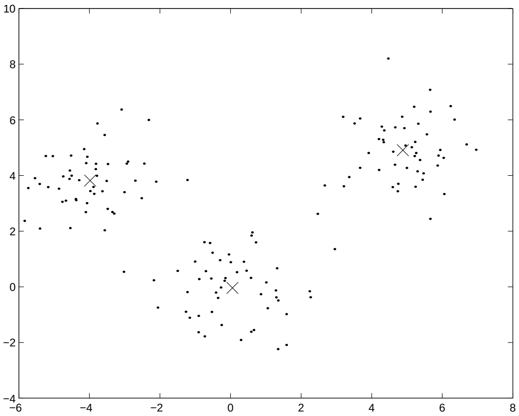


Figure 19.5: Result of fitting $K = 3$ means to 150 two dimensional datapoints. The means are plotted as crosses.

Note that the Kmeans algorithm can be dervived as the limit $\sigma \to 0$ for fitting isotropic Gaussian mixture components.

19.3.2 Uses of K Means

The K means algorithm, despite its simplicity is very useful. Firstly, it converges extremely quickly and often gives a reasonable clustering of the data, provided that the centres are initialised reasonably (using the above procedure for example). We can use the centres we found as positions in which to place basis function centres in the linear parametric models chapter.

19.4 Classification using Mixture Models

One popular use of mixture models is in classification. Consider the case in which we have two classes, 1 and 2. We can fit a Gaussian Mixture model to each class. That is, we could fit a mixture model to the data from class 1 :

$$p(x|c=1) = \sum_{k=1}^{K} p(x|k, c=1)p(k|c=1) \tag{19.4.1}$$

and a mixture model to the data from class 2. (One could use a different number of mixture components for the different classes, although in practice, one might need to avoid overfitting one class more than the other. Using the same number of mixture components for both classes avoids this problem.)

$$p(x|c=2) = \sum_{k=1}^{K} p(x|k, c=2)p(k|c=2) \tag{19.4.2}$$

So that each class has its own set of mixture model parameters. We can then form a classifier by using Bayes rule :

$$p(c=i|x) = \frac{p(x|c=i)p(c=i)}{p(x)} \tag{19.4.3}$$

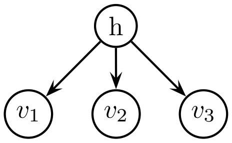


Figure 19.6: A mixture model has a trivial graphical representation as a DAG with a single hidden node, which can be in and one of H states, $i = 1 \dots H$.

Only the numerator is important in determining the classification since the denominator is the same for the case of $p(c = 2|x)$. This is a more powerful approach than our original approach in which we fitted a single Gaussian to each digit class. Using more Gaussians enables us to get a better model for how the data in each class is distributed and this will usually result in a better classifier.

19.5 Mixture of Multi-nomials

A company sends out a questionnaire, which contains a set of $v_1, \dots, v_n$ 'yes/no' questions. P customers send back their questionnaires, $v^1, \dots, v^P$, and the company wishes to perform an analysis to find what kinds of customers it has. Let us assume that the company has good reason to suspect that there are H essential type of customer and, given that we know which type we are dealing with, the profile of their responses will be quite well defined in the sense that $p(v_1, \dots, v_n|h)$ may be assumed to be factorised, $\prod_{i=1}^n p(v_i|h)$. (This is the same assumption as the Naive Bayes classifier – here the difference ultimately is that we will not have any training labels for the class of the data).

A model that captures the above situation would be a mixture of Binomials

$$p(v_1, \dots, v_n) = \sum_{h=1}^{H} p(h) \prod_{i=1}^{n} p(v_i|h)$$

where each term $p(v_i|h)$ is a Binomial distribution – that is, there are two states $v_i \in \{0, 1\}$. The generalisation to many states is straightforward.

EM training

In order to train the above model, we can use the EM algorithm, since we have a hidden variable. Formally, we can figure out the algorithm by, as usual, writing down the energy:

$$\sum_{\mu} \langle \log p(v_1^\mu, v_2^\mu, v_3^\mu, h) \rangle_{q^\mu(h)} = \sum_{\mu} \sum_{i} \langle \log p(v_i^\mu|h) \rangle_{q^\mu(h)} + \sum_{\mu} \langle \log p(h) \rangle_{q^\mu(h)}$$

and then performing the maximisation over the table entries. However, from our general intuition, we may immediately jump to the results:

$$p(v_i = 1|h = j) \propto \sum_{\mu} I[v_i^\mu = 1] q^\mu(h = j)$$

$$p(h = j) \propto \sum_{\mu} q^\mu(h = j)$$

$$q^\mu(h = j) \propto p(v^\mu|h = j)p(h = j) \propto p(h = j) \prod_{i} p(v_i^\mu|h = j)$$

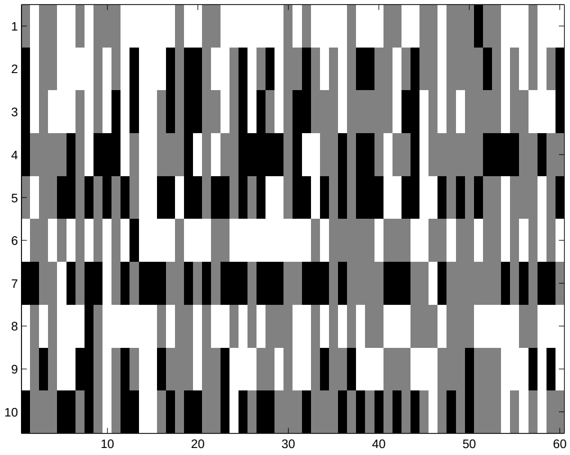


Figure 19.7: Data from questionnaire responses. There are 10 questions, and 60 people responded. White denotes a 'yes' and black denotes 'no'. Gray denotes that the absence of a response (missing data). This training data was generated by three component Binomial mixture. Missing data was then simulated by randomly removing values from the dataset.

These equations are iterated until convergence. Code that implements the above method is provided later in this chapter.

One of the pleasing aspects of this model is that if one of the attribute values is missing in the data set, the only modification to the algorithm is to drop the corresponding factor $p(v_i^\mu|h)$ from the algorithm. The verification that this is is a valid thing to do is left to the reader.

Example : Questionnaire

Data from a questionnaire is presented below in fig(19.7). The data has a great number of missing values. We have reason to believe that there are three kinds of respondents.

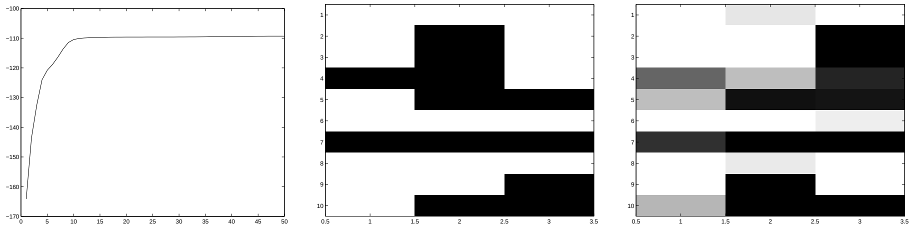
Running the EM algorithm on this data, with random initial values for the tables, gives an evolution for the lower bound on the likelihood as presented in fig(19.8a).

The EM algorithm finds a good solution. The 3 hidden states probabilities learned are $p(h = 1) \approx 1/3$,$p(h = 2) \approx 1/3$,$p(h = 3) \approx 1/3$, which is in rough agreement with the data generating process. The solution is permuted, but otherwise fine, and the three basic kinds of respondents have been well identified. Note how difficult this problem is to solve by visual inspection of the data fig(19.7).

Code that implements this problem is given at the end of this chapter.

Mixtures of HMM

A useful way to cluster temporal sequences is to use a mixture of HMMs. This can be trained in the usual EM way, and good for clustering temporal sequences : BioInformatics, Music etc.



(a) (b) “true” $p(v_i = 1|h)$ values. (c) learned $p(v_i = 1|h)$ values

Figure 19.8: (a) The evolution of the lower bound on the likelihood as we go through eventually 50 iterations of EM. The different regimes initially in the evolution are signatures of the symmetry breaking. (b) The ‘true’ value for the parameters $p(v_i = 1|h)$. Black corresponds to the value 0.95 and white to the value 0.1. (c) The solution $p(v_i = 1|h)$ found by the converged EM approach. (b) and (c) are closely related, except for a trivial permutation of the hidden label.

The key intuition in this chapter is that clustering corresponds to using a mixture model. These may be trained with EM, although some care is required with initialisation and symmetry breaking.

ANSWER: It’s true that the likelihood will be higher for the more complex model. What we need to do is to introduce a prior over the parameters of the model, and then integrate (as opposed to finding the set of parameters that maximises the likelihood). This will provide the effective Occam’s factor that will penalise the overly complex model. Need to do this. Note that this phenomenon only kicks in when we *integrate* over the unknown model parameters. Essentially, it’s similar to the fact that a more complex model will always have a lower training error – however, what we need to measure is something more like the effective volume of parameter space that has a low training error – this is given by the Bayesian solution.

```
function [pv,ph]=fitmixbern(v,nh,num_em_loops)

% MIXTURE OF BERNOULLI's trained using EM
% missing data coded with -1

n=size(v,1); P=size(v,2);
pv = rand(n,nh); % random initialisation for the probs
ph = rand(1,nh); ph = ph./sum(ph);

for em = 1:num_em_loops

  for mu = 1:P
    for i = 1:nh
      p(i,mu)=bern_prob(v(:,mu),pv(:,i))*ph(i); % p(i|vmu)*const.
    end
    p = p ./repmat(sum(p,1),nh,1); % p(i|vmu)
  end


  % update hidden probs
  sp = sum(p,2);
  for i = 1:nh
    phnew(i) = sp(i);
  end
  phnew = phnew./sum(phnew); % hidden probabilities

  % now update the tables p(v|i):
  pv1 = zeros(n,nh); pv0 = zeros(n,nh);

  for i = 1:nh
    for datadim = 1:n
      for mu = 1:P
      pv1(datadim,i) = pv1(datadim,i) + (v(datadim,mu)==1).*p(i,mu);
      pv0(datadim,i) = pv0(datadim,i) + (v(datadim,mu)==0).*p(i,mu);
      end
    end
  end

  for i = 1:nh
    for datadim = 1:n
      pvnew(datadim,i) = pv1(datadim,i)./(pv0(datadim,i)+pv1(datadim,i));
    end
  end

entropy = -sum(sum(p.*log(0.0000001+p))); energy=0;

for mu=1:P
  for datadim=1:n
    energy = energy + (v(datadim,mu)==1)*sum(log(0.0000001+pvnew(datadim,:)).*(p(:,mu)'));
    energy = energy + (v(datadim,mu)==0)*sum(log(0.0000001 + 1-pvnew(datadim,:)).*(p(:,mu)'));
  end
end

energy=energy+sum((log(0.0000001+phnew))*p);

bound(em) = entropy+energy; plot(bound); drawnow pv=pvnew;
ph=phnew;

end


function p = bern_prob(c,p)

  p = prod(p(find(c==1)))*prod((1-p(find(c==0))));
```

19.6 Problems

Exercise 73 *If a and b are $d \times 1$ column vectors and M is a $d \times d$ symmetric matrix, show that $a^T M b = b^T M a$.*

Exercise 74 *Write the quadratic forms $x_1^2 - 4x_1x_2 + 7x_2^2$ and $(x_1+x_2)^2 + (x_3+x_4)^2$ in the form $x^T C x$ where C is a symmetric matrix.*

Exercise 75 *Consider data points generated from two different classes. Class 1 has the distribution $P(x|\mathcal{C}_1) \sim N(\mu_1, \sigma^2)$ and class 2 has the distribution $P(x|\mathcal{C}_2) \sim N(\mu_2, \sigma^2)$. The prior probabilities of each class are $P(\mathcal{C}_1) = P(\mathcal{C}_2) = 1/2$. Show that the posterior probability $P(\mathcal{C}_1|x)$ is of the form*

$$P(\mathcal{C}_1|x) = \frac{1}{1 + \exp -(ax + b)}$$

and determine a and b in terms of μ_1, μ_2 and σ^2. The function $f(z) = 1/(1 + e^{-z})$ is known as the logistic function, and is a commonly-used transfer function in artificial neural networks.

Exercise 76 *The Poisson distribution is a discrete distribution on the non-negative integers, with*

$$P(x) = \frac{e^{-\lambda}\lambda^x}{x!} \qquad x = 0,\ 1,\ 2, \ldots$$

You are given a sample of n observations $x_1, \ldots, x_n$ drawn from this distribution. Determine the maximum likelihood estimator of the Poisson parameter λ.

Exercise 77 *Generate a sample of 100 points from a bivariate Gaussian distribution. You should use the matlab functions randn and chol (the Cholesky decomposition) to help you. Hint: the Cholesky decomposition U of a matrix Σ is such that $U^T U = \Sigma$. If x is a random vector drawn from $N(0, I)$, then $\boldsymbol{y = U^T x}$ is a random vector that has mean 0 and covariance $\boldsymbol{E[yy^T] = E[U^T x x^T U] = U^T U = \Sigma}$.*

19.7 Solutions

20 Factor Analysis and PPCA

Introduction

The notion of a continuous mixture is somewhat less clear than in the discrete case, where each discrete mixture component has the intuitive meaning of representing a "cluster". Continuous mixture models generally do not have the same cluster type intuition, since the hidden space will usually be connected. What happens in the continuous case is that preferences for how the hidden variables are distributed are expressed.

Such models have many extremely useful properties, and are widely applied. They correspond to our belief that there is some continuous hidden process $p(h)$, from which (usually continuous) visible variables are observed, $p(v|h)$. The literature in this area is vast, and in this chapter we will consider only some of the most well known examples, beginning with some relatively numerically tractable modelling of subspaces.

20.1 Linear Subspace Methods

If data lies in a high dimensional space, we might hope that it lies close to a hyperplane, as in fig(20.1). We then can approximate each data point by using the vectors that span the hyperplane alone. I will sometimes refer to this small set of vectors as the "basis" set. Strictly speaking this is not a basis for the whole space, rather is is a 'basis' which approximately spans the space where the data is concentrated. Effectively, we are trying to choose a more appropriate low dimensional co-ordinate system that will approximately represent the data. Mathematically, we write If the dimension of the data space, $dim(x) = N$, our hope is that we can describe the data using only a small number M of vectors. If we can do so, we can reduce greatly the information needed to accurately describe the data.

In general, datapoints will not lie exactly on the hyperplane, and we may wish to model such non-linearity as noise. Two well known models, Factor Analysis and Principal Components Analysis, differ only in how this noise is dealt with.

Although not a particularly flexible density estimator, the clarity of the linear relationship between the variables may prove insightful, and is a useful starting point for more complex approaches.

The assumption is that the process generating the data is linear, dependent on a set of latent or hidden variables, h. For consistency with notation in other chapters, we will use h to represent the hidden or latent variables, and v to represent the visible or observable variables. Then the assumption of a linear subspace corresponds to

$$v = Wh + b + \boldsymbol{\epsilon} \tag{20.1.1}$$

where the noise $\boldsymbol{\epsilon}$ is Gaussian distributed, $\boldsymbol{\epsilon} \sim N(0; \Psi)$, and the matrix W parameterises the linear mapping. The constant bias b essentially sets the origin of the

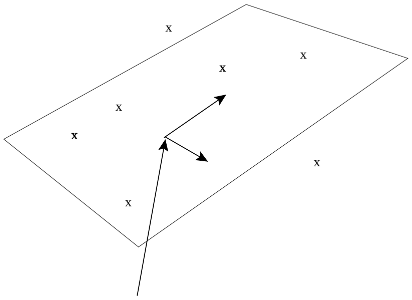


Figure 20.1: In linear modelling of a subspace, we hope that data in the high dimensional space lies close to a hyperplane that can be spanned by a smaller number of vectors. Here, each three-dimensional datapoint can be roughly described by using only two components.

coordinate system. The essential difference between PCA and Factor Analysis is in the choice of $\boldsymbol{\Psi}$.

Factor Analysis

In factor analysis, one assumes that the covariance for the noise is diagonal $\Psi = diag\left(\psi_1, \ldots, \psi_n\right)$. This is a reasonable assumption if we believe that each component of the data, x_i has Gaussian measurement error, independent of the other components. We see therefore that, given h, the data is assumed to be Gaussian distributed with mean $Wh + b$ and covariance Ψ

$$p\left(v|h\right) \propto e^{-\frac{1}{2}(v-Wh-b)^T \boldsymbol{\Psi}^{-1}(x-Wh-b)} \tag{20.1.2}$$

To complete the model, we need to specify the hidden distribution $p(h)$. Since tractability is always a concern for continuous distributions, an expedient choice is a Gaussian

$$p\left(h\right) \propto e^{-h^T h/2} \tag{20.1.3}$$

This therefore means that the coordinates h will be limited, and will most likely be concentrated around values close to 0. If we were to sample from such a $p(h)$ and then draw a value for v using $p(v|h)$, we would see that the v vectors that we sample would be look like a saucer in the v space.

Indeed, in this case we can easily calculate the exact form of $p(v)$:

$$p\left(v\right) = \int p\left(v|h\right) p\left(h\right) dh \tag{20.1.4}$$

Since $v = Wh + b + \boldsymbol{\eta}$ and we know that $p(h)$ is a zero mean Gaussian with unit covariance, and $\boldsymbol{\eta}$ is zero mean with Covariance $\boldsymbol{\Psi}$, v will be Gaussian distributed with mean b and covariance matrix $WW^T + \boldsymbol{\Psi}$.

The form of the covariance matrix is interesting and tells us some thing about the solution: Since the matrix W only appears in the final model $p(v)$ in the form

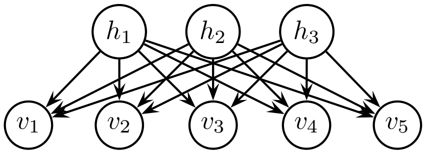


Figure 20.2: Graphical representation of factor analysis for a model with 3 hidden or latent variables, which generate the visible or output variable $v = (v_1, \ldots, v_5)^T$.

$WW^T + \mathbf{\Psi}$, an equivalent model is $WR(WR)^T + \mathbf{\Psi}$, where R is any orthogonal matrix $RR^T = I$. Hence, the solution space for W is not unique.

Warning! Since the so-called factor loadings W are equally likely as any rotated version of them, one should be very careful about interpreting the coefficients of the W – in particular, about attributing meaning to each of the values. Such practice is commonplace in the social sciences and, in general, is very poor science.

Training FA using EM

We assume that we are given a set of data $v^\mu, \mu = 1, \ldots, P$, and wish to adjust the parameters W, b and $\mathbf{\Psi}$ to maximise the likelihood of the observed data.

A natural way to train Factor Analysis, is to use our standard variational learning framework. Of course, in one could also attempt to maximise the likelihood directly (and the likelihood is relatively simple to calculate here). However, as usual, the variational procedure tends to converge rather more quickly, and is the one we shall describe here.

As usual, we need to consider the energy which, neglecting constants is

$$E = -\sum_\mu \left\langle \frac{1}{2} \left(v^\mu - Wh - b\right)^T \mathbf{\Psi}^{-1} \left(v^\mu - Wh - b\right) \right\rangle_{q^\mu(h)} - \frac{P}{2} \log \det \mathbf{\Psi}$$

It is left as an exercise for the interested reader to show that the following conditions hold at the maximum of the the energy. Maximising E with respect to b gives

$$b = \frac{1}{P} \sum_\mu v^\mu - W \frac{1}{P} \sum_\mu \langle h \rangle_{q^\mu(h)}$$

Maximising E with respect to W gives

$$W = AH^{-1}$$

where

$$A = \frac{1}{P} \sum_\mu c^\mu \langle h \rangle^T_{q^\mu(h)}$$

$$c^\mu = v^\mu - b$$

$$H = \frac{1}{P} \sum_\mu \left\langle hh^T \right\rangle_{q^\mu(h)}$$

Finally

$$\begin{aligned}\mathbf{\Psi} &= \frac{1}{P}\sum_{\mu} diag\left\langle (c^{\mu} - Wh)\,(c^{\mu} - Wh)^{T}\right\rangle_{q^{\mu}(h)} \\ &= diag\left\{\frac{1}{P}\sum_{\mu} c^{\mu}(c^{\mu})^{T} - 2WA^{T} + WHW^{T}\right\}\end{aligned}$$

The above recursions depend on the statistics $\langle h\rangle_{q^{\mu}(h)}$ and $\left\langle hh^{T}\right\rangle_{q^{\mu}(h)}$. Using the EM optimal choice

$$q^{\mu}(h) \propto p(v^{\mu}|h)p(h)$$

which is a Gaussian with covariance

$$\mathbf{\Sigma} = \left(I + W^{T}\mathbf{\Psi}^{-1}W\right)^{-1}$$

and mean

$$m^{\mu} = \langle h\rangle_{q^{\mu}(h)} = \left(I + W^{T}\mathbf{\Psi}^{-1}W\right)^{-1} W^{T}\mathbf{\Psi}^{-1}c^{\mu}$$

From which

$$H = \mathbf{\Sigma} + \frac{1}{P}\sum_{\mu} m^{\mu}(m^{\mu})^{T}$$

The above equations then define recursions in the usual EM manner. Unfortunately, the lack of a closed form solution to these equations means that FA is less widely used than the simpler PCA (and its probabilistic variant).

A nice feature of FA is that one can perform the calculations on very high dimensional data without difficulty. (In the standard PCA this is an issue, although these problems can be avoided – see the text).

Also, unlike in PCA, the matrix W that is learned need not be orthogonal.

What about a correlated $p(h)$?

A seemingly obvious extension of FA analysis is to consider dependent or, equivalently, in the Gaussian case, a correlated hidden distribution

$$p(h) = N0, \mathbf{\Sigma}_{H})$$

Does this really improve the representative power of the model? For notational simplicity, let's consider

$$v = Wh + \boldsymbol{\epsilon}$$

where $\boldsymbol{\epsilon} \sim N(0, \sigma^{2}I)$. Then it is clear that

$$v \sim N(0, W\mathbf{\Sigma}_{H}W^{T} + \sigma^{2}I)$$

which, may be written in the reparameterised form

$$v \sim N(0, W'W'^{T} + \sigma^{2}I)$$

where $W' \equiv W\mathbf{\Sigma}_{H}^{\frac{1}{2}}$. Hence, there is nothing to be gained from using a correlated Gaussian prior $p(h)$.

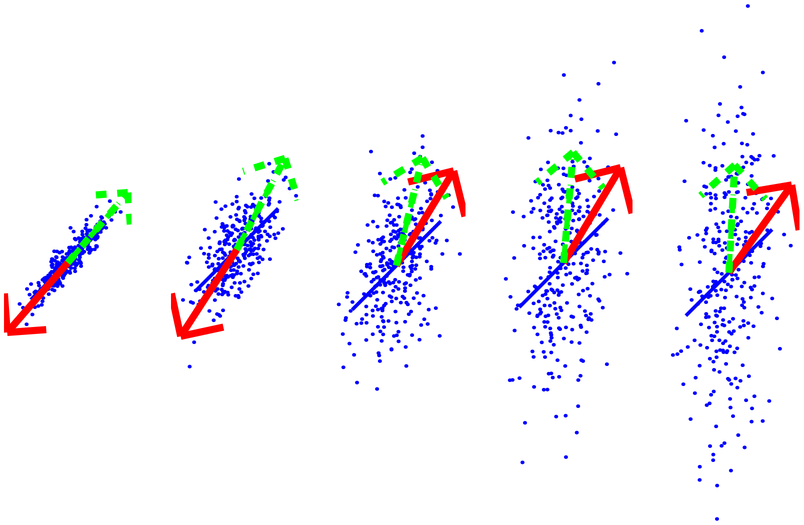


Figure 20.3: A comparison of factor analysis and PCA. The underlying data generating process is $y = x + \epsilon$, where ϵ is Gaussian noise of standard deviation σ. In the plots from left to right, σ takes the values 0.5, 1.2, 2, 3, 4. The FA solution is given by the solid arrow, and the PCA solution by the dashed arrow. The correct direction is given by the solid line. Note how the PCA solution "rotates" upwards as the noise level increases, whereas the FA solution remains a better estimate of the underlying correct direction.

Probabilistic Principal Components Analysis

(See also the section on Linear Dimension reduction for more details). PPCA is a special case of factor analysis in which the noise term Ψ is isotropic,$\Psi = \sigma^2 \mathbf{I}$. In this case, we can calculate the ML solution exactly. The optimal matrix W (so-called factor loadings), for a H dimensional hidden space and a V dimensional visible space, is given by

$$W = U_H \left(\lambda_H - \sigma^2 \mathbf{I}\right)^{\frac{1}{2}} R \tag{20.1.5}$$

where the H column vectors in U_H are the first H eigenvectors of the sample covariance matrix S,

$$S = \frac{1}{N} \sum_{\mu=1}^{P} \left(v^\mu - m\right) \left(v^\mu - m\right)^T \tag{20.1.6}$$

where m is the sample mean $\sum_\mu v^\mu / P$. λ_H is a diagonal matrix containing the corresponding eigenvalues of S. R is an arbitrary orthogonal matrix (representing an arbitrary rotation). For this choice of W, the optimal ML noise is given by

$$\sigma^2 = \frac{1}{V - H} \sum_{j=H+1}^{V} \lambda_j \tag{20.1.7}$$

where λ_j is the jth eigenvalues of S. This has the interpretation as the variance lost in the projection, averaged over the lost dimensions.

This means that we can rapidly find a ML linear subspace fit based on the eigendecomposition of the sample covariance matrix and sample mean.

An advantage of a proper probabilistic approach to PCA is that one can then, in a principled manner, for example, contemplate discrete mixtures of Principal Component Analysers, or indeed, a mixture of different kinds of models. Without a probabilistic framework, it is difficult to justify how a set models should be combined.

Standard Principal Component Analysis

There are several ways to understand PCA. However, in the current context, PCA is defined as the limit of PPCA in which $\sigma \to 0$ and $R = \mathbf{I}$. That is, the mapping from the latent space to the data space is deterministic. In this case, the columns of W are given by simply the eigenvectors of the sample covariance matrix, scaled by the square root of their corresponding eigenvalues.

Very High Dimensional Data

You might be wondering how it is possible to perform PCA on extremely high dimensional data. For example, if we have 500 images each of $1000 \times 1000 = 10^6$ pixels, the covariance matrix will be $10^6 \times 10^6$ dimensional – well beyond the storage capacities of many computers.

One approach around this difficulty is to perform the calculations in a lower dimensional space. Note that there can only be at most P non-zero eigenvalues.

Using X to denote the (zero mean) data and E the matrix of eigenvectors – this is non-square since there will be fewer eigenvalues than dimensions. We write the eigenvalues as a diagonal matrix Λ. The eigenvalue requirement is

$$XX^TE = E\Lambda \tag{20.1.8}$$

$$X^TXX^TE = X^TE\Lambda \tag{20.1.9}$$

$$X^TX\tilde{E} = \tilde{E}\Lambda \tag{20.1.10}$$

where we defined $\tilde{E} = X^TE$. The last line above represents the eigenvector equation for X^TX. This is a matrix of dimensions $P \times P$ – in the above example, a 500×500 matrix as opposed to a $10^6 \times 10^6$ matrix previously. We then can calculate the eigenvectors $\tilde{E}$ and eigenvalues Λ of this matrix more easily. Once found, we then use

$$E = X\tilde{E}\Lambda^{-1} \tag{20.1.11}$$

20.2 A Toy Comparision of FA and PPCA

We trained both PPCA and FA to try to model handwritten digits of the number 7. From a database of 100 such images, we fitted both PPCA and FA (100 iterations of EM) using 5 hidden units. The learned values for these models are in fig(20.4). To get a feeling for how well each of these models the data, we drew 25 samples from each model, as given in fig(**??**) and fig(20.5). In FA, clearly, the individual noise on each visible variable enables a cleaner representation of the regions of zero variance, compared to the PPCA approach. However, on the whole, despite the FA model being in principal a more powerful model, it does not here constitute a dramatic improvement over the PCA model.

Certainly one advantage of these probabilistic approaches is that they may now be used in discrete mixture models in a principled way, and this can indeed improve performance considerably.

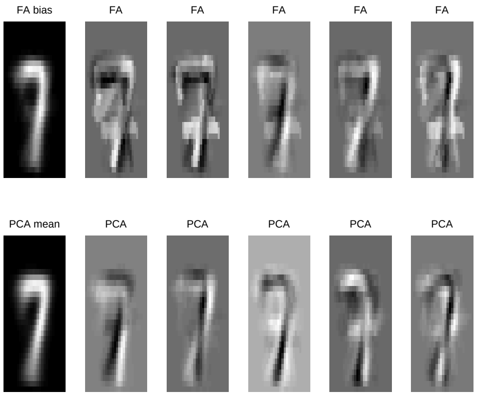


Figure 20.4: For a 5 hidden unit model, here are plotted the results of training PPCA and FA on 100 examples of the handwritten digit seven. Along with the PPCA mean and FA bias, the 5 columns of W are plotted for FA, and the 5 largest eigenvectors from PPCA are plotted.

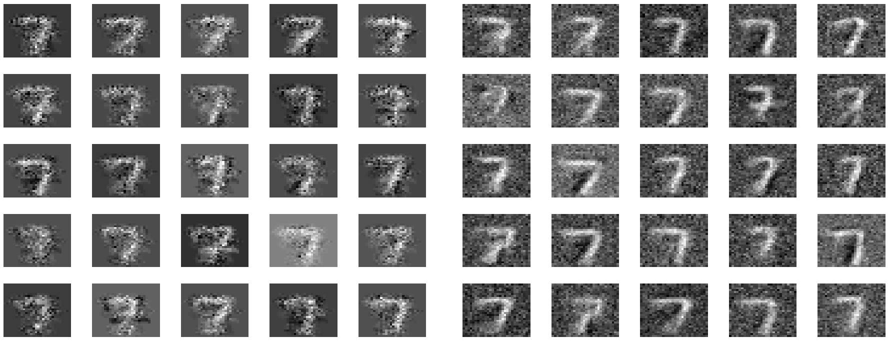
```
function [W,Psi,b]=fa(v,H,num_em_loops)

% Factor Analysis Training using EM

P = length(v); V = size(v{1},1);

b=randn(V,1); W = rand(V,H); Psi = rand(V,1);

for emloop =1:num_em_loops
  Sigma = inv(eye(H)+W’*diag(1./Psi)*W);
  mtot = zeros(H,H);
```



(a) Factor Analysis

(b) PCA

Figure 20.5: (a) 25 samples from the learned FA model. (b) 25 samples from the learned PPCA model.

```
  diagcont = zeros(V,1);
  A =zeros(V,H);
  btot = zeros(V,1);
  for mu=1:P
    c{mu} = v{mu}-b;
    diagcont = diagcont + c{mu}.^2;
    m{mu} = Sigma*W'*diag(1./Psi)*c{mu};
    mtot = mtot + m{mu}*m{mu}';
    A = A + c{mu}*m{mu}';
    btot = btot + v{mu}-W*m{mu};
  end
  Hmat = Sigma + mtot./P;
  A = A./P;
  diagcont = diagcont./P;
  diagWA = diag(W*A');
  Psi = diagcont -2*diagWA+diag(W*Hmat*W');
  b = btot./P;
  W = A/Hmat;
end
```

20.3 Non-linear Subspace Methods

20.3.1 Non linear Factor Analysis

The idea is the same as in FA, except that the transformation from the latent space to the data space is non-linear. That is

$$p\left(x|h\right) \propto \exp\left(-\frac{1}{2}\left(x-u\right)\Psi^{-1}\left(x-u\right)\right) \tag{20.3.1}$$

where $u = \boldsymbol{\phi}\left(h\right)$ where $\phi(t)$ is, in general, a non-linear function of t. If we take the same Gaussian prior as before, in general, we cannot calculate the integral over the latent space analytically anymore.

This can be approximated by

$$p\left(x\right) = \frac{1}{L}\sum_{l=1}^{L} p\left(x|h^{l}\right) \tag{20.3.2}$$

where we have sampled L latent points from the density $p\left(h\right)$. This is straightforward to do in the case of using a Gaussian prior on h.

What this means is that the density model is therefore a mixture of Gaussians, constrained somewhat through the non-linear function.

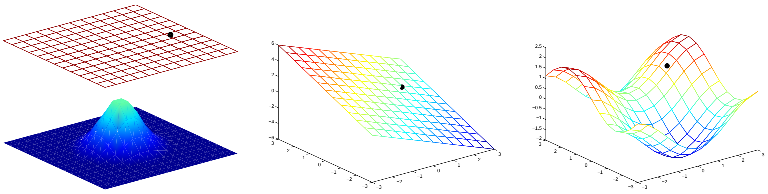
One approach is to parameterise the non-linearity as

$$\boldsymbol{\phi}(h) = \sum_{i} w_i \phi_i\left(h\right) \tag{20.3.3}$$

where the ϕ_i are fixed functions and the weights w_i form the parameters of the mapping. These parameters, along with the other parameters can then be found using variational learning (EM).

GTM

The Generative Topographic Mapping (GTM) is a special case of the non-linear factor analysis. It is a density estimator, but is most appropriate for very low (latent) dimensional representations of data, typically two or three. For this reason, it is mostly used to visualise data, and we shall describe this in some detail later.



(a) Latent Space (b) Data Space (c) Data Space

Figure 20.6: (a) The latent space usually corresponds to a low dimensional space, here 2 dimensional, so that a point h represented as the black dot in this space is specified by coordinates (h_1, h_2). Associated with this latent space is a prior belief about where the latent parameters are. Here this is a Gaussian distribution. (b) Each point in the latent space is mapped to some point in a typically higher dimensional space, here 3 dimensional. The mapping here is linear so that the object in the higher dimensional space is simply a plane – that is, a point in the lower dimensional space gets mapped to corresponding point (black dot) in the plane. Similarly, there will be an associated density function in this higher dimensional space, inherited from the density function in latent space. (c) Here the mapping from latent space to data space is non-linear, and produces a two dimensional manifold embedded in the three dimensional space.

20.4 Probabilistic PCA

A probabilistic version of PCA would be advantageous, since all the usual benefits inherited from the probabilistic viewpoint follow – automatic ways to do model selection, error bars, etc[38].

Since PCA fits a H dimensional hyperplane in a V dimensional space, it is natural to consider the model

$$p(v|h) = N(Wh, \sigma^2 I)$$

where V is a $V \times H$ matrix. Equivalently, we may write

$$v = Wh + \boldsymbol{\epsilon}$$

where $\boldsymbol{\epsilon} \sim N(0, \sigma^2 I)$. Here we use an isotropic noise,$\sigma^2 I$ in the original version of PCA, no preference was given for the error in any direction off the hyperplane. To complete the model, we need to specify a prior $p(h)$. To make things tractable, we specify this to be a Gaussian. The reader may satisfy herself that, without loss of generality, we may specify

$$p(h) = N(0, I)$$

What is the distribution $p(v)$? Formally, one can find this by calculating the integral $p(v) = \int_h p(v|h)p(h)$. Doing this, we find that $p(v)$ is Gaussian. This is an exercise that most people go though before they realise that there is a shortcut.

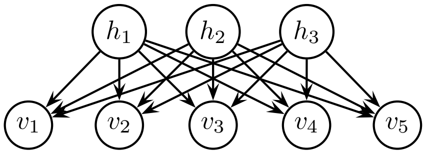


Figure 20.7: Graphical representation of PPCA for a model with 3 hidden or latent variables, which generate the visible or output variable $v = (v_1, \dots, v_5)^T$.

Since $p(v)$ is going to be Gaussian, all we need to do is find its mean and covariance. Since the noise is zero mean, then v will be zero mean. The covariance is given by

$$\left\langle vv^T \right\rangle = \left\langle (Wh + \boldsymbol{\epsilon})(Wh + \boldsymbol{\epsilon})^T \right\rangle$$

where the angled brackets denote an average with respect to all sources of fluctuations, namely the noise and the hidden distribution. Since these noise sources are uncorrelated, we have

$$\left\langle vv^T \right\rangle = WW^T + \sigma^2 I$$

Hence

$$p(v) = N(0, \mathbf{\Sigma} = WW^T + \sigma^2 I)$$

Now consider a dataset, $v^\mu, \mu = 1, \dots, P$. Under the usual assumption that the data are independently and identically distributed,

$$\begin{aligned}
p(v^1, \dots, v^P) &= \prod_{\mu=1}^{P} p(v^\mu) && (20.4.1)\\
&= \prod_{\mu=1}^{P} \frac{1}{\sqrt{\det 2\pi\mathbf{\Sigma}}} e^{-(v^\mu)^T \mathbf{\Sigma}^{-1} v^\mu/2} && (20.4.2)\\
&= \frac{1}{(\det 2\pi\mathbf{\Sigma})^{P/2}} e^{-\mathrm{trace}\left(\mathbf{\Sigma}^{-1}\sum_{\mu=1}^{P} v^\mu (v^\mu)^T/2\right)} && (20.4.3)\\
&= \frac{1}{(\det 2\pi\mathbf{\Sigma})^{P/2}} e^{-\mathrm{trace}\left(\mathbf{\Sigma}^{-1} S/2\right)} && (20.4.4)
\end{aligned}$$

where $S \equiv \sum_{\mu=1}^{P} v^\mu (v^\mu)^T$ is the sample correlation matrix of the data. To find the maximum likelihood setting of W, it's a little easier to consider the log likelihood:

$$L = -\frac{P}{2} \log\det \mathbf{\Sigma} - \frac{P}{2} \mathrm{trace}\left(\mathbf{\Sigma}^{-1} S\right)$$

where we dropped irrelevant constants. Now, differentiating with respect to W, and equating to zero, we obtain

$$0 = -\mathrm{trace}\left(\mathbf{\Sigma}^{-1}\partial_W \mathbf{\Sigma}\right) + \mathrm{trace}\left(\mathbf{\Sigma}^{-1}(\partial_W \mathbf{\Sigma})\mathbf{\Sigma}^{-1} S\right)$$

Using $\partial_W(WW^T) = W(\partial_W W^T) + (\partial_W W)W^T$, and using the symmetries, a stationary point is certainly given when

$$\mathbf{\Sigma}^{-1} W = \mathbf{\Sigma}^{-1} S \mathbf{\Sigma}^{-1} W$$

Or, assuming invertibility of $\mathbf{\Sigma}$,

$$W = S\mathbf{\Sigma}^{-1}W$$

This is reminiscent of an eigen-problem. If we represent W using its singular value decomposition, $W = ULV^T$, where U is a $V \times H$ dimensional matrix, $U^T U = I$, L is a diagonal, matrix containing the singular values, and V is a $H \times H$ orthogonal matrix, we have

$$SUL = U\left(\sigma^2 I + L^2\right) L$$

which gives the eigen-equation

$$SU = U\left(\sigma^2 I + L^2\right)$$

Hence U are the eigenvectors of the correlation matrix S and $\lambda_i = \sigma^2 + l_i^2$ are the eigenvalues. This constraint requires therefore $l_i = (\lambda_i - \sigma^2)^{1/2}$, which means that the solutions are of the form

$$W = E\left(\mathbf{\Lambda} - \sigma^2 I\right)^{1/2} R$$

where R is an arbitrary orthogonal matrix. The reader may verify, by plugging this solution back into the log-likelihood expression, that the eigenvalues and associated eigenvectors which maximise the likelihood correspond to the H largest eigenvalues of S. The standard, non-probabilistic variant PCA is given as the limiting case $\sigma^2 \to 0$ of PPCA.

Let's order the eigenvalues so that $\lambda_1 \geq \lambda_2, ... \geq \lambda_V$. The value for the log likelihood is then (see exercises)

$$L = -\frac{P}{2}\left(V\log(2\pi) + \sum_{i=1}^{H}\log\lambda_i + \frac{1}{\sigma^2}\sum_{i=H+1}^{V}\lambda_i + (V-H)\log\sigma^2 + H\right)$$

The reader may then verify that the optimal ML setting for σ^2 is

$$\sigma^2 = \frac{1}{V-H}\sum_{j=H+1}^{V}\lambda_j$$

Of course, we could have trained the above method using the standard EM algorithm. What's convenient about PPCA is that the solution is analytic, and boils down to a simple eigenproblem.

Note: in the above, we clearly need $\lambda_i \geq \sigma^2$ for the retained eigenvectors. However, for the ML solution this is guaranteed, since σ^2 is set to the average of the discarded eigenvalues, which must therefore be smaller that any of the retained eignvalues.

Mixtures of PPCA

One of the benefits of a probabilistic model is that we can form a mixture[39]. We'll talk more about this in the chapter on discrete mixture models.

20.5 Problems

Exercise 78 *In one dimension, $dim(x) = 1$, the Gaussian distribution is defined as*

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

You decide to fit a Gaussian to each class and use the ML estimates of the means $\hat{\mu}_1$ and $\hat{\mu}_2$. From the data, you find that the ML estimates of σ_1^2 and σ_2^2 are equal, that is, $\hat{\sigma}_1^2 = \hat{\sigma}_2^2$. Write down the explicit x value that defines the decision boundary.

Point out any potential numerical difficulties in directly comparing the values $p(c = 1|x)$ and $p(c = 2|x)$ and explain how you might overcome this.

In more than one dimension, the multi-variate Gaussian is defined as

$$p(x) = \frac{1}{\sqrt{\det 2\pi S}} e^{-\frac{1}{2}(x-\boldsymbol{\mu})^T S^{-1}(x-\boldsymbol{\mu})}$$

Given a dataset of iid samples, $x^1, \ldots, x^P$, derive an expression for the Maximum Likelihood estimator $\hat{\boldsymbol{\mu}}$.

Explain with the aid of a diagram, the nature of the decision boundary in the case that $\hat{\sigma}_1^2 \neq \hat{\sigma}_2^2$

Exercise 79 *In factor analysis there is a prior distribution $P(z) \sim N(0, I_m)$ over the m-dimensional latent variables, and a likelihood term $P(x|z) \sim N(Wz, \Psi)$ of a data point x given a value z of the latent variable vector.*

The posterior distribution for z is given by $\boldsymbol{P(z|x) \propto P(z)P(x|z)}$ (we don't need to worry too much about the normalization of the posterior distribution in this question).

Show that the posterior distribution is Gaussian with mean $(I_m + W^T\Psi^{-1}W)^{-1}W^T\Psi^{-1}x$, and state the covariance matrix of the posterior distribution.

Hint: If $P(y) \sim N(m, V)$ then $P(y) \propto \exp\{-\frac{1}{2}(y-m)^T V^{-1}(y-m)\}$.

Exercise 80 *Factor analysis and scaling. Assume that a m-factor model holds for x. Now consider the the transformation $\boldsymbol{y = Cx}$, where C is a non-singular diagonal matrix. Show that factor analysis is scale invariant, i.e. that the m-factor model also holds for y, with the factor loadings appropriately scaled. How must the specific factors be scaled?*

Exercise 81 *Consider data points generated from two different classes. Class 1 has the distribution $P(x|\mathcal{C}_1) \sim N(\mu_1, \sigma^2)$ and class 2 has the distribution $P(x|\mathcal{C}_2) \sim N(\mu_2, \sigma^2)$. The prior probabilities of each class are $P(\mathcal{C}_1) = P(\mathcal{C}_2) = 1/2$. Show that the posterior probability $P(\mathcal{C}_1|x)$ is of the form*

$$P(\mathcal{C}_1|x) = \frac{1}{1 + \exp -(ax + b)}$$

and determine a and b in terms of μ_1, μ_2 and σ^2. The function $f(z) = 1/(1+e^{-z})$ is known as the logistic function, and is a commonly-used transfer function in artificial neural networks.

Exercise 82 *Consider a one-dimensional probabilistic PCA model, so that $P(x|z) \sim N(wz, \sigma^2 I)$ and $P(z) \sim N(0,1)$. We are given data vectors x_i drawn from the model, but do not have the corresponding latent variables z_i, so this is a missing-data problem.*

Calculate the EM update for the factor loadings w.

Generalize this to the case where the latent variable is a vector.

Exercise 83 *Consider a $V \times H$ dimensional matrix $E = [e^1, \ldots, e^H]$ where $(e^a)^T e^b = \delta_{ab}$ for all $a, b \leq H$, and a $H \times H$ dimensional diagonal matrix D. Show that*

$$\det(EDE^T + \sigma^2 I) = (\sigma^2)^{(V-H)} \prod_{i=1}^{H} (D_{ii} + \sigma^2)$$

HINT : use the fact that the determinant of a matrix is the product of its eigenvalues.

Using $Se^a = \tilde{\lambda}^a e^a$, $a = 1, \ldots, V$ calculate explicitly the value of the expression

$$\mathrm{trace}\left((EDE^T + \sigma^2 I)^{-1} S\right)$$

in terms of the $\tilde{\lambda}^a$ and σ^2. HINT: use the fact that the trace of a matrix is the sum of its eigenvalues.

Exercise 84 *Explain why there is no closed form solution to the extension of PPCA to the case where the noise is diagonal, but non-isotropic. That is, $\langle \epsilon_i \epsilon_j \rangle = \sigma_i^2 \delta_{ij}$.*

20.6 Solutions

21 Independent Component Analysis

The cocktail Party Problem

You're a sound engineer working for a major TV company. A famous music awards takes place in your town, and you're invited to the post-awards cocktail party. The H guests request that no images be taken, but do consent to the placement of V microphones in the room. The fixed microphones record the H speakers, many of which speak at the same time. Your task is to isolate, from the mixed voices recorded, what is said by each individual.

Humans are very good at solving the cocktail-party problem – that is, to isolate the individual voices from a mixed set of voices, or other distracting background noises.

A little Physics

Sound is essentially a pressure wave, which may be represented by a real-valued variable $x(t)$, being the pressure level at a fixed spatial location at time t. For example, a classic "violin" note of a single characteristic base frequency traces out approximately a sinusoidal pattern, $x(t) = \sin(2\pi f(t + a) + b)$ where f is the frequency of the oscillation, and a and b are constants.

Linear Superposition

Another sound generator may give rise to a pressure $y(t)$ at the same spatial location. When both x and y are sounding, the waves add *linearly* to give a total pressure of $x(t) + y(t)$. It is a law of physics that, under ideal conditions, such sound waves add linearly – the so-called linear superposition of waves.

If we consider an extreme case of only one microphone, then our task is to identify $x(t)$ and $y(t)$ based on receiving $x(t) + y(t)$. Clearly, this is an ill-posed problem, and any solution we wish to attempt must make some extra assumptions about the problem.

Blind Source Separation

More formally, and more generally, the problem may be stated as, given a vector of observations $v = Wh + \boldsymbol{\eta}$, being a noisy (linear) combination of sources h, determine h. This problem is also known as "blind source separation".

Independence of the Sources

What constraints can we use to help make the problem well-posed? From a probabilistic viewpoint, we have specified $p(v|h)$. What about $p(h)$? Let's consider the case of two voices, $H = 2$, and one microphone, $V = 1$, and that each person is "talking to themselves" – that is, they are both loudly talking at the same time without reference to the other's conversation. In this case, a reasonable assumption would be that the two individuals are making sounds independent of the other, that is, $p(h) = p(h_1)p(h_2)$. Indeed, this independence of sources assumption (which clearly will not always be appropriate), is the usual baseline assumption made in solving this kind of problem. This assumption motivates the terminology Independent Components Analysis, where the components are the individual speaker's voices.

What about time?

In the above, no reference was made to time. In general, of course, a source $h(t)$ will have some dependency (due to smoothness) with a temporally close neighbour, say $h(t+1)$. In cases where such temporal correlations can be modelled, it clearly makes sense to include them, since this will give rise, generally, to better solutions. Historically, many of the original methods (for simplicity) assumed independence through time, so that $p(h_1(t), h_1(t+1)) = p(h_1(t))p(h_1(t+1))$. Though, clearly erroneous, surprisingly good results can sometimes be obtained, certainly in low-noise scenarios, which are commonplace assumptions in the early literature. For the moment, we will make the strong temporal independence assumption as well, and consider a couple of algorithms, before considering the more general case.

Gaussian Sources?

If the independent sources are assumed Gaussian, $p(h_i) = N(\mu_i, \sigma_i^2)$, then the model is the same as Factor Analysis. From FA, we know that the final model for the visible units is $p(v) = N(b, WW^T + \boldsymbol{\Psi})$. Indeed, we know that any "rotated" solution WR found is as equally likely as a W. This means that FA cannot identify (up to trivial permutations) the mixing matrix W, and means that we cannot solve the problem, as stated, by assuming that the sources are Gaussian.

No Output Noise

This section follows the derivation in David MacKay's book (see link on course web page).

To make things relatively simple, we consider the case $H = V$, that is, where the number of hidden sources is the same as the dimension of the visible observations. In this case,

$$p(v^\mu|h) = \boldsymbol{\delta}\left(v^\mu - Wh\right)$$

where $\boldsymbol{\delta}$ is the (vector) Dirac Delta function. In this case, we can easily integrate over the hidden units to write down the likelihood directly:

$$\log p(v^\mu) = -\log|\det(W)| + \sum_{i=1}^{H} \log p\left(\left[W^{-1}v^\mu\right]_i\right)$$

Using the notation $\widetilde{W} = W^{-1}$, this is

$$\log p(v^\mu) = \log|\det(\widetilde{W})| + \sum_{i=1}^{H} \log p\left(\left[\widetilde{W}v^\mu\right]_i\right)$$

One can then find the matrix $\widetilde{W}$ that maximises the likelihood by gradient ascent. However, the resulting algorithm is not well behaved, and a modified algorithm is preferred which can be shown to approximate a Newton update (see MacKay book).

1. Put v^μ through a linear mapping:

   $$a^\mu = \widetilde{W} v^\mu$$

2. Put a^μ through a non-linear map:

   $$z_i^\mu = \phi_i(a_i^\mu),$$

   where a popular choice for ϕ is $\phi(x) = -\tanh(x)$.

3. Put a^μ back through $\widetilde{W}$:

   $$v'^\mu = \widetilde{W}^T a^\mu$$

4. Adjust the weights in accordance to

   $$\Delta \widetilde{W} = \eta \left(\widetilde{W} + z^\mu (v'^\mu)^T \right)$$

   where η is a small learning rate parameter.

This algorithm tends to converge rapidly. Many other “fast” approaches based on direct maximisation of the likelihood result in similar algorithms.

“Sphering” the Data

Another approach which leads to simplified algorithms is to “sphere” the data. Since

$$v = Wh$$

the mean and the covariance of v are given by

$$\boldsymbol{\mu}_V = W \boldsymbol{\mu}_H$$

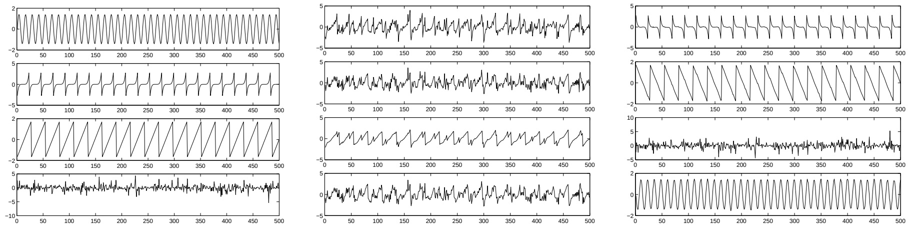
and

$$\boldsymbol{\Sigma}_V = W \boldsymbol{\Sigma}_H W^T$$

where $\boldsymbol{\mu}_H$ and $\boldsymbol{\Sigma}_H$ are the mean and covariance of the hidden distribution $p(h)$.

If we “sphere” the input data – that is, we transform (by a simple linear transformation) the observations such that they have zero mean and unit covariance, then we require, for this transformed data, $\boldsymbol{\mu}_H = 0$, and $W \boldsymbol{\Sigma}_H W^T = I$. Since $\boldsymbol{\Sigma}_H$ is diagonal (the sources are independent), and assuming that the sources all have the same variance, then (up to an arbitary prefactor), we may assume that $WW^T = I$. This also means that $W^T W = I$ (if we assume that W is a square matrix). (We can easily relax this to being $W^T W = \boldsymbol{\Psi}$, where $\boldsymbol{\Psi}$ is a diagonal matrix with positive entries). In the parlance of probability (see the section below), this means that the posterior distribution $p(h|v^\mu)$ is factorised. This is a great simplification, and enables one to develop fast algorithms.

However, one should bear in mind that, whilst sphereing the data can be done without any loss of generality in the noise-free case, in the noise case, the sphereing transformation *correlates* the previously independent additive noise. Thus, in those applications where noise is considered to be an inherent part of the data generation process, sphereing the data may be a bad idea (although this practice is widespread).



(a) (b) (c)

Figure 21.1: (a) The four original signals. (b) The mixed signals. (c) The four independent components found by the algorithm "fast-ica".

Toy example

In fig(21.1) we demonstrate an application of ICA to four time series fig(21.1a) which have been mixed together to form a four dimensional visible space, fig(21.1b). Visual inspection of the mixed signals does not help much in identifying the underlying independent signals. Using the algorithm "fast-ica", which is based on a noise-free maximum-likelihood fixed point criterion, we find the decorrelated signals fig(21.1c). Of course, the decorrelated signals can be permuted from the original signals, and their sign may be reversed.

Variational algorithms

In a more general, noisy framework, a natural way in our framework to find W is to use the variational approach. The energy is given by

$$E = -\frac{1}{2\sigma^2}\left\langle (v^\mu - Wh)^2 \right\rangle_{q^\mu(h)} - \frac{V}{2}\log\sigma^2 + \sum_{i=1}^{H} \left\langle \log p(h_i) \right\rangle_{q^\mu(h_i)}$$

Optimising this with respect to W gives

$$\sum_\mu v^\mu \left\langle h \right\rangle^T_{q^\mu(h)} = W \sum_\mu \left\langle hh^T \right\rangle_{q^\mu(h)}$$

which can be solved by Gaussian Elimination (or taking the inverse if desired).

The Posterior Distribution

What about the distribution $q^\mu(h)$? The optimal (EM) choice is

$$\begin{aligned} q^\mu(h) &\propto p(v^\mu|h)p(h) \\ &= e^{-\frac{1}{2\sigma^2}(v^\mu - Wh)^2 + \sum_{i=1}^H \log p(h_i)} \\ &\propto e^{-\frac{1}{2\sigma^2}\left(h^T W^T W h - 2(v^\mu)^T W h\right) + \sum_{i=1}^H \log p(h_i)} \end{aligned}$$

There are some difficulties with this expression. Since $p(h_i)$ cannot be Gaussian, the term in the exponent is not quadratic – that is, the posterior distribution is non-Gaussian. Unless the posterior distribution is factorised (which means $W^T W$ is a diagonal matrix), then it is unlikely that we will be able to calculate exactly the necessary moments $\langle h_i \rangle_{q^\mu}$ and $\langle h_i h_j \rangle_{q^\mu}$.

There are several things we could do at this point – for example, we could restrict attention to orthogonal matrices W, and re-derive an algorithm for which then the posterior would be diagonal. As an alternative, we present here an application of the more general variational framework.

Caveat : Simple Variational methods may not work

Consider the case when we have learned the correct parameter W. In this case, since, $v \approx Wh^0$, then the posterior roughly has the form

$$e^{-\frac{1}{2\sigma^2}\left(h^TW^TWh-2(Wh^0)^TWh\right)+\sum_{i=1}^{H}\log p(h_i)}$$

The point here is that the quadratic expression has roughly the same numerical contribution as the linear term – that is, it will not be dominated by the linear term, and posterior will always have significant correlations. This is the reason why simple factorised variational methods like the one below do not work well, and slightly more complex methods are required.

Restricting $q^\mu(h)$

According to the variational approach, we can choose any distribution q^μ from a given class of distributions, and the best q^μ is that one that minimises the KL divergence $KL(q^\mu(h), p(h|v^\mu))$. Two natural candidates for q^μ would be either a Gaussian, or a factorised distribution. The Gaussian case is left as an exercise for the reader, and here we concentrate on a free-form optimisation with respect to the class of factorised approximations.

Optimal Factorised Distributions

Let $p(x)$ be a (in general) non-factorised distribution. Consider that we wish to find a factorised approximation $q(x) = \prod_i q(x_i)$to this distribution using the KL divergence.

$$KL(q(x), p(x)) = \langle \log q(x) - \log p(x)\rangle_{q(x)} = \sum_i \langle \log q(x_i)\rangle_{q(x_i)} - \langle \log p(x)\rangle_{\prod_i q(x_i)}$$

If we isolate the contribution of $q(x_j)$ in the above equation, keeping the other components $q(x_k), k \neq j$ fixed, we have

$$\langle \log q(x_j)\rangle_{q(x_j)} - \left\langle \langle \log p(x)\rangle_{\prod_{k\neq j} q(x_k)} \right\rangle_{q(x_j)}$$

which is the KL divergence between $q(x_j)$ and $\langle \log p(x)\rangle_{\prod_{k\neq j} q(x_k)}$. Hence, the optimal choice is

$$q(x_j) \propto e^{\langle \log p(x)\rangle_{\prod_{k\neq j} q(x_k)}}$$

Application to ICA

In ICA, $p(x)$ in the above is replaced by $p(v^\mu, h)$. Hence

$$\log p(v^\mu, h) = \log p(v^\mu|h) + \log p(h)$$

The reader may verify the following:

$$q^\mu(h_i) \propto p(h_i) e^{-\frac{1}{2\sigma^2}\left(U_{ii}h_i^2 + 2h_i c_i^\mu\right)}$$

where

$$U = W^T W$$

and

$$c_i^\mu = \sum_{j \neq i} U_{ij} \left\langle h_j \right\rangle_{q^\mu(h_j)} - b_i^\mu$$

$$b^\mu = W^T v^\mu$$

One then iterates the above equations, each time finding a new c_i^μ for each hidden component i. The one dimensional averages in general need to be computed numerically.

Whilst this algorithm may work in principal, in practice, convergence tends to be rather poor, possibly due to the problem not being well conditioned, and cause a fixed point algorithm to be unstable.

A likely explanation for this difficulty is that the posterior correlations are too strong to make a factorised approximation a reasonable assumption.

The use of such variational methods for ICA therefore needs to be applied with some caution. To be continued next year....

BASCIALLY, THIS IS SOLVED BY OPPER AND WINTHER – THEY USE BP TO DO THE INFERENCE – WORKS MUCH BETTER. CITE THEIR PAPER ON THIS AND DO THE APPLICATION HERE.

21.0.1 An alternative variational approach

$$p(h|v) = p(v|h)p(h)/p(v)$$

If we consider the special case of Gaussian noise (for convenience here), and bear in mind the factorial nature of the prior:

$$p(h|v) \propto \exp -\frac{1}{2}(v - Wh)/\sigma^2 \prod_i p(h_i)$$

The non-Gaussian nature of the posterior arises from the non-Gaussian nature of the prior components $p(h_i)$. In the standard EM learning algorithm, we need to compute the first and second order moments of h given v. Consider a general KL variational approximation:

$$KL(q(h|v)|p(h|v)) = q(h|v) \log q(h|v) - q(h|v) \left(\log p(v|h) - \log p(h) \right) + \log p(v)$$

Plugging in the form of the prior and likelihood terms:

$$KL = q(h|v) \log q(h|v) - q(h|v) \left(-\frac{1}{2\sigma^2} \left(v - Wh\right)^2 + \sum_i \log p(h_i) \right)$$

Now, let's consider what happens for the special class of prior $p(h)$ that can be written as the Gaussian maximisation (the Laplace transform would also be equally useful):

$$p(h) = \max_t e^{-\frac{1}{2}h - t^2} \phi(t)$$

Then

$$KL \geq q(h|v)\log q(h|v) - q(h|v)\left(-\frac{1}{2\sigma^2}(v - Wh)^2 - \frac{1}{2}\sum_i (h_i - t_i)^2\right) + \log\phi(t_i)$$

To find a posterior approximation $q(h|v)$ we can then maximise the rhs alternately wrt to t and $q(h|v)$. For given t, the optimal $q(h|v)$ is clearly Gaussian since the rhs contains only linear and quadratic terms in h. We can then maximise wrt the variational parameters t. This leads to a simple update formula which will directly yield an estimate of the first and second moments. Indeed, in principle, there is no need to look for other bounds since if the global maximium can be found, then the moments will be computed exactly (since the bound potentially touches the function exactly). However, it may be that other bounds result in procedures less likely to get stuck in local optima.

Another interesting point here is that one can (of course) just use the same idea as above to derive a method that learns the parameters W at the same time.

21.0.2 Static Case : ICA with external Inputs

Question : if we use external inputs, surely this breaks the non-identifiability problem? This is also related maybe to Silvia's work. Would be good to do this. Could have nice impact on people who use ICA for feature extraction.

21.1 Expectation Propagation

The posterior is of the form

$$p(h|v) \propto p(v|h)\prod_i p(h_i)$$

The term $p(v|h)$ is unproblematic. However, the non-Gaussian factors $p(h_i)$ make the posterior distribution $p(h|v)$ non-Gaussian.

One approach to dealing with this is called EP. In general, EP works by replacing a difficult term in a distribution by a simpler distribution, and finding the optimal simpler distribution.

Let's replace the difficult factors $p(h_i)$ by simpler Gaussian factors $q(h_i)$.

Then the approximation is

$$p(h|v) \propto p(v|h)\prod_i p(h_i) \approx q(h) \propto p(v|h)\prod_i q(h_i)$$

The distribution $p(v|h)\prod_i q(h_i)$ is easy – this is just a Gaussian. The idea is then to replace a single term $q(h_i)$ by a correct term $p(h_i)$, and model the effect that this has on the posterior, by adjusting $q(h_i)$ appropriately.

$$\tilde{q} \propto p(v|h)p(h_i)\prod_{j\neq i} q(h_j)$$

$$KL(\tilde{q}|q) = \langle \log \tilde{q}\rangle_{\tilde{q}} - \langle \log q\rangle_{\tilde{q}}$$

Minimising the above with respect to $q(h_i)$ means that we only need to consider the term $\langle \log q \rangle_{\tilde{q}}$ since the first term does not depend on $q(h_i)$ (The fact that $\tilde{q}$ is not normalised is irrelevant.)

Problems with EM

In the small noise limit, EM doesn't work correctly. To see this, the posterior distribution in this case has covariance and mean

$$\sigma^2 \left(W^\mathsf{T} W\right)^{-1}$$

$$\left(W^\mathsf{T} W\right)^{-1} W^\mathsf{T} v^\mu$$

When we plug this into the formula for the update, we have

$$\sum_\mu v^\mu (v^\mu)^\mathsf{T} W \left(W^\mathsf{T} W\right)^{-1} \left(\sum_\mu \left(W^\mathsf{T} W\right)^{-1} W^\mathsf{T} v^\mu \left(v^\mu\right)^\mathsf{T} W \left(W^\mathsf{T} W\right) \right)^{-1}$$

If we let $S = \sum_\mu v^\mu (v^\mu)^\mathsf{T}$, we have then

$$SW \left(W^\mathsf{T} W\right)^{-1} \left(\left(W^\mathsf{T} W\right)^{-1} W^\mathsf{T} S W \left(W^\mathsf{T} W\right)^{-1} \right)^{-1}$$

which simplifies to

$$SW \left(\left(W^\mathsf{T} W\right)^{-1} W^\mathsf{T} S W \right)^{-1}$$

which (need some assumptions of invertibility) is just

$$W$$

This means that the EM update gives $W^{new} = W!$. The problem is that the prior term is completely lost in this approach. Even if we use the gradient trick, we will still have the same problem since the gradient will be dominated by the constraint that (up to first order) $v = Wh$.

21.1.1 A possible solution

Consider the result

$$\frac{1}{\sqrt{\det 2\pi \boldsymbol{I}}} \int_t e^{-\frac{1}{2}\left(t^\mathsf{T} t - 2t^\mathsf{T} a\right)} = e^{\frac{1}{2}a^\mathsf{T} a}$$

This means that

$$p(v) = \int_h p(v|h) \prod_i p(h_i)$$

$$p(v) = \int_h e^{-\frac{1}{2\sigma^2}(v-Wh)^\mathsf{T}(v-Wh)} \prod_i p(h_i)$$

Let $a = i\left(v - Wh\right)/\sigma$ then

$$p(v) = K \int_{h,t} e^{-\frac{1}{2}t^\mathsf{T} t - 2it^\mathsf{T} \frac{v-Wh}{\sigma}} \prod_i p(h_i) dt$$

where $K = \frac{1}{\sqrt{\det 2\pi \boldsymbol{I}}}$. Then change variables $f = t/\sigma$. Then $df = dt/\sigma^H$

$$p(v) = K\sigma^H \int_{f,h} e^{-\frac{\sigma^2}{2} f^\mathsf{T} f + i f^\mathsf{T} (v - Wh)} \prod_i p(h_i) df$$

$$p(v) = K\sigma^H \int_f e^{-\frac{\sigma^2}{2} f^\mathsf{T} f + i f^\mathsf{T} v} \prod_j \int_{h_j} p(h_j) e^{i h_j [W^\mathsf{T} f]_j} dh_j$$

Then

$$\frac{\partial}{\partial W_{ab}} p(v) = K\sigma^H \int_f e^{-\frac{\sigma^2}{2} f^\mathsf{T} f + i f^\mathsf{T} v} \int_{h_a} i f_b h_a p(h_a) e^{i h_a [W^\mathsf{T} f]_a} dh_a$$

Then this is

$$\frac{\partial}{\partial W_{ab}} p(v) = K\sigma^H \int_{f,h_a} e^{-\frac{\sigma^2}{2} f^\mathsf{T} f + i f^\mathsf{T} (v - h_a w_a)} i f_b h_a p(h_a)$$

where $w_a = W_{ja}$. Then defining

$$\tilde{v} = i\left(v - h_a w_a\right)/\sigma^2$$

$$\frac{\partial}{\partial W_{ab}} p(v) = K\sigma^H \int_{f,h_a} e^{-\frac{\sigma^2}{2}\left(f^\mathsf{T} f - 2 f^\mathsf{T} \tilde{v} + \tilde{v}^\mathsf{T}\tilde{v} - \tilde{v}^\mathsf{T}\tilde{v}\right)} i f_b h_a p(h_a)$$

$$\frac{\partial}{\partial W_{ab}} p(v) = K \int_{h_a} e^{\frac{\sigma^2}{2}\tilde{v}^\mathsf{T}\tilde{v}} \tilde{v}_b i h_a p(h_a)$$

This is then

$$-\frac{1}{\sigma^2} K \int_{h_a} e^{-\frac{1}{2\sigma^2}(v - h_a w_a)^\mathsf{T}(v - h_a w_a)} \left[v - h_a w_a\right]_b h_a p(h_a)$$

which is

$$-\frac{1}{\sigma^2} K e^{-\frac{1}{2\sigma^2}\left(v^\mathsf{T} v - \beta^2/\alpha\right)} \left\langle \left[v - h_a w_a\right]_b h_a p(h_a)\right\rangle_{q(h_a)}$$

where $q(h_a)$ is a Gaussian with mean β/α and variance σ^2/α,

$$\alpha = w_a^\mathsf{T} w_a, \qquad \beta = v^\mathsf{T} w_a$$

For the case that σ^2 is very small, this will be zero unless the exponent is zero, which is true, for example if the data has been whitened. Then the derivative will be huge unless $v = Wh$. When this is true, then the derivative will be

Ignoring for the moment issues about taking logs of negative or complex values, we can write this as

$$p(f) = \sigma^H \int_f e^{-\frac{\sigma^2}{2} f^\mathsf{T} f}$$

$$p(v|f) = K e^{i f^\mathsf{T} v} \prod_j \int_{h_j} p(h_j) e^{i h_j [W^\mathsf{T} f]_j} dh_j$$

Then

$$\frac{\partial}{\partial W} \log p(v) = \left\langle \frac{\partial}{\partial W} \log p(v|f) \right\rangle_{p(f|v)}$$

Which gives (I think)

$$\frac{\partial}{\partial W_{ab}} \log p(v) = i \left\langle f_b \int_{h_a} \frac{h_a p(h_a) e^{i h_a [W^\mathsf{T} f]_a}}{\int_{h_a} p(h_a) e^{i h_a [W^\mathsf{T} f]_a}} \right\rangle_{p(f|v)}$$

22 Dynamic Bayesian Networks : Discrete Hidden Variables

22.1 The Importance of Time

In many physical applications, time plays a crucial role. Indeed, many models of the physical world are based on differential equations, such as Newton's First Law

$$\frac{dx}{dt} = v$$

meaning that things keep moving in a straight line when not acted upon by an external force. A computational approximate implementation of this differential equation is given by a difference equation :

$$\frac{x(t+\delta) - x(t)}{\delta} = v \Rightarrow x(t+\delta) = x(t) + v\delta$$

For convenience, we can rescale time so that a unit time step represents δ seconds :

$$x(t+1) = x(t) + c, \qquad c = const.$$

In an imperfect world, however, physical processes are often perturbed slightly by noise, (perhaps slight air currents are disturbing the otherwise perfect motion, for example). These effects could be modelled by

$$x(t+1) = x(t) + c + \epsilon(t)$$

where $\epsilon(t)$ is a random variable sampled from some distribution. For example, if the noise is Gaussian with zero mean and variance σ^2, then

$$p(x(t+1)|x(t)) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x(t+1)-x(t)-c)^2}$$

Markov Process

This is an example of a Markov chain. Processes are Markov if the future state $x(t+1)$ only depends on the current state $x(t)$. This is called a first order Markov process which refers to the dependency on only the first immediately preceding state in time. More formally,

$$p(x(t+1)|x(t), x(t-1), \ldots, x(1)) = p(x(t+1)|x(t))$$

Therefore the joint distribution admits the factorisation

$$p(x(T), \ldots, x(1)) = p(x(1)) \prod_{t=1}^{T-1} p(x(t+1|x(t))$$

Other common physical systems depend on second order differential equations. For example, Newton's 2nd law states that, under a force F,

$$\frac{d^2x}{dt^2} = k_1, \qquad k_1 = const$$

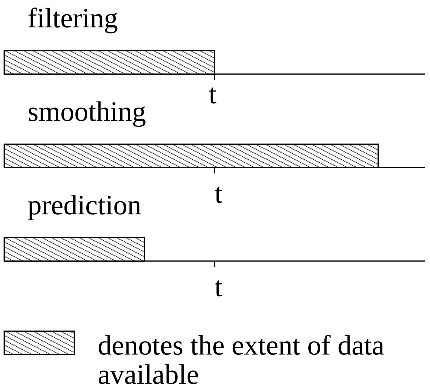


Figure 22.1:

Again, if we were to write down a discrete time difference approximation, we would have an equation of the form (including a possibly stochastic noise term):

$$x(t+1) - 2x(t) + x(t-1) = k_2 + \epsilon(t)$$

Or

$$x(t+1) = 2x(t) - x(t-1) + k_2 + \epsilon(t)$$

Second Order

Here the state of the future world depends on the present and the immediate past[1]. In general, then we would have

$$p(x(T), \dots, x(1)) = p(x(1), x(2)) \prod_{t=1}^{T-2} p(x(t+2|x(t+1), x(t))$$

The above is an example of a second order Markov process. The generalisation to a k^{th} order process is obvious. We also call these models Markov Models. In the above examples it would be natural to consider the variables $x(t)$ to be continuous.

Inference Problems

- $p(h_t|v_1, \dots, v_t)$ filtering
- $p(h_t|v_1, \dots, v_s)$ $t > s$, prediction
- $p(h_t|v_1, \dots, v_u)$ $t < u$, smoothing
- $p(v_1, \dots, v_T)$ likelihood calculation
- Find sequence $h_1^*, \dots, h_T^*$ that maximizes $p(h_1, \dots, h_T|v_1, \dots, v_T)$ [Viterbi alignment]

Transition Diagrams and Finite State Automata

Models with discrete variables are common, and have significant application in many fields, ranging from sequence modelling in Bioinformatics to text and speech processing. Here is a simple example:

[1] It is a deep (and at least to me, somewhat mysterious) property that all laws of physics are only maximally second order differential equations, and hence can be well approximated by second order stochastic differential equations.

Hilde is an interesting chimp. She has been trained to press the buttons 1, 2 and 3 always in sequence, although the starting state doesn't matter. For example, 2,3,1,2,3,1,2,3,1,2,3. Hilde is quite good at this, but sometimes makes a mistake and presses a button out of sequence. The probability that she makes a transition from state j to state i, $p(i|j)$ is given by the matrix elements below:

	1	2	3
1	0.1	0.1	0.8
2	0.8	0.1	0.1
3	0.1	0.8	0.1

which can be represented as a matrix $p_{ij} \equiv p(i|j)$. Alternatively, a state transition diagram can be used, as in fig(22.2) below. To make this more informative, one sometimes shows also the values of the transitions on the links. This is a generalisation of Finite State Automata. In FSAs, the transitions are deterministic, the corresponding table entries being either 0 or 1.

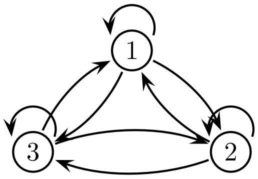


Figure 22.2: A state transition diagram, for a three state markov chain. Note that a state transition diagram is not a graphical model – it simply graphically displays the non-zero entries of the transition matrix $p(i|j)$

Hidden Markov Models

Vernon is another, slightly less reliable chimp. He has been trained such that whenever he sees that Hilde has pressed either button 1 or 2, he grunts A and whenever he sees a 3 he grunts B. However, he also makes mistakes, as characterised by the probability state transition matrix below,

	1	2	3
A	0.7	0.6	0.25
B	0.3	0.4	0.75

which can be represented by $p_{ij} \equiv p(v(t) = i|h(t) = j)$.

Flippa is a super clever dolphin. She has sent to her a sequence of grunts from Vernon, e.g. B,A,A,B,A,B and has been trained to figure out from the sequence Vernon grunts, what is is that Hilde typed. Of course, this is not strictly solvable in an exact sense. Flippa reports back to her trainer the most likely sequence of buttons pressed by Hilde.

A model of the above situation is given in fig(22.3), a so called Hidden Markov Model (HMM)[40].

The idea here is that there is some internal (hidden) dynamics (modelled by Hilde in our example above), and an observation or visible variable for each of the hidden variables (modelled by Vernon). In our example, each hidden variable would have 3 states, and each visible variable would have 2 states. We assume stationarity

Stationarity

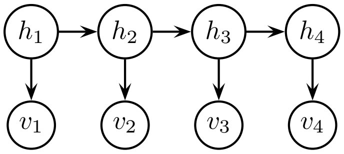


Figure 22.3: A Hidden Markov Model.

– that is, the model for each time step to time step holds for all times. The distribution is therefore fully described by the

Transition matrix

$$A_{i',i} = p(h_{t+1} = i'|h_t = i)$$

an Emission matrix

$$B_{j,i} = p(v_t = i|h_t = j)$$

and finally a Prior

$$\pi_i = p(h_1 = i).$$

Structured Transition Matrices

In some applications (often in Automatic Speech Recognition), it is useful to constrain the state sequence to go 'forwards' through time. For example, if the HMM models a certain word, where the utterance may be broken into a subsequence of phonemes, then we expect the word to follow a set of phonemes/states, and not to go back to them. Sometimes, we might stay in one state longer than another, or even maybe skip some states. However, the important thing is that there may be a sense of structure therefore to the transition matrices. For those that go forwards in this sense, the structure of the transition matrices is upper triangular (or lower, depending on your tastes), or even a banded triangular matrix. Such forward constraints describe a so-called left-to-right transition matrix.

JTA for HMMs

How can Flippa solve her problem? That is find $\underset{h_1,\ldots,h_T}{\text{argmax}}\, p(h_1, \ldots, h_T|v_1, \ldots, v_T)$? As we saw previously, such most probable state calculations can be carried out by a slight modification of the JTA. The first step then is to find a Junction Tree for the HMM. The HMM is already moralised and triangularised. A suitable JT, along with a valid assignment of the potentials is given in fig(22.4).

JTA and the Forward-Backward Algorithm for HMMs

A valid sequence of absorptions would consist of a forward sweep, and then a backward sweep.

Forward sweep For $t = 1, \ldots T-2$:
Absorb from $\Psi(v_t, h_t)$ to $\Psi(h_t, h_{t+1})$. Then from $\Psi(h_t, h_{t+1})$ to $\Psi(h_{t+1}, h_{t+2})$. Repeat.
Absorb from $\Psi(v_{T-1}, h_{T-1})$ to $\Psi(h_{T-1}, h_T)$. Absorb from $\Psi(h_{T-1}, h_T)$ to $\Psi(v_T, h_T)$.

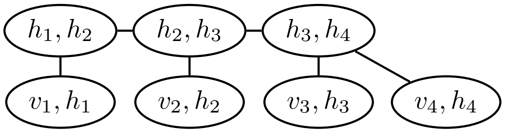


Figure 22.4: A first order Hidden Markov Model. A suitable assignment of the potentials is: $\Psi(h_1, h_2) = p(h_1)p(h_2|h_1)$, $\Psi(v_1, h_1) = p(v_1|h_1)$, $\Psi(h_2, h_3) = p(h_3|h_2)$, $\Psi(v_2, h_2) = p(v_2|h_2)$, $\Psi(h_3, h_4) = p(h_4|h_3)$, $\Psi(v_3, h_3) = p(v_3|h_3)$,$\Psi(v_4, h_4) = p(v_4|h_4)$. All separator potentials (not shown) are initially set to unity.

Backward sweep Absorb from $\Psi(v_T, h_T)$ to $\Psi(h_{T-1}, h_T)$
For $t = T-1, \ldots 2$:
Absorb from $\Psi(h_t, h_{t+1})$ to $\Psi(v_t, h_t)$. Then from $\Psi(h_t, h_{t+1})$ to $\Psi(h_{t-1}, h_t)$.
Repeat.
Absorb from $\Psi(h_1, h_2)$ to $\Psi(v_1, h_1)$

This formulation of the JTA is equivalent to the classic Forward-Backward algorithm for HMMs. Flippa can solve her problem by using the *max* version of absorption. The complexity of hidden variable inference in a HMM scales linearly in T, V (the number of visible states) and quadratically in H, giving a time complexity of $O(TH(V + H))$. The important point is that it scales linearly with time so that, even with a lengthy observed sequence, we can still perform inference quickly.

22.1.1 Parallel and Sequential Inference

We are interested in the so-called 'smoothed' posterior $p(h_t|v_{1:T})$. There are two main approaches to computing this.

Parallel Method

$$\begin{aligned} p(h_t|v_{1:T}) &\propto p(h_t, v_{1:T}) \\ &\propto p(h_t, v_t, v_{t+1:T}) \\ &\propto p(h_t, v_{1:t}, v_{t+1:T}) \\ &\propto p(v_{t+1:T}|h_t, \cancel{v_{1:T}})p(h_t|v_{1:t}) \end{aligned} \tag{22.1.1}$$

Hence, in this method, we need to compute the so-called 'filtered' posteriors $p(h_t|v_{1:t})$, and combine them with the conditional terms $p(v_{t+1:T}|h_t)$. As we will see below, the filtered posteriors are easy to get using a Forward recursion. The terms $p(v_{t+1:T}|h_t)$ are also easy to obtain using a Backward recursion. The Forward and Backward recursions may be called in parallel, with their results combined to obtain the smoothed posterior.

$$
\begin{aligned}
p(h_t|v_{1:T}) &\propto \sum_{h_{t+1}} p(h_t, h_{t+1}, v_{1:T}) \\
&\propto \sum_{h_{t+1}} p(h_t|h_{t+1}, v_{1:t}, \cancel{v_{t+1:T}})p(h_{t+1}|v_{1:T})
\end{aligned}
\tag{22.1.2}
$$

This then gives a Backwards recursion for $p(h_t|v_{1:T})$. As we will see below, the term $p(h_t|h_{t+1}, v_{1:t}$ may be computed based on the filtered results $p(h_t|v_{1:t}$.

The classical $\alpha - \beta$ recursions

As we mentioned in the chapter on inference, the difference between Belief Propagation and the Junction Tree algorithm on singly-connected structures, is just in the parameterisation of the messages.

For readers less familiar with the probabilistic approach, we'll briefly describe here an alternative derivation of Belief Propagation inference on simple chain distributions[10, 19]. The presentation here follows that presented in[41]. First, let's simplify the notation, and write the distribution as

$$p = \prod_t \phi\left(x_{t-1}, v_{t-1}, x_t, v_t\right)$$

where $x_t \equiv h_t$, and $\phi\left(x_{t-1}, v_{t-1}, x_t, v_t\right) = p(x_t|x_{t-1})p(v_t|x_t)$. Our aim is to define 'messages' ρ, λ (these correspond to the α and β messages in the Hidden Markov Model framework[40, 42]) which contain information from past observations and future observations respectively. Explicitly, we define $\rho_t(x_t) \propto p(x_t|v_{1:t})$ to represent knowledge about x_t given all information from time 1 to t. Similarly, $\lambda_t(x_t)$ represents knowledge about state x_t given all information from the future observations from time T to time $t+1$. In the sequel, we drop the time suffix for notational clarity. An important point is that $\lambda(x_t)$ is not a distribution in x_t, but rather implicitly defined through the requirement that the marginal inference is then given by

$$p(x_t|v_{1:T}) \propto \rho\left(x_t\right)\lambda\left(x_t\right) \tag{22.1.3}$$

Similarly, the pairwise marginal is given by

$$p(x_{t-1}, x_t|v_{1:T}) \propto \rho\left(x_{t-1}\right)\phi\left(x_{t-1}, v_{t-1}, x_t, v_t\right)\lambda\left(x_t\right) \tag{22.1.4}$$

Taking the above equation as a starting point, we can calculate the marginal from this

$$p(x_t|v_{1:T}) \propto \sum_{x_{t-1}} \rho\left(x_{t-1}\right)\phi\left(x_{t-1}, v_{t-1}, x_t, v_t\right)\lambda\left(x_t\right) \tag{22.1.5}$$

Consistency with equation (24.2.1) requires (neglecting irrelevant scalings)

$$\rho\left(x_{t-1}\right)\lambda\left(x_t\right) \propto \sum_{x_{t-1}} \rho\left(x_t\right)\phi\left(x_{t-1}, v_{t-1}, x_t, v_t\right)\lambda\left(x_t\right) \tag{22.1.6}$$

Similarly, we can integrate equation (24.2.2) over x_t to get the marginal at time x_{t-1} which by consistent should be proportional to $\rho\left(x_{t-1}\right)\lambda\left(x_{t-1}\right)$. From such considerations we arrive at

$$\rho\left(x_t\right) \propto \frac{\sum_{x_{t-1}} \rho\left(x_{t-1}\right)\phi\left(x_{t-1}, x_t\right)\lambda\left(x_t\right)}{\lambda\left(x_t\right)}, \tag{22.1.7}$$

$$\lambda\left(x_{t-1}\right) \propto \frac{\sum_{x_t} \rho\left(x_{t-1}\right)\phi\left(x_{t-1}, x_t\right)\lambda\left(x_t\right)}{\rho\left(x_{t-1}\right)} \tag{22.1.8}$$

where the divisions can be interpreted as preventing overcounting of messages. The common factors in the numerator and denominator exactly cancel to give

$$\text{Forward Recursion: } \rho\left(x_t\right) \propto \sum_{x_{t-1}} \rho\left(x_{t-1}\right)\phi\left(x_{t-1}, v_{t-1}, x_t, v_t\right) \tag{22.1.9}$$

$$\text{Backward Recursion: } \lambda\left(x_{t-1}\right) \propto \sum_{x_t} \phi\left(x_{t-1}, v_{t-1}, x_t, v_t\right)\lambda\left(x_t\right) \tag{22.1.10}$$

which are the usual definitions of the messages defined as a set of independent recursions. In engineering, the ρ message is called the α message, and the λ message is called the β message. This method of performing inference is called a *parallel* method since the α and β recursions are independent of each other and can therefore be implemented in parallel. After computation, they may then be combined to compute the smoother posterior.

The extension to more general singly connected structures is straightforward and results in partially independent recursions which communicate only at branches of the tree [10].

From equation (24.2.1) it is straightforward to see that $\lambda\left(x_t\right) \propto p(v_{t+1:T}|x_t, v_{1:t}) = p(v_{t+1:T}|x_t)$. By definition $\rho(x_t) \propto p(x_t|v_{1:t})$ is the filtered estimate.

Logs or Normalise?

The repeated application of the recursions equation (22.1.9) and equation (22.1.9) may lead to numerical under/over flow. There are two strategies for dealing with this. One is to work in log space, so that only the log of the messages are defined. The other (which is more common in the machine learning literature) is to normalise the messages ρ and λ at each stage of the iteration, so that the messages sum to unity. Normalisation is valid since both the filtered $p(x_t|v_{1:T}) \propto \rho\left(x_t\right)$ and smoothed inferences $p(x_t|v_{1:T}) \propto \rho\left(x_t\right)\lambda\left(x_t\right)$ are simply proportional to the messages. The missing proportionality constants can be worked out easily since we know that distributions must sum to one.

22.1.2 Rauch Tung Striebel and the $\alpha - \gamma$ recursions

In the above, we found a forward (α or ρ) recursion for the filtered inference $p(h_t|v_{1:t})$. Explicitly, for the HMM, the forward pass is

$$
\begin{aligned}
p(h_t|v_{1:t}) &\propto \sum_{h_{t-1}} p(h_t, h_{t-1}, v_{1:t-1}, v_t) && (22.1.11)\\
&= \sum_{h_{t-1}} p(v_t|v_{1:t-1}, h_{1:t}, h_t, h_{t-1}) p(h_t|v_{1:t-1}, h_{t-1}) p(v_{1:t-1}, h_{t-1}) && (22.1.12)\\
&\propto \sum_{h_{t-1}} p(v_t|h_t) p(h_t|h_{t-1}) p(h_{t-1}|v_{1:t-1}) && (22.1.13)
\end{aligned}
$$

Here we derive an alternative way to compute the smoothed inference $p(h_t|v_{1:T})$ by correcting these filtered results. We start with the recursion

$$
\begin{aligned}
\gamma(h_t) \equiv p(h_t|v_{1:T}) &= \sum_{h_{t+1}} p(h_t, h_{t+1}|v_{1:T}) && (22.1.14)\\
&= \sum_{h_{t+1}} p(h_t|h_{t+1}, v_{1:t}) p(h_{t+1}|v_{1:T}) && (22.1.15)
\end{aligned}
$$

Hence, we can form a backwards recursion for the smoothed inference. $p(h_t, h_{t+1}|v_{1:T})$ is given by the above before summing over h_{t+1}. We therefore need

$$p(h_t|h_{t+1}, v_{1:t}) \propto p(h_{t+1}, h_t|v_{1:t}) \propto p(h_{t+1}|h_t) p(h_t|v_{1:t})$$

The denominator is just found by normalisation. In the above, we see that the smoothed recursion makes explicit use of the filtered results. In contrast to the $\alpha{-}\beta$ independent recursion, the above procedure is called a sequential procedure since we need to first complete the α recursions, after which the γ recursion may begin. Formally, the $\alpha - \beta$ and $\alpha - \gamma$ recursions are related through $\gamma(h_t) \propto \alpha(h_t)\beta(h_t)$.

The Likelihood $p(v_{1:T})$

The likelihood is found from recursing

$$p(v_{1:t}) = \prod_t p(v_t|v_{1:t-1})$$

Each factor

$$
\begin{aligned}
p(v_t|v_{1:t-1}) &= \sum_{h_t} p(v_t, h_t|v_{1:t-1}) && (22.1.16)\\
&= \sum_{h_t} p(v_t|h_t) p(h_t|v_{1:t-1}) && (22.1.17)\\
&= \sum_{h_t} p(v_t|h_t) \sum_{h_{t-1}} p(h_t|h_{t-1}) p(h_{t-1}|v_{1:t-1}) && (22.1.18)
\end{aligned}
$$

where the final term $p(h_{t-1}|v_{1:t-1})$ are just the filtered inferences. Note, therefore, that the likelihood of a output sequence requires only a forward computation.

22.1.3 Viterbi

Consider the general HMM problem: Find the most likely state of

$$p(h_{1:T}|y_{1:T})$$

This is the same as the most likely state of

$$p(h_{1:T}, y_{1:T}) = \prod_t p(y_t|h_t)p(h_t|h_{t-1})$$

This is easy to find, by using the *max* version of the JTA/BP algorithms. To make this explicit though, we write down exactly how this would proceed:

To make the notation a little easier, let's define the potential functions :

$$\phi(h_{t-1}, h_t) = p(y_t|h_t)p(h_t|h_{t-1})$$

where for the first time step we just define $\phi(h_1) = p(y_1|h_1)p(h_1)$. The finding the most likely hidden state sequence is equivalent to finding the state $h_{1:T}$ that maximises the funcion

$$\phi = \phi(h_1)\prod_{t=2}^{T}\phi(h_{t-1}, h_t)$$

The dependency on h_1 appears only in the first two terms $\phi(h_1)$ and $\phi(h_1, h_2)$. Hence when we perform the max over h_1, we can write

$$\max_{h_{1:T}} \phi = \max_{h_{2:T}} \underbrace{\max_{h_1} \phi(h_1)\phi(h_1, h_2)}_{f(h_2)} \prod_{t=3}^{T}\phi(h_{t-1}, h_t)$$

At the next stage, we can perform the max over h_2 :

$$\max_{h_{1:T}} \phi = \max_{h_{3:T}} \underbrace{\max_{h_2} f(h_2)\phi(h_2, h_3)}_{f(h_3)} \prod_{t=4}^{T}\phi(h_{t-1}, h_t)$$

We can continue this procedure, at each stage defining the new potentials

$$f(h_t) = \max_{h_{t-1}} f(h_{t-1})\phi(h_{t-1}, h_t)$$

until we reach the end of the chain, and we have defined $f(h_2), \ldots, f(h_T)$. Then, to find which states actually correspond to the maxima, we need to backtrack: We have at the end of the chain $f(h_T)$. Hence, the most likely state is given by

$$h_T^* = \arg\max_{h_T} f(h_T)$$

With this most likely state, we can write

$$\max_{h_{1:T-1}} \phi = f(h_{T-1})\phi(h_{T-1}, h_T^*)$$

So that we can find the optimal state h_{T-1} by computing

$$h_{T-1}^* = \arg\max_{h_{T-1}} f(h_{T-1})\phi(h_{T-1}, h_T^*)$$

and similarly,

$$h^*_{t-1} = \arg\max_{h_{t-1}} f(h_{t-1})\phi(h_{t-1}, h^*_t)$$

for $t = T-1, \ldots, 1$, where we define $f(h_1) \equiv p(h_1)$.

For the HMM this special case of the max-product algorithm is called the Viterbi algorithm, a terminology from speech research.

Second Order HMM

We can look at more complex time dependencies in the hidden variables by increasing the range of the temporal dependencies. For example, a second order HMM is given in fig(22.5). The inference can again be carried out using the JTA,

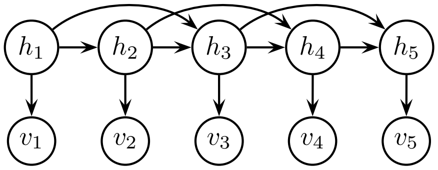


Figure 22.5: A 2nd order Hidden Markov Model.

and a suitable JT is given in fig(22.6).

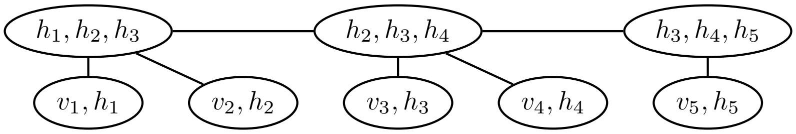


Figure 22.6: A 2nd order Hidden Markov Model. A suitable assignment of the potentials is: $\Psi(h_1, h_2, h_3) = p(h_1)p(h_2|h_1)p(h_3|h_1, h_2)$, $\Psi(h_2, h_3, h_4) = p(h_4|h_2, h_3)$,$\Psi(h_3, h_4, h_5) = p(h_5|h_3, h_4)$, $\Psi(v_1, h_1) = p(v_1|h_1)$, $\Psi(v_2, h_2) = p(v_2|h_2)$, $\Psi(v_3, h_3) = p(v_3|h_3)$,$\Psi(v_4, h_4) = p(v_4|h_4)$,$\Psi(v_5, h_5) = p(v_5|h_5)$. Again, separator potentials are not shown. They may all be set initially to unity.

Now the complexity is still linear in time, but $O(TH(V + H^2))$. In general, the complexity will be exponential in the order of the interactions.

Learning HMMs

Previously, we concentrated on inference in HMMs. We also have a general framework for learning in graphical models. Here, as an example application of our previous general framework, we show how it applies to HMMs. Historically, the procedure of using the EM algorithm was called the Baum-Welch algorithm. Personally, I think that you should really remember the general approach, and be delighted to see that it produces BW algorithm as a special case.

Baum-Welch Algorithm

A HMM is trained by treating the output nodes as evidence nodes and the state nodes as hidden nodes. This is clearly tractable since the moralization and triangulation steps do not add any extra links. The cliques are of size N^2 where N is the dimension of the state nodes. Inference therefore scales as $O(N^2T)$ where T is the length of the times series.

To find the parameters of the model, A, B, π, a variational type (EM) procedure can be used, which can be constructed using our previous EM framework.

To make the notation reasonably simple, we write $v = (v_1, v_2, \ldots, v_T)$, and similarly, $h = (h_1, h_2, \ldots, h_T)$.

Let's look at the energy function :

$$\sum_\mu \sum_t \langle \log p(v_1^\mu, v_2^\mu, \ldots, v_T^\mu, h_1^\mu, h_2^\mu, \ldots h_T^\mu \rangle_{q^\mu(h|v)}$$

Using the form of the HMM, we obtain

$$\sum_\mu \left\{ \langle \log p(h_1) \rangle_{q^\mu(h_1|v)} + \sum_{t=1}^{T-1} \langle \log p(h_{t+1}|h_t) \rangle_{q^\mu(h_t,h_{t+1}|v^\mu)} + \sum_{t=1}^{T} \langle \log p(v_t^\mu|h_t) \rangle_{q^\mu(h_t|v^\mu)} \right\}$$

To avoid potential confusion, we write $p^{new}(h_1 = i)$ to denote the (new) table entry for probability that the intial hidden variable is in state i. The prior term, by the previously derived EM approach then gives

$$\pi_i^{new} \equiv p^{new}(h_1 = i) \propto \sum_\mu p^{old}(h_1 = i|v^\mu) \tag{22.1.19}$$

which is the average number of times that the first hidden variable is in state i. Similarly,

$$A_{i',i}^{new} \equiv p^{new}(h_{t+1} = i'|h_t = i) \propto \sum_\mu \sum_{t=1}^{T-1} p^{old}(h_t = i, h_{t+1} = i'|v^\mu) \tag{22.1.20}$$

which is the number of times that a transition from hidden state i to hidden state i' occurs, averaged over all times (since we assumed stationarity) and training sequences. Finally,

$$B_{j,i}^{new} \equiv p^{new}(v_t = j|h_t = i) \propto \sum_\mu \sum_{t=1}^{T} I[v_t^\mu = j] p^{old}(h_t = i|v^\mu) \tag{22.1.21}$$

which is the expected number of times that, for the observation being in state j, we are in hidden state i. The proportionalities are trivially determined by the normalisation constraint. Together, the above three equations define the new prior, transition and emission probabilities. Using these values for the HMM CPTs, at the next step we can calculate the quantities $p^{old}(h_1 = i|v^\mu)$, $p^{old}(h_t = i, h_{t+1} = i'|v^\mu)$ and $p^{old}(h_t = i|v^\mu)$ using the JTA (or the so-called 'Forward-Backward' algorithm, which is equivalent). The equations (22.1.19,22.1.20,22.1.21) are repeated until convergence.

Parameter Initialisation

The above EM approach is guaranteed to converge to a local maxima of the likelihood (one can show explicitly that the re-estimation formulae correspond to fixed point equations representing the point where the gradient of the likelihood is zero).

(Of course, if we were to use a restricted class of q^μ functions, we would only converge to a local maximum of the lower bound on the likelihood). There is no guarantee that the algorithm will find the global maximum, and indeed, the value of the local maximum found is often critically dependent on the initial settings of the parameters. How best to initialise the parameters is a thorny issue. According to Rabiner :

"Experience has shown that either random (subject to the stochastic and the nonzero value constraints) or uniform initial estimates of the π and A parameters is adequate for giving useful re-estimates of these parameters in almost all cases. However, for the B parameters, experience has shown that good initial estimates are helpful in the discrete case, and are essential in the continuous distribution case (see later). Such initial estimates can be obtained in a number of ways, including manual segmentation of the observation sequence(s) into states with averaging of observations within states, maximum likelihood segmentation of observations with averaging, and segmental k-means segmentation with clustering."

Continuous Output, Discrete Hiddens

In many cases, the observation sequence is continuous. In fig(22.8) we saw an approach using vector quantisation to transform continuous outputs into discrete variables. Here we will consider a different approach which retains continuous outputs. In later sections we will also deal with the case of continuous hidden variables. However, for the moment, it is simpler to deal with discrete hidden dynamics.

What is required is a specification

$$p(v|h)$$

where v is continuous vector variable and h is discrete. Using a continuous output will not significantly affect the previous update equations provided we assume that all the observations are indeed visible. In this case, the contributions to the energy are as before, but simply the factors $p(v(t)|h(t))$ will have a numerical value determined by the new density.

A natural candidate for the above are Mixture Models

$$p(v|h) = \sum_k p(k|h)p(v|k,h)$$

where k is a discrete summation variable. And a common choice is that each distribution $p(v|k,h)$ is a Gaussian

$$p(v|k,h) = N(\boldsymbol{\mu}_{k,h}, \boldsymbol{\Sigma}_{k,h})$$

This Gaussian mixture model therefore requires the learning of KH V-dimensional mean vectors (where V is the dimension of the output, K is the number of mixture components and H is the number of hidden states), and also KH covariance matrices (having $V(V+1)/2$ parameters each). The update equations in learning for π and A remain the same. However, the emission probabilities now require us to find those optimal mean, covariances and $p(k|h)$ parameters that maximise

$$\sum_\mu \sum_{t=1}^{T} \langle \log p(v_t|h_t) \rangle_{q^\mu(h_t|V^\mu)}$$

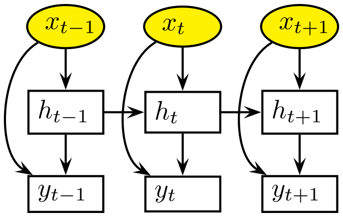


Figure 22.7: Graphical model of the IOHMM. Nodes represent the random variables and arrows indicate direct dependence between variables. In our case the output variable y_t is discrete and represents the class label, while the input variable x_t is the continuous (feature extracted from the) EEG observation. The yellow (shaded) nodes indicate that these variables are given, so that no associated distributions need be defined for $x_{1:T}$.

where V^μ is the total sequence of observation vectors. We may consider the k as hidden variables, and then use EM algorithm as before to obtain update equations for the parameters of the Gaussian Mixture Model (or indeed the parameters of any Mixture Model). We leave it to the reader to complete this straightforward derivation. GMMs for the emissions in this way are used in state of the art speech recognition software.

Related Models

Input-Output HMM

The IOHMM is just a HMM augmented with outputs (visible variables) $y_{1:T}$ and hidden states $h_{1:T}$. However, we now consider that we are given for each time step an input x_t. This input can be continuous or discrete and affects the transitions as

$$p(y_{1:T}, h_{1:T}|x_{1:T}) = \prod_t p(y_t|h_t, x_t)p(h_t|h_{t-1}, x_t)$$

This is just another HMM, and extending inference and learning to this case is straightforward. IOHMM is usually used as a conditional classifier, where the outputs y_t represent a class label at time t. (There are other ways to train this model, say by specifying a label only at the end of the sequence). In the case of continuous inputs, the tables $p(y_t|h_t, x_t)$ and $p(h_t|h_{t-1}, x_t)$ are usually parameterised using a non-linear function, eg.

$$p(y_t = y|h_t = h, x_t = x) \propto e^{w_{h,y}^{\mathsf{T}} x}$$

Inference then follows the same line as for the standard HMM:

$$p(h_t|x, y) = \sum_{h_{t+1}} p(h_t, h_{t+1}|x, y) \tag{22.1.22}$$

$$= \sum_{h_{t+1}} p(h_t|h_{t+1}, x_{1:t+1}, y_{1:t})p(h_{t+1}|x, y) \tag{22.1.23}$$

$$\tag{22.1.24}$$

Hence, we can form a backwards recursion. $p(h_t, h_{t+1}|x_{1:T}, y_{1:T})$ is given by the above before summing over h_{t+1}.

We therefore need

$$p(h_t|h_{t+1}, x_{1:t}, y_{1:t}) = \frac{p(h_{t+1}, h_t|x_{1:t+1}, y_{1:t})}{p(h_{t+1}|x_{1:t+1}, y_{1:t})} = \frac{p(h_{t+1}|h_t, x_{t+1})p(h_t|x_{1:t}, y_{1:t})}{p(h_{t+1}|x_{1:t+1}, y_{1:t})}$$

The denominator is just found by normalisation. To find the rest, we use a forward pass

$$p(h_t|x_{1:t}, y_{1:t}) \propto \sum_{h_{t-1}} p(h_t, h_{t-1}, x_{1:t}, y_{1:t-1}, y_t) \quad (22.1.25)$$

$$= \sum_{h_{t-1}} p(y_t|y_{1:t-1}, x_{1:t}, h_t, h_{t-1})p(h_t|y_{1:t-1}, x_{1:t}, h_{t-1})p(y_{1:t-1}, x_{1:t}, h_{t-1}) \quad (22.1.26)$$

$$\propto \sum_{h_{t-1}} p(y_t|x_t, h_t)p(h_t|x_t, h_{t-1})p(h_{t-1}|x_{1:t-1}, y_{1:t-1}) \quad (22.1.27)$$

The likelihood is found from recusing $p(y|x) = \prod p(y_t|y_{1:t-1}, x)$

$$p(y_t|y_{1:t-1}, x) = p(y_t|y_{1:t-1}, x_{1:t})$$

Direction Bias

The IOHMM and related conditionally trained models 'suffer' from the fact that any prediction $p(v_t|h_{1:T})$ in fact depends only on the past $p(v_t|h_{1:t})$. This is not true, of course, of the most likely output sequence. Such 'direction bias' is identified in some sections of the literature (particularly in natural language modelling) as problematic, and motivates the use of undirected models, such as the Conditional Random Field.

22.2 Applications of HMMs

Speech Processing

This is arguably one of the most successful application areas of HMMs.

This section is based on Mike Alder's excellent book on An Introduction to Pattern Recognition, `http://heavenforbooks.com`.

While by no means meant to be an attempt to explain the state of the art in speech recognition, we here give a flavour of one of the common approached to solving this problem. For simplicity, we here think of the case of recognizing only single words.

If you say a word, perhaps the name of a digit, for example 'one', into a microphone, then it is straightforward to sample and digitise the resulting signal, and feed it into a computer as a longish sequence of numbers measuring the voltage generated by the microphone and your voice. Typically, a word may take one third to half a second to enunciate, and the signal is sampled perhaps twenty thousand times a second, giving around seven thousand numbers. Each number will be quantised to perhaps 12 or 16 bit precision. Thus we may be looking at a data rate of around 30 to 40 kilobytes per second. This present paragraph, would, if spoken

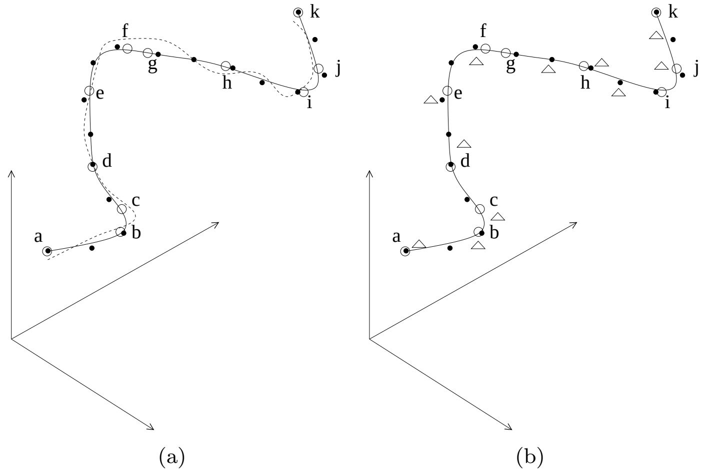


Figure 22.8: A simple way to transform continuous signals into discrete signals is to use vector quantisation. (a) After preprocessing, a section of speech is represented by a trajectory through a high dimensional space (here depicted as three dimensions). For example, we represent one trajectory by the dotted line. Many different utterances of the same word will hopefully produce similar trajectories to the mean trajectory (here shown as the solid curve). Codebook vectors are represented by circles. Points in this space on the mean trajectory that are equally separated in time are represented by a small dot. (b) A novel trajectory (the triangles) is compared to the codebook vectors so that it can be transformed into a string, here **abcdefhhhjk**. Note, however, that this string does not take the time aspect into account. To map this into a string which represents the state of the system at equally spaced times, this would be **aabcddeffhhhjjkk**.

at a reasonable reading rate, occupy over two megabytes of disk space. If printed, it would occupy around a kilobyte. There is therefore a considerable amount of compression involved in Automatic Speech Recognition (ASR).

There are various methods of proceeding from this point, but the most fundamental and conceptually simplest is to take a Discrete Fourier Transform (DFT) of a short chunk of the signal, referred to as the part of the signal inside a window. Imagine that we have a sound and something like a harp, the strings of which can resonate to particular frequencies. For any sound whatever, each string of the harp will resonate to some extent, as it absorbs energy at the resonant frequency from the input sound. So we can represent the input sound by giving the amount of energy in each frequency which the harp extracts, the so-called energy spectrum.

We take, then, some time interval, compute the fast Fourier transform (FFT) and then obtain the power spectrum of the wave form of the speech signal in that time interval of, perhaps, 32 msec. Then we slide the time interval, the window, down the signal, leaving some overlap in general, and repeat. We do this for the entire length of the signal, thus getting a sequence of perhaps ninety vectors, each vector in dimension perhaps 256, each of the 256 components being an estimate of the energy in some frequency interval between, say, 80 Hertz and ten KHz.

Practical problems arise from trying to sample a signal having one frequency with a sampling rate at another; this is called ‘aliasing’ in the trade, and is most commonly detected when the waggon wheels on the Deadwood Stage go backwards, or a news program cameraman points his camera at somebody's computer terminal and gets that infuriating black band drifting across the screen and the flickering that makes the thing unwatchable. There is a risk that high frequencies in the speech signal will be sampled at a lower frequency and will manifest themselves as a sort of flicker. So it is usual to kill off all frequencies not being explicitly looked for, by passing the signal through a filter which will not pass very high or very low frequencies. Very high usually means more than half the sampling frequency, and very low means little more than the mains frequency.

The 256 numbers may usefully be 'binned' into some smaller number of frequency bands, perhaps sixteen of them, also covering the acoustic frequency range.

This approach turns the utterance into a longish sequence of vectors in representing the time development of the utterance, or more productively as a trajectory. Many repetitions of the same word by the same speaker might reasonably be expected to be described as trajectories which are fairly close together. If we have a family of trajectories corresponding to one person saying 'yes' and another family corresponding to the same person saying 'no', then if we have an utterance of one of those words by the same speaker and wish to know which it is, then some comparison between the new trajectory and the two families we already have, should allow us to make some sort of decision as to which of the two words we think most likely to have been uttered.

Put in this form, we have opened up a variant of traditional pattern recognition which consists of distinguishing not between different categories of point in a space, but different categories of trajectory in the space. Everything has become time dependent; we deal with changing states.

An example of such a procedure is given in fig(22.8). There we see that a particular speech signal has been transformed into a string of states. What we would like to do is then, given a set of different utterances of the same word, with their corresponding strings, to learn a representation for the transitions between the states of these strings. This is where the HMM comes in. For each state (one of the symbols in the string) there is a probability (that we need to learn) of either staying in the same state (holding probability) or switching to one of the other states. We can use standard HMM algorithms to learn such transitions.

Given then two models, say one of "yes" and the other of "no", how do we use these models to classify a novel utterance? The way that we do this is to find for which of the two models was the sequence more likely. For example, imagine that we have an utterance, "yeh". Then we wish to find under which model is this utterance the most likely. That is, we compare $p(``yeh''|model``yes'')$ with $p(``yeh''|model`'no'')$. To calculate these likelihoods, we can use the standard marginalisation techniques for graphical models.

A book by one of the leading speech recognition experts is available online at `http://labrosa.ee.columbia.edu/doc/HTKBook21/HTKBook.html`.

BioInformatics

Biological sequences are often successfully modelled by HMMs, and have many interesting and powerful applications in BioInformatics, for example for multiple sequence alignment.

The technical report *Hidden Markov Models in Computational Biology : Applications to Protein Modelling UCSC-CRL-93-92* by Krogh *et. al.* is a nice introduction to this area. The following book is also very good, *Biological Sequence Analysis : Probabilistic Models of Proteins and Nucleic Acids* (Cambridge University Press, 1999) by Richard Durbin, Sean R. Eddy, Anders Krogh, Graeme Mitchison.

Linguistic Sequences e.g. part-of-speech tagging.

See `http://www.comp.lancs.ac.uk/ucrel/annotation.html#POS`
for a statement of the problem and some probabilistic solutions. For example, we have a sentence, as below, in which each word has been linguistically tagged (eg NN is the singular common noun tag, ATI is the article tag etc.).

```
hospitality_NN is_BEZ an_AT excellent_JJ virtue_NN ,_,
but_CC not_XNOT when_WRB the_ATI guests_NNS have_HV
to_TO sleep_VB in_IN rows_NNS in_IN the_ATI cellar_NN !_!
```

One can attempt a solution to these tagging problems by using a HMM to model the way that tag to tag transitions tend to occur from a corpus of tagged word sequences. This forms the hidden space dynamics. An emission probability to go from a tag to an observed word is also used, so that then for a novel sequence of words, the most likely tag (hidden) sequence can be inferred.

Multi-electrode spike-train analysis

There are many applications of HMMs to NeuroInformatics. One area assumes that there is a common underlying mechanism generating the observed data. This might be say an epileptic event (the hidden variables) which is recorded on the surface of the scalp by multiple electrodes (the observations). Based on the physics of how signals get dispersed by the skull (the emission probabilty), and some smoothness constraints on the underlying hidden transitions, one can make reasonable models of neurologically significant events such that a future epileptic event can be detected automatically.

Tracking objects through time

One of the original, and still very common applications of HMMs is in tracking. They have been particularly successful in tracking moving objects, whereby an understanding of newtonian dynamics in the hidden space, coupled with an understanding of how an object with a known position and momentum would appear on the screen/radar image, enables one to infer the position and momentum of an object based only on radar. This has obvious military applications and is one of the reasons that some of the algorithms associated with HMMs and related models were classified until recently (although doing the inference was probably well understood anyway!).

22.3 Problems

Exercise 85 *Consider a HMM with 3 states ($M = 3$) and 2 output symbols, with a left-to-right state transition matrix*

$$A = \begin{pmatrix} 0.5 & 0.0 & 0.0 \\ 0.3 & 0.6 & 0.0 \\ 0.2 & 0.4 & 1.0 \end{pmatrix}$$

where $A_{ij} \equiv p(h(t+1) = i|h(t) = j)$, an output probabilities matrix $B_{ij} \equiv p(v(t) = i|h(t) = j)$

$$B = \begin{pmatrix} 0.7 & 0.4 & 0.8 \\ 0.3 & 0.6 & 0.2 \end{pmatrix}$$

and an initial state probabilities vector $\pi = (0.9\ 0.1\ 0.0)^T$. Given that the observed symbol sequence is 011, *compute*

(i) $P(v_{1:T})$

(ii) $P(h_1|v_{1:T})$. *[As there are 3 observations the HMM will have three time slices—you are asked to compute the posterior distribution of the state variable in the second time slice, numbering the times 0, 1, 2.] You can check this calculation by setting up the HMM in JavaBayes.*

(iii) Find the best hidden state sequence given a sequence of observations, and apply it to the model (Viterbi algorithm)

Exercise 86 *Suppose the matrix A above had its columns all equal to the initial probabilities vector π. In this case the HMM reduces to a simpler model—what is it?*

Exercise 87 *Show that if a transition probability a_{ij} in a HMM is set to zero initially, then it will remain at zero throughout training.*

Exercise 88 *Consider the problem : Find the most likely joint output sequence $v_{1:T}$ for a HMM. That is,*

$$\arg\max_{v_{1:T}} p(v_{1:T})$$

where

$$p(h_{1:T}, v_{1:T}) = \prod_t p(v_t|h_t)p(h_t|h_{t-1})$$

(i) Explain how the above problem can be formulated as a mixed maxproduct/sumproduct criterion.

(ii) Explain why a local message passing algorithm cannot, in general, be found for this problem to guarantee to find the optimal solution.

(iii) Explain how to adapt the Expectation-Maximisation algorithm to form a recursive algorithm, with local message passing, to guarantee at each stage of the EM algorithm an improved joint output state.

Exercise 89 *Explain how to train a HMM using EM, but with a constrained transition matrix. In particular, explain how to learn a transition matrix with a triangular structure.*

22.4 Solutions

23 Dynamic Continuous Hiddens : Linear Dynamical Systems

Linear Gaussian State Space Models

Consider a dynamical system

$$h_{t+1} = Ah_t$$

where h is a vector, and A is a transition matrix. For example, if h is a two dimensional vector, and A is a rotation matrix through θ degrees, then h will trace out points on a circle through time. If we were to define a related variable, say

$$v(t) = [h_t]_1$$

namely the projection of the hidden variable dynamics, then v would describe a sinusoid through time. More generally, we could consider a model

$$v(t) = Bh(t)$$

which linearly related the visible variable $v(t)$ to the hidden dynamics at time t. This is therefore a linear dynamical system.

A drawback to the above models is that they are all deterministic. To account for possible stochastic behaviour, we generalise the above to

$$h_t = Ah_{t-1} + \eta_t^h$$

$$v_t = Bh_t + \eta_t^v$$

where η_t^h and η_t^v are noise vectors. As a graphical model, we write

$$p(h_{1:T}, v_{1:T}) = p(h_1)p(v_1|h_1)\prod_{t=2}^{T} p(h_t|h_{t-1})p(v_t|h_t)$$

It is computationally particularly convenient to consider Gaussian noise models, which defines a Linear Dynamical System, also known as a Linear Gaussian State Space model (LGSSM). In Engineering, these models are also called Kalman Filters. I prefer not to use this terminology here, since the word 'filter' refers to a specific kind of inference within the LGSSM. That is, we will reserve the word Kalman Filter for filtered inference, as we will see later.

A LGSSM is the Gaussian equivalent of a HMM. In this case, the hidden transition and emission probabilities are assumed Gaussian. Now each hidden variable is a multidimensional Gaussian distributed vector h_t, as is the vector output v_t.

$$p(h_t|h_{t-1}) = \frac{1}{\sqrt{|2\pi\Sigma_H|}} \exp\left(-\frac{1}{2}\left(h_t - Ah_{t-1}\right)^{\mathsf{T}} \Sigma_H^{-1}\left(h_t - Ah_{t-1}\right)\right)$$

which states that h_{t+1} has a mean equal to Ah_t and has Gaussian fluctuations described by the covariance matrix Σ_H.

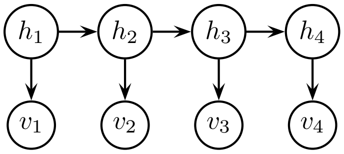


Figure 23.1: A LGSSM. Both hidden and visible variables are Gaussian Distributed.

Similarly,

$$p(v_t|h_t) = \frac{1}{\sqrt{|2\pi\Sigma_V|}} \exp\left(-\frac{1}{2}\left(v_t - Bh_t\right)^{\mathsf{T}} \Sigma_V^{-1} \left(v_t - Bh_t\right)\right)$$

describes an output v_t with mean Bh_t and covariance Σ_V. At time step 1,

$$p(h_1) \sim \mathcal{N}(\mu, \Sigma)$$

The above defines a stationary LGSSM since the parameters of the model are fixed through time. The non-stationary case allows for different parameters at each time step, for example $\Sigma_V(t)$. The above definitions are for the *first order* LGSSM, since the hidden state depends on only the first previous hidden state – the extension to higher order variants, where the hidden state depends on several past hidden states,is straightforward. We could also consider having an external known input at each time, which will change the mean of the hidden variable. The generalisation to this case is straightforward, and left as an exercise for the interested reader.

LGSSMs are extremely popular models in temporal sequence analysis. In particular, such models have a very natural application to tracking.

23.1 Inference

Consider an observation sequence $v_{1:T}$. How can we infer the marginals of the hiddens $p(h_t|v_{1:T})$?

We cannot in this case directly use the JTA, since we cannot in general pass the table entries for a continuous distribution – there are effectively an infinite number of table entries. However, since Gaussians are fully specified by a small set of parameters (the sufficient statistics), namely their mean and covariance matrix, we can alternatively pass parameters during the absorption procedure to ensure consistency. The reader is invited to carry out this procedure (or the alternative Belief Propagation method). Whilst this scheme is valid, the resulting recursions are numerically complex, and may be unstable. One approach to avoid this is to make use of the Matrix Inversion Lemma to reduce the recursions to avoid unnecessary matrix inversions. An alternative is to use the RTS-smoothing style scheme that we introduced in the HMM chapter.

23.1.1 The Forward Pass : The Kalman Filter

The forward pass is a Gaussian with mean f_t and covariance F_t,

$$p(h_t|v_{1:t}) \sim \mathcal{N}(f_t, F_t)$$

We can find the joint distribution $p(h_t, v_t|v_{1:t-1})$, and then condition on v_t to easily find the distribution $p(h_t|v_{1:t})$. The term $p(h_t, v_t|v_{1:t-1})$ is a Gaussian and can be found easily using the relations

$$v_t = Bh_t + \eta^v, \qquad h_t = Ah_{t-1} + \eta^h$$

Using the above, we readily find

$$\left\langle \Delta v_t \Delta v_t^\mathsf{T} | v_{1:t-1} \right\rangle = B \left\langle \Delta h_t \Delta h_t^\mathsf{T} | v_{1:t-1} \right\rangle B^\mathsf{T} + \Sigma^v$$

$$\left\langle \Delta h_t \Delta h_t^\mathsf{T} | v_{1:t-1} \right\rangle = A \left\langle \Delta h_{t-1} \Delta h_{t-1}^\mathsf{T} | v_{1:t-1} \right\rangle A^\mathsf{T}(s_t) + \Sigma^h$$

$$\left\langle \Delta v_t \Delta h_t^\mathsf{T} | v_{1:t-1} \right\rangle = B(s_t) \left\langle \Delta h_t \Delta h_t^\mathsf{T} | v_{1:t-1} \right\rangle$$

$$\langle v_t | v_{1:t-1} \rangle = BA \langle h_{t-1} | v_{1:t-1} \rangle, \qquad \langle h_t | v_{1:t-1} \rangle = A \langle h_{t-1} | v_{1:t-1} \rangle$$

In the above, using our moment representation of the forward messages

$$\langle h_{t-1} | v_{1:t-1} \rangle \equiv f_{t-1}, \qquad \left\langle \Delta h_{t-1} \Delta h_{t-1}^\mathsf{T} | v_{1:t-1} \right\rangle \equiv F_{t-1}$$

Then, using conditioning[1] $p(h_t|v_t, v_{1:t-1})$ will have mean

$$f_t \equiv \langle h_t | v_{1:t-1} \rangle + \left\langle \Delta h_t \Delta v_t^\mathsf{T} | v_{1:t-1} \right\rangle \left\langle \Delta v_t \Delta v_t^\mathsf{T} | v_{1:t-1} \right\rangle^{-1} (v_t - \langle v_t | v_{1:t-1} \rangle)$$

and covariance

$$F_t \equiv \left\langle \Delta h_t \Delta h_t^\mathsf{T} | v_{1:t-1} \right\rangle - \left\langle \Delta h_t \Delta v_t^\mathsf{T} | v_{1:t-1} \right\rangle \left\langle \Delta v_t \Delta v_t^\mathsf{T} | v_{1:t-1} \right\rangle^{-1} \left\langle \Delta v_t \Delta h_t^\mathsf{T} | v_{1:t-1} \right\rangle$$

A nice thing about the above approach is that we work always in the moment representation, and the iteration is expected to be numerically stable when the noise covariances are small. This procedure is called the Forward Pass in the LGSSM inference algorithm (albeit with a change to the standard notation in the literature for representing the filtered posterior).

Numerical Stability

In the standard notation, we can write the forward update using

$$\hat{P} = (I - KB)\, P, K = PB^\mathsf{T} \left(\Sigma_V + BPB^\mathsf{T} \right)^{-1}$$

A problem with this update formula for the covariance is that it is the difference of two positive definite matrices. Numerically, this may cause some problems since the resulting expression is not guaranteed to be positive definite.

An alternative is to use the definition of K, from which we can write

$$K \Sigma_V K^\mathsf{T} = (I - KB)\, PB^\mathsf{T} K^\mathsf{T}$$

hence

$$(I - KB)\, P\, (I - KB)^\mathsf{T} + K \Sigma_V K^\mathsf{T} \equiv (I - KB) \left(P\, (I - KB)^\mathsf{T} + PB^\mathsf{T} K^\mathsf{T} \right) \equiv (I - KB)\, P$$

The LHS is the addition of two positive definite matrices, so that numerically, the resulting update for the covariance is more stable. A similar trick can be used in the backward pass below.

[1] $p(x|y)$ is a Gaussian with mean $\mu_x + \Sigma_{xy} \Sigma_{yy}^{-1} (y - \mu_y)$ and covariance $\Sigma_{xx} - \Sigma_{xy} \Sigma_{yy}^{-1} \Sigma_{yx}$.

23.1.2 The Kalman Smoother : The Rauch-Tung-Striebel Smoother

In principle, we can apply the Belief Propagation method to form a backpass to find $p(h_t|v_{1:T})$ (see barberieee). However,we would like to avoid defining λ messages here since it is awkward to extend BP to the SKF case. Here we show how for the simple case of the Kalman Filter, how a smoothing backpass can be formed without defining λ messages. Instead we form directly a recursion for the smoothed distribution $p(h_t|v_{1:T})$.

Imagine that we have completed a forward pass, so that we have, for the KF, the filtered distributions $p(h_t|v_{1:t})$. We can form a recursion for the smoothed posteriors $p(h_t|v_{1:T})$, directly without using λ recursions as follows:

$$p(h_t|v_{1:T}) \propto \sum_{h_{t+1}} p(h_t|v_{1:T}, h_{t+1}) p(h_{t+1}|v_{1:T}) \tag{23.1.1}$$

$$\propto \sum_{h_{t+1}} p(h_t|v_{1:t}, h_{t+1}) p(h_{t+1}|v_{1:T}) \tag{23.1.2}$$

The term $p(h_t|v_{1:t}, h_{t+1})$ can be found by conditioning the joint distribution $p(h_t, h_{t+1}|v_{1:t}) = p(h_{t+1}|h_t)p(h_t|v_{1:t})$. We can work out this joint distribution in the usual manner by finding its mean and covariance. The term $p(h_t|v_{1:t})$ is a known Gaussian from the Forward Pass with mean f_t and covariance F_t. Hence the joint distribution $p(h_t, h_{t+1}|v_{1:t})$ has means

$$\langle h_t|v_{1:t}\rangle = f_t, \qquad \langle h_{t+1}|v_{1:t}\rangle = Af_t$$

and covariance elements

$$\left\langle \Delta h_t \Delta h_t^{\mathsf{T}} | v_{1:t}\right\rangle = F_t, \qquad \left\langle \Delta h_t \Delta h_{t+1}^{\mathsf{T}} | v_{1:t}\right\rangle = F_t A^{\mathsf{T}} \tag{23.1.3}$$

$$\left\langle \Delta h_{t+1} \Delta h_{t+1}^{\mathsf{T}} | v_{1:t}\right\rangle = AF_t A^{\mathsf{T}} + \Sigma^h \tag{23.1.4}$$

To find the conditional distribution $p(h_t|v_{1:t}, h_{t+1})$, we use the conditioned Gaussian results which says that the conditional mean will be

$$\langle h_t|v_{1:t}\rangle + \left\langle \Delta h_t \Delta h_{t+1}^{\mathsf{T}} | v_{1:t}\right\rangle \left\langle \Delta h_{t+1} \Delta h_{t+1}^{\mathsf{T}} | v_{1:t}\right\rangle^{-1} \left(h_{t+1} - \langle h_{t+1}|v_{1:t}\rangle\right)$$

and conditional covariance will be

$$\overleftarrow{\Sigma}_t \equiv \left\langle \Delta h_t \Delta h_t^{\mathsf{T}} | v_{1:t}\right\rangle - \left\langle \Delta h_t \Delta h_{t+1}^{\mathsf{T}} | v_{1:t}\right\rangle \left\langle \Delta h_{t+1} \Delta h_{t+1}^{\mathsf{T}} | v_{1:t}\right\rangle^{-1} \langle h_{t+1}|v_{1:t}\rangle$$

From this we can easily write

$$p(h_t|v_{1:t}, h_{t+1}) \equiv h_t = \overleftarrow{A}_t h_{t+1} + \overleftarrow{m}_t + \overleftarrow{\eta}_t$$

where

$$\overleftarrow{A}_t \equiv \left\langle \Delta h_t \Delta h_{t+1}^{\mathsf{T}} | v_{1:t}\right\rangle \left\langle \Delta h_{t+1} \Delta h_{t+1}^{\mathsf{T}} | v_{1:t}\right\rangle^{-1}$$

$$\overleftarrow{m}_t \equiv \langle h_t|v_{1:t}\rangle - \left\langle \Delta h_t \Delta h_{t+1}^{\mathsf{T}} | v_{1:t}\right\rangle \left\langle \Delta h_{t+1} \Delta h_{t+1}^{\mathsf{T}} | v_{1:t}\right\rangle^{-1} \langle h_{t+1}|v_{1:t}\rangle$$

and $\overleftarrow{\eta}_t \sim N(0, \overleftarrow{\Sigma}_t)$. Then $p(h_t|v_{1:T})$ is a Gaussian distribution with mean

$$g_t \equiv \langle h_t|v_{1:T}\rangle = \overleftarrow{A}_t \langle h_{t+1}|v_{1:T}\rangle + \overleftarrow{m}_t \equiv \overleftarrow{A}_t g_{t+1} + \overleftarrow{m}_t$$

and covariance

$$G_t \equiv \left\langle \Delta h_t \Delta h_t^{\mathsf{T}} | v_{1:T} \right\rangle = \overleftarrow{A}_t \left\langle \Delta h_{t+1} \Delta h_{t+1}^{\mathsf{T}} | v_{1:T} \right\rangle \overleftarrow{A}_t^{\mathsf{T}} + \overleftarrow{\Sigma}_t \equiv \overleftarrow{A}_t G_{t+1} \overleftarrow{A}_t^{\mathsf{T}} + \overleftarrow{\Sigma}_t$$

In this way, we directly find the smoothed posterior without defining the problematic λ messages. This procedure is equivalent to the Rauch-Tung-Striebel Kalman smoother[43]. A key trick was dynamics reversal. This is sometimes called a 'correction' method since it takes the filtered estimate $p(h_t|v_{1:t})$ and 'corrects' it to form a smoothed estimate $p(h_t|v_{1:T})$.

This procedure is called the Backward Pass in the LGSSM inference algorithm (albeit with a change to the standard notation in the literature for representing the smoothed posterior).

The cross moment

An advantage of the probabilistic interpretation given above is that the cross moment, which is required for learning is given by

$$\left\langle h_{t-1} h_t^{\mathsf{T}} \right\rangle_{p(h_{t-1}, h_t | v_{1:T})} = \overleftarrow{A_{t-1}} P_t^T + \hat{h}_t^T \left(\hat{h}_t^T \right)^{\mathsf{T}}$$

This is far simpler than common expressions found in the literature.

23.1.3 The Likelihood

The likelihood $p(v_{1:T})$ is often required. To compute this, the simplest way is to use the recursion:

$$p(v_{1:t+1}) = p(v_{t+1}|v_{1:t}) p(v_{1:t})$$

Clearly, $p(v_{t+1}|v_{1:t})$ will be a Gaussian in v_{t+1}. It is straightforward to show that this has mean and covariance (for $t > 1$)

$$\mu_t \equiv BAf_t, \qquad \Sigma_t \equiv B \left(AF_t A^{\mathsf{T}} + \Sigma^h \right) B^{\mathsf{T}} + \Sigma^v$$

Then the log likelihood is given by

$$\sum_{t=1}^{T} -\frac{1}{2} \left(v_t - \mu_t \right)^{\mathsf{T}} \Sigma_t^{-1} \left(v_t - \mu_t \right) - \frac{1}{2} \log \det \left(2\pi \Sigma_t \right)$$

where, at time 1,

$$\mu_1 \equiv B\mu, \qquad \Sigma_1 \equiv B \Sigma B^{\mathsf{T}} + \Sigma^v$$

MAP vs Marginal

In general, we have seen that there is a difference between the most probable joint posterior state, and the joint posterior mean. However, in the case of Gaussians, there is no difference between these two. The interested reader is invited to show formally that the most likely state of a Gaussian is its mean. Hence, in order to infer the most likely hidden state, this is equivalent to finding the marginal – that is, the mean of the hidden variables.

Algorithm 1 LGSSM: Forward and Backward Recursive Updates. The filtered posterior $p(h_t|v_{1:t})$ is returned with means $\hat{h}_t^t$ and covariances P_t^t. The smoothed posterior $p(h_t|v_{1:T})$ means and covariances are $\hat{h}_t^T$ and P_t^T.

```
procedure Forward
    $\hat{h}_1^0 \leftarrow \mu$
    $K \leftarrow \Sigma B^{\mathsf{T}}(B\Sigma B^{\mathsf{T}} + \Sigma_V)^{-1}$
    $P_1^1 \leftarrow (I - KB)\Sigma$
    $\hat{h}_1^1 \leftarrow \hat{h}_1^0 + K(v_t - B\hat{h}_1^0)$
    for $t \leftarrow 2, T$ do
        $P_t^{t-1} \leftarrow AP_{t-1}^{t-1}A^{\mathsf{T}} + \Sigma_H$
        $\hat{h}_t^{t-1} \leftarrow A\hat{h}_{t-1}^{t-1}$
        $K \leftarrow PB^{\mathsf{T}}(BP_t^{t-1}B^{\mathsf{T}} + \Sigma_V)^{-1}$
        $P_t^t \leftarrow (I - KB)P_t^{t-1}$
        $\hat{h}_t^t \leftarrow \hat{h}_t^{t-1} + K(v_t - B\hat{h}_t^{t-1})$
    end for
end procedure
procedure Backward
    for $t \leftarrow T - 1, 1$ do
        $\overleftarrow{A}_t \leftarrow P_t^t A^{\mathsf{T}}(P_{t+1}^t)^{-1}$
        $P_t^T \leftarrow P_t^t + \overleftarrow{A}_t(P_{t+1}^{\mathsf{T}} - P_{t+1}^t)\overleftarrow{A}_t^{\mathsf{T}}$
        $\hat{h}_t^T \leftarrow \hat{h}_t^t + \overleftarrow{A}_t(\hat{h}_{t+1}^{\mathsf{T}} - A\hat{h}_t^t)$
    end for
end procedure
```

23.2 EM Algorithm for Learning

A straightforward application of the EM algorithm yields the following updates:

$$\Sigma_V^{new} = \frac{1}{T}\sum_t \left(v_t v_t^{\mathsf{T}} - v_t \langle h_t \rangle^{\mathsf{T}} B^{\mathsf{T}} - B \langle h_t \rangle v_t^{\mathsf{T}} + B \left\langle h_t h_t^{\mathsf{T}} \right\rangle B^{\mathsf{T}}\right)$$

$$\Sigma_H^{new} = \frac{1}{T-1}\sum_t \left(\left\langle h_{t+1} h_{t+1}^{\mathsf{T}} \right\rangle - A \left\langle h_t h_{t+1}^{\mathsf{T}} \right\rangle - \left\langle h_{t+1} h_t^{\mathsf{T}} \right\rangle A^{\mathsf{T}} + A \left\langle h_t h_t^{\mathsf{T}} \right\rangle A^{\mathsf{T}}\right)$$

$$\mu^{new} = \langle h_1 \rangle$$

$$\Sigma_\pi^{new} = \left\langle h_1 h_1^{\mathsf{T}} \right\rangle - \mu\mu^{\mathsf{T}}$$

$$A^{new} = \sum_{t=1}^{T-1} \left\langle h_{t+1} h_t^{\mathsf{T}} \right\rangle \left(\sum_{t=1}^{T-1} \left\langle h_t h_t^{\mathsf{T}} \right\rangle\right)^{-1}$$

$$B^{new} = \sum_{t=1}^{T} v_t \langle h_t \rangle^{\mathsf{T}} \left(\sum_{t=1}^{T} \left\langle h_t h_t^{\mathsf{T}} \right\rangle\right)^{-1}$$

If B is updated according to the above, the reader may show that the first equation can be simplified to

$$\Sigma_V^{new} = \frac{1}{T}\sum_t \left(v_t v_t^{\mathsf{T}} - v_t \langle h_t \rangle^{\mathsf{T}} B^{\mathsf{T}}\right)$$

Similarly, if A is updated according to EM algorithm, then the second equation can be simplified to

$$\Sigma_H^{new} = \frac{1}{T-1}\sum_t \left(h_{t+1}h_{t+1}^{\mathsf{T}} - A\left\langle h_t h_{t+1}^{\mathsf{T}}\right\rangle\right)$$

Dealing with restricted forms of the matrices is also easy to deal with. For example, it may be that one wishes to search for independent generating processes, in which case A will have a block diagonal structure. This restriction is easy to deal with and left as an exercise for the reader.

The last two equations are solved by Gaussian Elimination. The averages in the above equations are the posterior averages conditioned on the visible variables – these are given by the Kalman Smoother routine.

The extension of learning to multiple time series is straightforward and left as an exercise for the reader.

An example : Simple Trajectory Analysis

A toy rocket is launched in the air. The rocket has unknown mass and initial velocity. In addition, the constant accelerations from rocket's propulsion system are unknown. What is known is that the Newton's laws apply. An instrument can measure the vertical height and horizontal distance of the rocket at each time $x(t), y(t)$ from the origin. Based on noisy measurements of $x(t)$ and $y(t)$, our task is to infer the position of the rocket at each time – a trajectory analysis.

Newton's law states that

$$\frac{d^2}{dt^2}x = \frac{f_x}{m}, \qquad \frac{d^2}{dt^2}y = \frac{f_y}{m}$$

where m is the mass of the object, and f is the (constant) vertical force applied (gravity in this case) Hence

$$\frac{dx}{dt} = t\frac{f_x}{m} + a, \qquad \frac{dy}{dt} = t\frac{f_y}{m} + b$$

$$x = t^2\frac{f_x}{2m} + at + c, \qquad y = t^2\frac{f_y}{2m} + bt + d$$

As they stand, these equations are not in a form directly usable in the LGSSM framework. There are several ways to rewrite them to make them suitable – here we choose a very naive approach. First, we reparameterise time to use the variable $\tilde{t}$ such that $t \equiv \tilde{t}\Delta$

$$x((\tilde{t}+1)\Delta) = x(\tilde{t}\Delta) + \Delta x'(\tilde{t}\Delta)$$

$$y((\tilde{t}+1)\Delta) = y(\tilde{t}\Delta) + \Delta y'(\tilde{t}\Delta)$$

where $y'(t) \equiv \frac{dy}{dt}$. We can write an update equation for the x' and y' as

$$x'((\tilde{t}+1)\Delta) = x'(\tilde{t}\Delta) + f_x\Delta/m, \qquad y'((\tilde{t}+1)\Delta) = y'(\tilde{t}\Delta) + f_y\Delta/m$$

These equations are then discrete time difference equations indexed by $\tilde{t}$. However, the instrument which measures $x(t)$ and $y(t)$ is not completely accurate. What is actually measured is $\hat{x}(t)$ and $\hat{y}(t)$, which are noisy versions of $x(t)$ and $y(t)$. For simplicity, we relabel $a_x(t) = f_x(t)/m(t)$, $a_y(t) = f_y(t)/m(t)$ – these accelerations will be assumed to be roughly constant, but unknown :

$$a_x((\tilde{t}+1)\Delta) = a_x(\tilde{t}\Delta) + \eta_x$$

where η_x is a very small noise term. The prior for a_x is chosen to be vague – a zero mean Gaussian with large variance. A similar equation holds for the a_y. (Of course, another approach would be to assume strictly constant accelerations and learn them).

One way to describe the above approach is to consider $x(t)$, $y(t)$, $x'(t)$, $y'(t)$, $a_x(t)$ and $a_y(t)$ as hidden variables. We can put a large variance prior on their initial values, and attempt to infer the unknown trajectory. A simple demonstration for this is given in fig(26.3), for which the code is given in the text. It is pleasing how well the Kalman Filter infers the object trajectory despite the large amount of measurement noise.

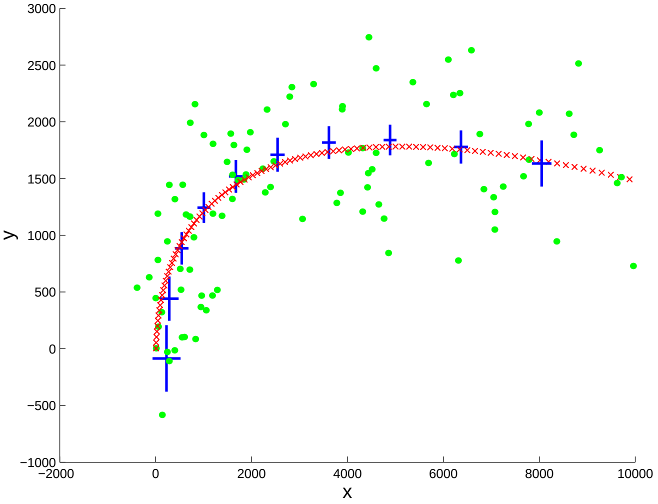


Figure 23.2: Kalman Smoother estimate of a trajectory based on noisy observations. The small points are the noisy observations (which have also a time label). The “x” points are the true positions of the object, and the crosses are the estimated positions of the object plotted every several time steps. The estimates are plotted with standard errors around their mean values.

23.3 Problems

Exercise 90 *A scaler R^{th} order Autoregressive Model (AR) model is defined as*

$$v_{t+1} = \sum_{i=1}^{R} a_i v_{t-i} + \eta_{t+1}$$

where v_t is a scalar, and η is Gaussian noise. Explain how to formulate an R^{th} order AR model as a first order LGSSM.

Exercise 91

- *Explain how to model a sinusoid, rotating with angular velocity ω using a two-dimensional LGSSM.*
- *Explain how to model a sinuoid using an AR model.*
- *Explain the relationship between the second order differential equation $\ddot{x} = -\lambda x$, which describes a Harmonic Oscillator, and the second order difference equation which approximates this differential equation. Is it possible to find a difference equation which exactly matches the solution of the differential equation at chosen points?*

23.4 Solutions

56

24 Switching Linear Dynamical Systems

The Linear Dynamical System (LDS), chapter(23) is a key temporal model in which a latent linear process generates the observed series. For more complex time-series which are not well described globally by a single LDS, we may break the time-series into segments, each modelled by a potentially different LDS. This is the basis for the Switching LDS (SLDS) where, for each time t, a switch variable $s_t \in 1, \ldots, S$ describes which of the LDSs is to be used[1]. The observation (or 'visible') $v_t \in \mathcal{R}^V$ is linearly related to the hidden state $h_t \in \mathcal{R}^H$ by

$$v_t = B(s_t)h_t + \eta^v(s_t), \qquad \eta^v(s_t) \sim \mathcal{N}\left(\bar{v}(s_t), \Sigma^v(s_t)\right) \tag{24.0.1}$$

where $\mathcal{N}(\mu, \Sigma)$ denotes a Gaussian distribution with mean μ and covariance Σ. The transition dynamics of the continuous hidden state h_t is linear,

$$h_t = A(s_t)h_{t-1} + \eta^h(s_t), \qquad \eta^h(s_t) \sim \mathcal{N}\left(\bar{h}(s_t), \Sigma^h(s_t)\right) \tag{24.0.2}$$

The switch variable s_t itself is Markovian, with transition $p(s_t|s_{t-1})$.

Here we consider the more general 'augmented' model in which the switch s_t is dependent on both the previous s_{t-1} and h_{t-1}. An equivalent probabilistic model is[2] (see fig(24.1))

$$p(v_{1:T}, h_{1:T}, s_{1:T}) = \prod_{t=1}^{T} p(v_t|h_t, s_t)p(h_t|h_{t-1}, s_t)p(s_t|h_{t-1}, s_{t-1})$$

with

$$p(v_t|h_t, s_t) = \mathcal{N}\left(\bar{v}(s_t) + B(s_t)h_t, \Sigma^v(s_t)\right), \qquad p(h_t|h_{t-1}, s_t) = \mathcal{N}\left(\bar{h}(s_t) + A(s_t)h_t, \Sigma^h(s_t)\right)$$

At time $t = 1$, $p(s_1|h_0, s_0)$ simply denotes the prior $p(s_1)$, and $p(h_1|h_0, s_1)$ denotes $p(h_1|s_1)$.

The SLDS is used in many disciplines, from econometrics to machine learning [44, 45, 46, 47, 48, 49]. The aSLDS has been used, for example, in state-duration modelling in acoustics [50] and econometrics [51]. See [52] and [53] for recent reviews of work.

The SLDS can be thought of as a marriage between a Hidden Markov Model and a Linear Dynamical system. Each of these two models are tractable. However, the SLDS is computationally *intractable*, and requires specialised approximations.

[1] These systems also go under the names Jump Markov model/process, switching Kalman Filter, Switching Linear Gaussian State Space models, Conditional Linear Gaussian Models.

[2] The notation $x_{1:T}$ is shorthand for $x_1, \ldots, x_T$.

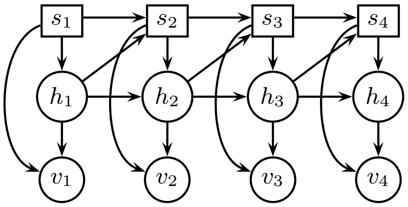


Figure 24.1: The independence structure of the aSLDS. Square nodes denote discrete variables, round nodes continuous variables. In the SLDS links from h to s are not normally considered.

Inference

We consider here the *filtered* estimate $p(h_t, s_t|v_{1:t})$ and the *smoothed* estimate $p(h_t, s_t|v_{1:T})$, for any $1 \leq t \leq T$. Both filtered and smoothed inference in the SLDS is intractable, scaling exponentially with time [52]. To see this informally, consider the filtered posterior, which may be recursively computed using

$$p(s_t, h_t|v_{1:t}) = \sum_{s_{t-1}} \int_{h_{t-1}} p(s_t, h_t|s_{t-1}, h_{t-1}, v_t) p(s_{t-1}, h_{t-1}|v_{1:t-1}) \qquad (24.0.3)$$

At timestep 1, $p(s_1, h_1|v_1) = p(h_1|s_1, v_1)p(s_1|v_1)$ is an indexed set of Gaussians. At timestep 2, due to the summation over the states s_1, $p(s_2, h_2|v_{1:2})$ will be an indexed set of S Gaussians; similarly at timestep 3, it will be S^2 and, in general, gives rise to S^t Gaussians.

24.1 Expectation Correction

EC mirrors the Rauch-Tung-Striebel 'correction' smoother for the LDS [43, 44] presented in chapter(23). The correction approach consists of a single forward pass to recursively find the filtered posterior $p(h_t, s_t|v_{1:t})$, followed by a single backward pass to correct this into a smoothed posterior $p(h_t, s_t|v_{1:T})$. The forward pass is equivalent to standard Assumed Density Filtering [54].

24.1.1 Forward Pass (Filtering)

Readers familiar with Assumed Density Filtering may wish to continue directly to section (24.1.3). Our aim is to form a recursion for $p(s_t, h_t|v_{1:t})$, based on a Gaussian mixture approximation[3] of $p(h_t|s_t, v_{1:t})$. Without loss of generality, we may decompose the filtered posterior as

$$p(h_t, s_t|v_{1:t}) = p(h_t|s_t, v_{1:t})p(s_t|v_{1:t}) \qquad (24.1.1)$$

The exact representation of $p(h_t|s_t, v_{1:t})$ is a mixture with a $O(S^t)$ components. We therefore approximate this with a smaller I-component mixture

$$p(h_t|s_t, v_{1:t}) \approx \sum_{i_t=1}^{I} p(h_t|i_t, s_t, v_{1:t})p(i_t|s_t, v_{1:t})$$

[3] This derivation holds also for the aSLDS, unlike that presented in [55].

where $p(h_t|i_t, s_t, v_{1:t})$ is a Gaussian parameterised with mean[4] $f(i_t, s_t)$ and covariance $F(i_t, s_t)$. To find a recursion for these parameters, consider

$$\begin{aligned} p(h_{t+1}|s_{t+1}, v_{1:t+1}) &= \sum_{s_t, i_t} p(h_{t+1}, s_t, i_t|s_{t+1}, v_{1:t+1}) \\ &= \sum_{s_t, i_t} p(h_{t+1}|s_t, i_t, s_{t+1}, v_{1:t+1}) p(s_t, i_t|s_{t+1}, v_{1:t+1}) \end{aligned} \tag{24.1.2}$$

Evaluating $p(h_{t+1}|s_t, i_t, s_{t+1}, v_{1:t+1})$

We find $p(h_{t+1}|s_t, i_t, s_{t+1}, v_{1:t+1})$ from the joint distribution $p(h_{t+1}, v_{t+1}|s_t, i_t, s_{t+1}, v_{1:t})$, which is a Gaussian with covariance and mean elements[5]

$$\begin{aligned} \Sigma_{hh} &= A(s_{t+1})F(i_t, s_t)A^{\mathsf{T}}(s_{t+1}) + \Sigma^h(s_{t+1}), \\ \Sigma_{vv} &= B(s_{t+1})\Sigma_{hh}B^{\mathsf{T}}(s_{t+1}) + \Sigma^v(s_{t+1}) \\ \Sigma_{vh} &= B(s_{t+1})F(i_t, s_t) \\ \mu_v &= B(s_{t+1})A(s_{t+1})f(i_t, s_t) \\ \mu_h &= A(s_{t+1})f(i_t, s_t) \end{aligned} \tag{24.1.3}$$

These results are obtained from integrating the forward dynamics, Equations (24.0.1,24.0.2) over h_t, using the results in Appendix (G.2). To find $p(h_{t+1}|s_t, i_t, s_{t+1}, v_{1:t+1})$ we may then condition $p(h_{t+1}, v_{t+1}|s_t, i_t, s_{t+1}, v_{1:t})$ on v_{t+1} using the results in Appendix (G.1).

Evaluating $p(s_t, i_t|s_{t+1}, v_{1:t+1})$

Up to a trivial normalisation constant the mixture weight in equation (24.1.2) can be found from the decomposition

$$p(s_t, i_t|s_{t+1}, v_{1:t+1}) \propto p(v_{t+1}|i_t, s_t, s_{t+1}, v_{1:t})p(s_{t+1}|i_t, s_t, v_{1:t})p(i_t|s_t, v_{1:t})p(s_t|v_{1:t}) \tag{24.1.4}$$

The first factor in equation (24.1.4), $p(v_{t+1}|i_t, s_t, s_{t+1}, v_{1:t})$ is given as a Gaussian with mean μ_v and covariance Σ_{vv}, as given in equation (24.1.3). The last two factors $p(i_t|s_t, v_{1:t})$ and $p(s_t|v_{1:t})$ are given from the previous iteration. Finally, $p(s_{t+1}|i_t, s_t, v_{1:t})$ is found from

$$p(s_{t+1}|i_t, s_t, v_{1:t}) = \langle p(s_{t+1}|h_t, s_t)\rangle_{p(h_t|i_t, s_t, v_{1:t})} \tag{24.1.5}$$

where $\langle\cdot\rangle_p$ denotes expectation with respect to p. In the standard SLDS, equation (24.1.5) is replaced by the Markov transition $p(s_{t+1}|s_t)$. In the aSLDS, however, equation (24.1.5) will generally need to be computed numerically. A simple approximation is to evaluate equation (24.1.5) at the mean value of the distribution $p(h_t|i_t, s_t, v_{1:t})$. To take covariance information into account an alternative would be to draw samples from the Gaussian $p(h_t|i_t, s_t, v_{1:t})$ and thus approximate the average of $p(s_{t+1}|h_t, s_t)$ by sampling[6].

[4] Strictly speaking, we should use the notation $f_t(i_t, s_t)$ since, for each time t, we have a set of means indexed by i_t, s_t. This mild abuse of notation is used elsewhere in the paper.

[5] We derive this for $\bar{h}_{t+1}, \bar{v}_{t+1} \equiv 0$, to ease notation.

[6] Whilst we suggest sampling as part of the aSLDS update procedure, this does not equate this with a sequential sampling procedure, such as Particle Filtering. The sampling here is a form of exact sampling, for which no convergence issues arise, being used only to numerically compute equation (24.1.5).

Closing the recursion

We are now in a position to calculate equation (24.1.2). For each setting of the variable s_{t+1}, we have a mixture of $I \times S$ Gaussians which we numerically collapse back to I Gaussians to form

$$p(h_{t+1}|s_{t+1}, v_{1:t+1}) \approx \sum_{i_{t+1}=1}^{I} p(h_{t+1}|i_{t+1}, s_{t+1}, v_{1:t+1}) p(i_{t+1}|s_{t+1}, v_{1:t+1})$$

Any method of choice may be supplied to collapse a mixture to a smaller mixture. A straightforward approach that we use in our code is based on repeatedly merging low-weight components, as explained in section (24.1.2). In this way the new mixture coefficients $p(i_{t+1}|s_{t+1}, v_{1:t+1})$, $i_{t+1} \in 1, \ldots, I$ are defined.

The above completes the description of how to form a recursion for $p(h_{t+1}|s_{t+1}, v_{1:t+1})$ in equation (24.1.1). A recursion for the switch variable is given by

$$p(s_{t+1}|v_{1:t+1}) \propto \sum_{i_t, s_t} p(s_{t+1}, i_t, s_t, v_{t+1}, v_{1:t})$$

The r.h.s. of the above equation is proportional to

$$\sum_{s_t, i_t} p(v_{t+1}|s_{t+1}, i_t, s_t, v_{1:t}) p(s_{t+1}|i_t, s_t, v_{1:t}) p(i_t|s_t, v_{1:t}) p(s_t|v_{1:t})$$

where all terms have been computed during the recursion for $p(h_{t+1}|s_{t+1}, v_{1:t+1})$.

The Likelihood $p(v_{1:T})$

The likelihood $p(v_{1:T})$ may be found by recursing $p(v_{1:t+1}) = p(v_{t+1}|v_{1:t})p(v_{1:t})$, where

$$p(v_{t+1}|v_t) = \sum_{i_t, s_t, s_{t+1}} p(v_{t+1}|i_t, s_t, s_{t+1}, v_{1:t}) p(s_{t+1}|i_t, s_t, v_{1:t}) p(i_t|s_t, v_{1:t}) p(s_t|v_{1:t})$$

In the above expression, all terms have been computed in forming the recursion for the filtered posterior $p(h_{t+1}, s_{t+1}|v_{1:t+1})$.

The procedure for computing the filtered posterior is presented in 2.**??**

24.1.2 Collapsing Gaussians

The user may provide any algorithm of their choice for collapsing a set of Gaussians to a smaller set of Gaussians [56]. Here, to be explicit, we present a simple one which is fast, but has the disadvantage that no spatial information about the mixture is used.

First, we describe how to collapse a mixture to a *single* Gaussian: We may collapse a mixture of Gaussians $p(x) = \sum_i p_i \mathcal{N}(x|\mu_i, \Sigma_i)$ to a single Gaussian with mean $\sum_i p_i \mu_i$ and covariance $\sum_i p_i \left(\Sigma_i + \mu_i \mu_i^{\mathsf{T}}\right) - \mu\mu^{\mathsf{T}}$.

To collapse a mixture to a K-component *mixture* we retain the $K-1$ Gaussians with the largest mixture weights – the remaining $N-K$ Gaussians are simply merged to a single Gaussian using the above method. The alternative of recursively merging the two Gaussians with the lowest mixture weights gave similar experimental performance.

Algorithm 2 aSLDS Forward Pass. Approximate the filtered posterior $p(s_t|v_{1:t}) \equiv \rho_t$, $p(h_t|s_t, v_{1:t}) \equiv \sum_{i_t} w_t(i_t, s_t)\mathcal{N}(f_t(i_t, s_t), F_t(i_t, s_t))$. Also we return the approximate log-likelihood $\log p(v_{1:T})$. We require $I_1 = 1, I_2 \leq S, I_t \leq S \times I_{t-1}$. $\theta(s) = A(s), B(s), \Sigma^h(s), \Sigma^v(s), \bar{h}(s), \bar{v}(s)$.

```
for $s_1 \leftarrow 1$ to $S$ do
    $\{f_1(1, s_1), F_1(1, s_1), \hat{p}\} = \text{LDSFORWARD}(0, 0, v_1; \theta(s_1))$
    $\rho_1 \leftarrow p(s_1)\hat{p}$
end for

for $t \leftarrow 2$ to $T$ do
    for $s_t \leftarrow 1$ to $S$ do
        for $i \leftarrow 1$ to $I_{t-1}$, and $s \leftarrow 1$ to $S$ do
            $\{\mu_{x|y}(i, s), \Sigma_{x|y}(i, s), \hat{p}\} = \text{LDSFORWARD}(f_{t-1}(i, s), F_{t-1}(i, s), v_t; \theta(s_t))$
            $p^*(s_t|i, s) \equiv \langle p(s_t|h_{t-1}, s_{t-1} = s)\rangle_{p(h_{t-1}|i_{t-1}=i, s_{t-1}=s, v_{1:t-1})}$
            $p'(s_t, i, s) \leftarrow w_{t-1}(i, s)p^*(s_t|i, s)\rho_{t-1}(s)\hat{p}$
        end for
        Collapse the $I_{t-1} \times S$ mixture of Gaussians defined by $\mu_{x|y}$,$\Sigma_{x|y}$, and weights $p(i, s|s_t) \propto p'(s_t, i, s)$ to a Gaussian with $I_t$ components, $p(h_t|s_t, v_{1:t}) \approx \sum_{i_t=1}^{I_t} p(i_t|s_t, v_{1:t})p(h_t|s_t, i_t, v_{1:t})$. This defines the new means $f_t(i_t, s_t)$, covariances $F_t(i_t, s_t)$ and mixture weights $w_t(i_t, s_t) \equiv p(i_t|s_t, v_{1:t})$.
        Compute $\rho_t(s_t) \propto \sum_{i,s} p'(s_t, i, s)$
    end for
    normalise $\rho_t$
    $L \leftarrow L + \log \sum_{s_t,i,s} p'(s_t, i, s)$
end for
```

More sophisticated methods which retain some spatial information would clearly be potentially useful. The method presented in [46] is a suitable approach which considers removing Gaussians which are spatially similar (and not just low-weight components), thereby retaining a sense of diversity over the possible solutions.

24.1.3 Backward Pass (Smoothing)

The main difficulty is to find a suitable way to 'correct' the filtered posterior $p(s_t, h_t|v_{1:t})$ obtained from the forward pass into a smoothed posterior $p(s_t, h_t|v_{1:T})$. We initially derive this for the case of a single Gaussian representation. The extension to the mixture case is straightforward and is given in section (24.1.5). Our derivation holds for both the SLDS and aSLDS. We approximate the smoothed posterior $p(h_t|s_t, v_{1:T})$ by a Gaussian with mean $g(s_t)$ and covariance $G(s_t)$, and our aim is to find a recursion for these parameters. A useful starting point for a recursion is:

$$p(h_t, s_t|v_{1:T}) = \sum_{s_{t+1}} p(s_{t+1}|v_{1:T})p(h_t|s_t, s_{t+1}, v_{1:T})p(s_t|s_{t+1}, v_{1:T})$$

The term $p(h_t|s_t, s_{t+1}, v_{1:T})$ may be computed as

$$\begin{aligned} p(h_t|s_t, s_{t+1}, v_{1:T}) &= \int_{h_{t+1}} p(h_t, h_{t+1}|s_t, s_{t+1}, v_{1:T}) \\ &= \int_{h_{t+1}} p(h_t|h_{t+1}, s_t, s_{t+1}, v_{1:T}) p(h_{t+1}|s_t, s_{t+1}, v_{1:T}) \\ &= \int_{h_{t+1}} p(h_t|h_{t+1}, s_t, s_{t+1}, v_{1:t}) p(h_{t+1}|s_t, s_{t+1}, v_{1:T}) \end{aligned} \tag{24.1.6}$$

The recursion therefore requires $p(h_{t+1}|s_t, s_{t+1}, v_{1:T})$, which we can write as

$$p(h_{t+1}|s_t, s_{t+1}, v_{1:T}) \propto p(h_{t+1}|s_{t+1}, v_{1:T}) p(s_t|s_{t+1}, h_{t+1}, v_{1:t}) \tag{24.1.7}$$

The difficulty here is that the functional form of $p(s_t|s_{t+1}, h_{t+1}, v_{1:t})$ is not squared exponential in h_{t+1}, so that $p(h_{t+1}|s_t, s_{t+1}, v_{1:T})$ will not be Gaussian. One possibility would be to approximate the non-Gaussian $p(h_{t+1}|s_t, s_{t+1}, v_{1:T})$ by a Gaussian (or mixture thereof) by minimising the Kullback-Leilbler divergence between the two, or performing moment matching in the case of a single Gaussian. A simpler alternative is to make the assumption $p(h_{t+1}|s_t, s_{t+1}, v_{1:T}) \approx p(h_{t+1}|s_{t+1}, v_{1:T})$, see fig(24.2). This makes life easy since $p(h_{t+1}|s_{t+1}, v_{1:T})$ is already known from the previous backward recursion. Under this assumption, the recursion becomes

$$p(h_t, s_t|v_{1:T}) \approx \sum_{s_{t+1}} p(s_{t+1}|v_{1:T}) p(s_t|s_{t+1}, v_{1:T}) \left\langle p(h_t|h_{t+1}, s_t, s_{t+1}, v_{1:t}) \right\rangle_{p(h_{t+1}|s_{t+1}, v_{1:T})} \tag{24.1.8}$$

The procedure resulting from the conditional independence assumption is called 'standard' EC. Equation (24.1.8) forms the basis of the standard EC backward pass. How to implement the recursion for the continuous and discrete factors is detailed below[7].

Evaluating $\langle p(h_t|h_{t+1}, s_t, s_{t+1}, v_{1:t})\rangle_{p(h_{t+1}|s_{t+1}, v_{1:T})}$

$\langle p(h_t|h_{t+1}, s_t, s_{t+1}, v_{1:t})\rangle_{p(h_{t+1}|s_{t+1}, v_{1:T})}$ is a Gaussian in h_t, whose statistics we will now compute. First we find $p(h_t|h_{t+1}, s_t, s_{t+1}, v_{1:t})$ which may be obtained from the joint distribution

$$p(h_t, h_{t+1}|s_t, s_{t+1}, v_{1:t}) = p(h_{t+1}|h_t, s_{t+1}) p(h_t|s_t, v_{1:t}) \tag{24.1.9}$$

which itself can be found from a forward dynamics from the filtered estimate $p(h_t|s_t, v_{1:t})$. The statistics for the marginal $p(h_t|s_t, s_{t+1}, v_{1:t})$ are simply those of $p(h_t|s_t, v_{1:t})$, since s_{t+1} carries no extra information about h_t[8]. The only remaining

[7] Equation (24.1.8) has the pleasing form of an RTS backpass for the continuous part (analogous to LDS case), and a discrete smoother (analogous to a smoother recursion for the HMM). In the standard Forward-Backward algorithm for the HMM [40], the posterior $\gamma_t \equiv p(s_t|v_{1:T})$ is formed from the product of $\alpha_t \equiv p(s_t|v_{1:t})$ and $\beta_t \equiv p(v_{t+1:T}|s_t)$. This approach is also analogous to EP [41]. In the correction approach, a direct recursion for γ_t in terms of γ_{t+1} and α_t is formed, without explicitly defining β_t. The two approaches to inference are known as $\alpha - \beta$ and $\alpha - \gamma$ recursions.

[8] Integrating over h_{t+1} means that the information from s_{t+1} passing through h_{t+1} via the term $p(h_{t+1}|s_{t+1}, h_t)$ vanishes. Also, since s_t is known, no information from s_{t+1} passes through s_t to h_t.

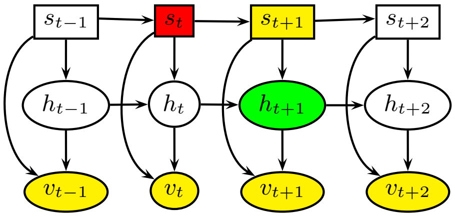


Figure 24.2: The EC backpass approximates $p(h_{t+1}|s_{t+1}, s_t, v_{1:T})$ by $p(h_{t+1}|s_{t+1}, v_{1:T})$. Motivation for this is that s_t only influences h_{t+1} through h_t. However, h_t will most likely be heavily influenced by $v_{1:t}$, so that not knowing the state of s_t is likely to be of secondary importance. The green (darker) node is the variable we wish to find the posterior state of. The yellow (lighter shaded) nodes are variables in known states, and the hashed node a variable whose state is indeed known but assumed unknown for the approximation.

uncomputed statistics are the mean of h_{t+1}, the covariance of h_{t+1} and cross-variance between h_t and h_{t+1}, which are given by

$$\langle h_{t+1} \rangle = A(s_{t+1}) f_t(s_t)$$
$$\Sigma_{t+1,t+1} = A(s_{t+1}) F_t(s_t) A^{\mathsf{T}}(s_{t+1}) + \Sigma^h(s_{t+1}), \qquad \Sigma_{t+1,t} = A(s_{t+1}) F_t(s_t)$$

Given the statistics of equation (24.1.9), we may now condition on h_{t+1} to find $p(h_t|h_{t+1}, s_t, s_{t+1}, v_{1:t})$. Doing so effectively constitutes a reversal of the dynamics,

$$h_t = \overleftarrow{A}(s_t, s_{t+1}) h_{t+1} + \overleftarrow{\eta}(s_t, s_{t+1})$$

where $\overleftarrow{A}$ and $\overleftarrow{\eta}(s_t, s_{t+1}) \sim \mathcal{N}(\overleftarrow{m}(s_t, s_{t+1}), \overleftarrow{\Sigma}(s_t, s_{t+1}))$ are easily found using the conditioned Gaussian results in Appendix (G.1). Averaging the above reversed dynamics over $p(h_{t+1}|s_{t+1}, v_{1:T})$, we find that $\langle p(h_t|h_{t+1}, s_t, s_{t+1}, v_{1:t}) \rangle_{p(h_{t+1}|s_{t+1}, v_{1:T})}$ is a Gaussian with statistics

$$\mu_t = \overleftarrow{A}(s_t, s_{t+1}) g(s_{t+1}) + \overleftarrow{m}(s_t, s_{t+1}), \quad \Sigma_{t,t} = \overleftarrow{A}(s_t, s_{t+1}) G(s_{t+1}) \overleftarrow{A}^{\mathsf{T}}(s_t, s_{t+1}) + \overleftarrow{\Sigma}(s_t, s_{t+1})$$

These equations directly mirror the standard RTS backward pass.

Evaluating $p(s_t|s_{t+1}, v_{1:T})$

The main departure of EC from related methods is in treating the term

$$p(s_t|s_{t+1}, v_{1:T}) = \langle p(s_t|h_{t+1}, s_{t+1}, v_{1:t}) \rangle_{p(h_{t+1}|s_{t+1}, v_{1:T})} \tag{24.1.10}$$

The term $p(s_t|h_{t+1}, s_{t+1}, v_{1:t})$ is given by

$$p(s_t|h_{t+1}, s_{t+1}, v_{1:t}) = \frac{p(h_{t+1}|s_{t+1}, s_t, v_{1:t}) p(s_t, s_{t+1}|v_{1:t})}{\sum_{s'_t} p(h_{t+1}|s_{t+1}, s'_t, v_{1:t}) p(s'_t, s_{t+1}|v_{1:t})} \tag{24.1.11}$$

Here $p(s_t, s_{t+1}|v_{1:t}) = p(s_{t+1}|s_t, v_{1:t}) p(s_t|v_{1:t})$, where $p(s_{t+1}|s_t, v_{1:t})$ occurs in the forward pass, equation (24.1.5). In equation (24.1.11), $p(h_{t+1}|s_{t+1}, s_t, v_{1:t})$ is found by marginalising equation (24.1.9).

Computing the average of equation (24.1.11) with respect to $p(h_{t+1}|s_{t+1}, v_{1:T})$ may be achieved by any numerical integration method desired. The simplest approximation is to evaluate the integrand at the mean value of the averaging distribution[9] $p(h_{t+1}|s_{t+1}, v_{1:T})$. Otherwise, sampling from the Gaussian $p(h_{t+1}|s_{t+1}, v_{1:T})$, has the advantage that covariance information is used[10].

Closing the Recursion

We have now computed both the continuous and discrete factors in equation (24.1.8), which we wish to use to write the smoothed estimate in the form $p(h_t, s_t|v_{1:T}) = p(s_t|v_{1:T})p(h_t|s_t, v_{1:T})$. The distribution $p(h_t|s_t, v_{1:T})$ is readily obtained from the joint equation (24.1.8) by conditioning on s_t to form the mixture

$$p(h_t|s_t, v_{1:T}) = \sum_{s_{t+1}} p(s_{t+1}|s_t, v_{1:T})p(h_t|s_t, s_{t+1}, v_{1:T})$$

which may be collapsed to a single Gaussian (or mixture if desired). The smoothed posterior $p(s_t|v_{1:T})$ is given by

$$\begin{aligned} p(s_t|v_{1:T}) &= \sum_{s_{t+1}} p(s_{t+1}|v_{1:T})p(s_t|s_{t+1}, v_{1:T}) \\ &= \sum_{s_{t+1}} p(s_{t+1}|v_{1:T}) \left\langle p(s_t|h_{t+1}, s_{t+1}, v_{1:t}) \right\rangle_{p(h_{t+1}|s_{t+1}, v_{1:T})} . \qquad (24.1.12) \end{aligned}$$

Numerical Stability

Numerical stability is a concern even in the LDS, and the same is to be expected for the aSLDS. Since the standard LDS recursions LDSFORWARD and LDSBACKWARD are embedded within the EC algorithm, we may immediately take advantage of the large body of work on stabilizing the LDS recursions, such as the Joseph or square root forms [57].

24.1.4 Remarks

The standard-EC Backpass procedure is closely related to Kim's method [58, 48]. In both standard-EC and Kim's method, the approximation $p(h_{t+1}|s_t, s_{t+1}, v_{1:T}) \approx p(h_{t+1}|s_{t+1}, v_{1:T})$, is used to form a numerically simple backward pass. The other 'approximation' in EC is to numerically compute the average in equation (24.1.12). In Kim's method, however, an update for the discrete variables is formed by replacing the required term in equation (24.1.12) by

$$\left\langle p(s_t|h_{t+1}, s_{t+1}, v_{1:t}) \right\rangle_{p(h_{t+1}|s_{t+1}, v_{1:T})} \approx p(s_t|s_{t+1}, v_{1:t}) \qquad (24.1.13)$$

This approximation[11] decouples the discrete backward pass in Kim's method from the continuous dynamics, since $p(s_t|s_{t+1}, v_{1:t}) \propto p(s_{t+1}|s_t)p(s_t|v_{1:t})/p(s_{t+1}|v_{1:t})$

[9] Replacing h_{t+1} by its mean gives the simple approximation

$$\left\langle p(s_t|h_{t+1}, s_{t+1}, v_{1:t}) \right\rangle_{p(h_{t+1}|s_{t+1}, v_{1:T})} \approx \frac{1}{Z} \frac{e^{-\frac{1}{2} z_{t+1}^{\mathsf{T}}(s_t, s_{t+1}) \Sigma^{-1}(s_t, s_{t+1}|v_{1:t}) z_{t+1}(s_t, s_{t+1})}}{\sqrt{\det \Sigma(s_t, s_{t+1}|v_{1:t})}} p(s_t|s_{t+1}, v_{1:t})$$

where $z_{t+1}(s_t, s_{t+1}) \equiv \langle h_{t+1}|s_{t+1}, v_{1:T}\rangle - \langle h_{t+1}|s_t, s_{t+1}, v_{1:t}\rangle$ and Z ensures normalisation over s_t. $\Sigma(s_t, s_{t+1}|v_{1:t})$ is the filtered covariance of h_{t+1} given s_t, s_{t+1} and the observations $v_{1:t}$, which may be taken from Σ_{hh} in equation (24.1.3).

[10] This is a form of exact sampling since drawing samples from a Gaussian is easy. This should not be confused with meaning that this use of sampling renders EC a sequential Monte-Carlo sampling scheme.

[11] In the HMM, this is exact, but in the SLDS the future observations carry information about s_t.

Algorithm 3 aSLDS: EC Backward Pass. Approximates $p(s_t|v_{1:T})$ and $p(h_t|s_t, v_{1:T}) \equiv \sum_{j_t=1}^{J_t} u_t(j_t, s_t)\mathcal{N}(g_t(j_t, s_t), G_t(j_t, s_t))$ using a mixture of Gaussians. $J_T = I_T, J_t \leq S \times I_t \times J_{t+1}$. This routine needs the results from 2.**??**

```
$G_T \leftarrow F_T$, $g_T \leftarrow f_T$, $u_T \leftarrow w_T$
for $t \leftarrow T-1$ to $1$ do
    for $s \leftarrow 1$ to $S$, $s' \leftarrow 1$ to $S$, $i \leftarrow 1$ to $I_t$, $j' \leftarrow 1$ to $J_{t+1}$ do
        $(\mu, \Sigma)(i, s, j', s') = \text{LDSBACKWARD}(g_{t+1}(j', s'), G_{t+1}(j', s'), f_t(i, s), F_t(i, s), \theta(s'))$
        $p(i, s|j', s') = \langle p(s_t = s, i_t = i|h_{t+1}, s_{t+1} = s', j_{t+1} = j', v_{1:t})\rangle_{p(h_{t+1}|s_{t+1}=s', j_{t+1}=j', v_{1:T})}$
        $p(i, s, j', s'|v_{1:T}) \leftarrow p(s_{t+1} = s'|v_{1:T})u_{t+1}(j', s')p(i, s|j', s')$
    end for
    for $s_t \leftarrow 1$ to $S$ do
        Collapse the mixture defined by weights $p(i_t = i, s_{t+1} = s', j_{t+1} = j'|s_t, v_{1_T}) \propto p(i, s_t, j', s'|v_{1:T})$, means $\mu(i_t, s_t, j_{t+1}, s_{t+1})$ and covariances $\Sigma(i_t, s_t, j_{t+1}, s_{t+1})$ to a mixture with $J_t$ components. This defines the new means $g_t(j_t, s_t)$, covariances $G_t(j_t, s_t)$ and mixture weights $u_t(j_t, s_t)$.
        $p(s_t|v_{1:T}) \leftarrow \sum_{i_t, j', s'} p(i_t, s_t, j', s'|v_{1:T})$
    end for
end for
```

can be computed simply from the filtered results alone. The fundamental difference therefore between EC and Kim's method is that the approximation, equation (24.1.13), is not required by EC. The EC backward pass therefore makes fuller use of the future information, resulting in a recursion which intimately couples the continuous and discrete variables. Unlike [58] and [46], where $g_t, G_t \equiv f_t, F_t$ and only the backward pass mixture weights are updated from the forward pass, EC actually changes the Gaussian parameters g_t, G_t in a non-trivial way. The resulting effect on the quality of the approximation can be profound, as we will see in the experiments.

The Expectation Propagation algorithm, discussed in more detail in section (24.2), makes the central assumption, as in EC, of collapsing the posteriors to a Gaussian family [53]. However, in EP, collapsing to a mixture of Gaussians is difficult – indeed, even working with a single Gaussian may be numerically unstable. In contrast, EC works largely with moment parameterisations of Gaussians, for which relatively few numerical difficulties arise. As explained in the derivation of equation (24.1.8), the conditional independence assumption $p(h_{t+1}|s_t, s_{t+1}, v_{1:T}) \approx p(h_{t+1}|s_{t+1}, v_{1:T})$ is not strictly necessary in EC. We motivate it by computational simplicity, since finding an appropriate moment matching approximation of $p(h_{t+1}|s_t, s_{t+1}, v_{1:T})$ in equation (24.1.7) requires a relatively expensive non-Gaussian integration. The important point here is that, if we did treat $p(h_{t+1}|s_t, s_{t+1}, v_{1:T})$ more correctly, the only assumption in EC would be a collapse to a mixture of Gaussians, as in EP. As a point of interest, as in EC, the exact computation requires only a single forward and backward pass, whilst EP is an 'open' procedure requiring iteration to convergence.

24.1.5 Using Mixtures in the Backward Pass

The extension to the mixture case is straightforward, based on the representation

$$p(h_t|s_t, v_{1:T}) \approx \sum_{j_t=1}^{J} p(j_t|s_t, v_{1:T})p(h_t|j_t, v_{1:T}).$$

Analogously to the case with a single component,

$$p(h_t, s_t|v_{1:T}) = \sum_{i_t, j_{t+1}, s_{t+1}} p(s_{t+1}|v_{1:T}) p(j_{t+1}|s_{t+1}, v_{1:T}) p(h_t|j_{t+1}, s_{t+1}, i_t, s_t, v_{1:T}) \cdot \langle p(i_t, s_t|h_{t+1}, j_{t+1}, s_{t+1}, v_{1:t}) \rangle_{p(h_{t+1}|j_{t+1}, s_{t+1}, v_{1:T})}$$

The average in the last line of the above equation can be tackled using the same techniques as outlined in the single Gaussian case. To approximate $p(h_t|j_{t+1}, s_{t+1}, i_t, s_t, v_{1:T})$ we consider this as the marginal of the joint distribution

$$p(h_t, h_{t+1}|i_t, s_t, j_{t+1}, s_{t+1}, v_{1:T}) = p(h_t|h_{t+1}, i_t, s_t, j_{t+1}, s_{t+1}, v_{1:t}) p(h_{t+1}|i_t, s_t, j_{t+1}, s_{t+1}, v_{1:T})$$

As in the case of a single mixture, the problematic term is $p(h_{t+1}|i_t, s_t, j_{t+1}, s_{t+1}, v_{1:T})$. Analogously to before, we may make the assumption

$$p(h_{t+1}|i_t, s_t, j_{t+1}, s_{t+1}, v_{1:T}) \approx p(h_{t+1}|j_{t+1}, s_{t+1}, v_{1:T})$$

meaning that information about the current switch state s_t, i_t is ignored. As in the single component case, in principle, this assumption may be relaxed and a moment matching approximation be performed instead. We can then form

$$p(h_t|s_t, v_{1:T}) = \sum_{i_t, j_{t+1}, s_{t+1}} p(i_t, j_{t+1}, s_{t+1}|s_t, v_{1:T}) p(h_t|i_t, s_t, j_{t+1}, s_{t+1}, v_{1:T})$$

This mixture can then be collapsed to smaller mixture using any method of choice, to give

$$p(h_t|s_t, v_{1:T}) \approx \sum_{j_t} p(j_t|s_t, v_{1:T}) p(h_t|j_t, v_{1:T})$$

The resulting algorithm is presented in 3.**??** which includes using mixtures in both forward and backward passes.

24.2 Relation to other methods

Approximate inference in the SLDS has been a long-standing research topic, generating an extensive literature, to which it is difficult to serve justice. See [52] and [53] for good reviews of previous work. A brief summary of some of the major existing approaches follows.

Assumed Density Filtering : Since the exact filtered estimate $p(h_t|s_t, v_{1:t})$ is an (exponentially large) mixture of Gaussians a useful remedy is to project at each stage of the recursion equation (24.0.3) back to a limited set of K Gaussians. This is a *Gaussian Sum Approximation* [55], and is a form of *Assumed Density Filtering* (ADF) [54]. Similarly, Generalised Pseudo Bayes2 (GPB2) [44, 59] also performs filtering by collapsing to a mixture of Gaussians. This approach to filtering is also taken in [46] which performs the collapse by removing spatially similar Gaussians, thereby retaining diversity.

Several smoothing approaches directly use the results from ADF. The most popular is Kim's method, which updates the filtered posterior weights to form the smoother. As discussed in section (24.1.4), Kim's smoother corresponds to a potentially severe loss of future information and, in general, cannot be expected to improve much on the filtered results from ADF. The more recent work of [46] is similar in spirit to Kim's method, whereby the contribution from the continuous variables is ignored in forming an approximate recursion for the smoothed $p(s_t|v_{1:T})$. The main difference is that for the discrete variables, Kim's method is based on a correction smoother, [43], whereas Lerner's method uses a Belief Propagation style backward pass [9]. Neither method correctly integrates information from the continuous variables. How to form a recursion for a mixture approximation, which does not ignore information coming through the continuous hidden variables is a central contribution of our work.

[47] used a two-filter method in which the dynamics of the chain are reversed. Essentially, this corresponds to a Belief Propagation method which defines a Gaussian sum approximation for $p(v_{t+1:T}|h_t, s_t)$. However, since this is not a density in h_t, s_t, but rather a conditional likelihood, formally one cannot treat this using density propagation methods. In [47], the singularities resulting from incorrectly treating $p(v_{t+1:T}|h_t, s_t)$ as a density are heuristically finessed.

Expectation Propagation : EP [54] corresponds to an approximate implementation of Belief Propagation[12] [9, 41]. Whilst EP may be applied to multiply-connected graphs, it does not fully exploit the numerical advantages present in the singly-connected aSLDS structure. Nevertheless, EP is the most sophisticated rival to Kim's method and EC, since it makes the least assumptions. For this reason, we'll explain briefly how EP works. First, let's simplify the notation, and write the distribution as $p = \prod_t \phi\left(x_{t-1}, v_{t-1}, x_t, v_t\right)$, where $x_t \equiv h_t \otimes s_t$, and $\phi\left(x_{t-1}, v_{t-1}, x_t, v_t\right) \equiv p(x_t|x_{t-1})p(v_t|x_t)$. EP defines 'messages' ρ, λ[13] which contain information from past and future observations respectively[14]. Explicitly, we define $\rho_t(x_t) \propto p(x_t|v_{1:t})$ to represent knowledge about x_t given all information from time 1 to t. Similarly, $\lambda_t(x_t)$ represents knowledge about state x_t given all observations from time T to time $t+1$. In the sequel, we drop the time suffix for notational clarity. We define $\lambda(x_t)$ implicitly through the requirement that the marginal smoothed inference is given by

$$p(x_t|v_{1:T}) \propto \rho\left(x_t\right)\lambda\left(x_t\right) \tag{24.2.1}$$

Hence $\lambda\left(x_t\right) \propto p(v_{t+1:T}|x_t, v_{1:t}) = p(v_{t+1:T}|x_t)$ and represents all future knowledge about $p(x_t|v_{1:T})$. From this

$$p(x_{t-1}, x_t|v_{1:T}) \propto \rho\left(x_{t-1}\right)\phi\left(x_{t-1}, v_{t-1}, x_t, v_t\right)\lambda\left(x_t\right) \tag{24.2.2}$$

Taking the above equation as a starting point, we have

$$p(x_t|v_{1:T}) \propto \int_{x_{t-1}} \rho\left(x_{t-1}\right)\phi\left(x_{t-1}, v_{t-1}, x_t, v_t\right)\lambda\left(x_t\right)$$

Consistency with equation (24.2.1) requires (neglecting irrelevant scalings)

$$\rho\left(x_t\right)\lambda\left(x_t\right) \propto \int_{x_{t-1}} \rho\left(x_{t-1}\right)\phi\left(x_{t-1}, v_{t-1}, x_t, v_t\right)\lambda\left(x_t\right)$$

[12] Non-parametric belief propagation [60], which performs approximate inference in general continuous distributions, is also related to EP applied to the aSLDS, in the sense that the messages cannot be represented easily, and are approximated by mixtures of Gaussians.

[13] These correspond to the α and β messages in the Hidden Markov Model framework [40].

[14] In this Belief Propagation/EP viewpoint, the backward messages, traditionally labeled as β, correspond to conditional likelihoods, and not distributions. In contrast, in the EC approach, which is effectively a so-called $\alpha-\gamma$ recursion, the backward γ messages correspond to posterior distributions.

Similarly, we can integrate equation (24.2.2) over x_t to get the marginal at time x_{t-1} which, by consistency, should be proportional to $\rho\left(x_{t-1}\right)\lambda\left(x_{t-1}\right)$. Hence

$$\rho\left(x_t\right) \propto \frac{\int_{x_{t-1}} \rho\left(x_{t-1}\right)\phi\left(x_{t-1},x_t\right)\lambda\left(x_t\right)}{\lambda\left(x_t\right)}, \qquad \lambda\left(x_{t-1}\right) \propto \frac{\int_{x_t} \rho\left(x_{t-1}\right)\phi\left(x_{t-1},x_t\right)\lambda\left(x_t\right)}{\rho\left(x_{t-1}\right)} \tag{24.2.3}$$

where the divisions can be interpreted as preventing overcounting of messages. In an exact implementation, the common factors in the numerator and denominator cancel. EP addresses the fact that $\lambda(x_t)$ is not a distribution by using equation (24.2.3) to form the projection (or 'collapse'). In the numerator, the terms $\int_{x_{t-1}} \rho\left(x_{t-1}\right)\phi\left(x_{t-1},x_t\right)\lambda\left(x_t\right)$ and $\int_{x_t} \rho\left(x_{t-1}\right)\phi\left(x_{t-1},x_t\right)\lambda\left(x_t\right)$ represent $p(x_t|v_{1:T})$ and $p(x_{t-1}|v_{1:T})$. Since these *are* distributions (an indexed mixture of Gaussians in the SLDS), they may be projected/collapsed to a single indexed Gaussian. The update for the ρ message is then found from division by the λ potential, and vice versa[15]. To perform this division, the potentials in the numerator and denominator are converted to their canonical representations. To form the ρ update, the result of the division is then reconverted back to a moment representation. The collapse is nominally made to a single Gaussian since then explicit division is well defined. The resulting recursions, due to the approximation, are no longer independent and [41] show that using more than a single forward sweep and backward sweep often improves on the quality of the approximation. This coupling is a departure from the exact recursions, which should remain independent, as in our EC approach.

Applied to the SLDS, EP suffers from severe numerical instabilities [41] and finding a way to minimize the corresponding EP free energy in an efficient, robust and guaranteed way remains an open problem. Damping the parameter updates is one suggested approach to heuristically improve convergence.

Variational Methods : [45] used a variational method which approximates the joint distribution $p(h_{1:T}, s_{1:T}|v_{1:T})$ rather than the marginal inference $p(h_t, s_t|v_{1:T})$. This is a disadvantage when compared to other methods that directly approximate the marginal. The variational methods are nevertheless potentially attractive since they are able to exploit structural properties of the distribution, such as a factored discrete state-transition.

Sequential Monte Carlo (Particle Filtering) : These methods form an approximate implementation of equation (24.0.3), using a sum of delta functions to represent the posterior (see, for example, [61]). Whilst potentially powerful, these non-analytic methods typically suffer in high-dimensional hidden spaces since they are often based on naive importance sampling, which restricts their practical use. ADF is generally preferential to Particle Filtering since in ADF the approximation is a mixture of non-trivial distributions, which is better at capturing the variability of the posterior. In addition, for applications where an accurate com-

[15] In EP the explicit division of potentials only makes sense for members of the exponential family. More complex methods could be envisaged in which, rather than an explicit division, the new messages are defined by minimising some measure of divergence between $\rho(x_t)\lambda(x_t)$ and $\int_{x_{t-1}} \rho\left(x_{t-1}\right)\phi\left(x_{t-1},x_t\right)\lambda\left(x_t\right)$, such as the Kullback-Leibler divergence. Whilst this is certainly feasible, it is somewhat unattractive computationally since this would require for each timestep an expensive minimization.

putation of the likelihood of the observations is required (see, for example [62]), the inherent stochastic nature of sampling methods is undesirable.

25 Gaussian Processes

VERY DRAFTY!!!

25.1 The Bayesian approach to Regression

We write our model of the data generating process as

$$y = f(x) \tag{25.1.1}$$

and we aim to choose a function f that fits the data well. We assume that the data output we observe, t has been corrupted with additive Gaussian noise[1]

$$t = f(x) + \eta \tag{25.1.2}$$

where $\eta \sim \text{normal}(0, \sigma^2)$. This means that the *likelihood* of a datapoint t under this model is

$$p(t|f) = \text{normal}(f(x), \sigma^2) \propto e^{-\frac{1}{2\sigma^2}(t-f(x))^2} \tag{25.1.3}$$

Note that it is not necessary to explicitly write normalising constants for distributions since they are uniquely given by the normalisation condition for probabilities. In this case the normalising constant is $1/\left(2\pi\sigma^2\right)^{1/2}$.

Assuming that individual datapoints are independently and identically distributed, the likelihood for a vector of observed data outputs $\mathbf{t} = (t^1 \dots t^P)$ is[2]

$$p(\mathbf{t}|f) = \prod_{i=1}^{P} p(t_i|f) \propto e^{-\frac{1}{2\sigma^2}(\mathbf{t}-\mathbf{f})^2} \tag{25.1.4}$$

where $\mathbf{f} = (f(x^1) \dots f(x^P))$. The term $(\mathbf{t} - \mathbf{f})^2$ in the exponent is referred to in neural networks as the training error, so that maximum likelihood fitting in this context is equivalent to finding that function f which minimizes the training error. However, we may have some extra beliefs about the model f which we can express through a *prior* distribution $p(f)$. For example, certain f may not be smooth enough, and we would believe them to be unlikely. We may prefer to express these beliefs by assigning a prior on the parameters θ.

Together, the likelihood and prior on the function f complete the specification of the model. We can then use Bayes rule to find the *posterior* distribution of f in light of the observed data D.

$$P(f|\mathbf{t}) = \frac{p(\mathbf{t}|f)p(f)}{p(\mathbf{t})} \tag{25.1.5}$$

[1] Although not necessary, this assumption is convenient so that the following theory is analytically tractable.

[2] In equation (25.1.4) we use the notation $\mathbf{v}^2$ to mean $\sum_i v_i^2$ for an arbitrary vector $\mathbf{v}$.

25.1.1 Parameterised Models

It may be that our function f is parameterised in a certain way,

$$y = f(x|\theta) \tag{25.1.6}$$

Where θ represents all the parameters that we could adjust to make the function f fit the data well. For example, $f(x|\theta) = \theta_1 x + \theta_2$ would parameterize straight line models. In this case, it is more natural to think of the likelihood $p(\mathbf{t}|\theta)$ and the prior $p(\theta)$ as functions of these parameters.

25.1.2 Making Predictions

In contrast to other formalisms, the Bayesian approach automatically gives rise to an ensemble of models, namely the posterior distribution $p(f|\mathbf{t})$. Assuming a squared error model, the best estimate of the model is given by the posterior average of the functions[3]

$$\langle f(x) \rangle \equiv \int f(x) p(f|\mathbf{t}) df \tag{25.1.7}$$

Similarly, we can calculate error bars for the predictions,

$$\mathrm{var}(f(x)) \equiv \int \left[f(x) - \langle f(x) \rangle \right]^2 p(f|\mathbf{t}) df \tag{25.1.8}$$

which gives a measure of the confidence in the prediction $\langle f(x) \rangle$. Note that if we wish to make *predictive error bars*, we need to include possible noise corrupting processes. For example, in the case of additive Gaussian noise, we believe that the actual data points we observe are modelled by the process $f(x) + \eta$, where $\eta \sim \mathrm{normal}(0, \sigma^2)$. Given that the posterior distribution is independent of the noise process, this means that predicitve error bars are given by simply adding σ^2 to equation (25.1.8).

In principle, in contrast to non-Bayesian approaches, there is no need to set aside a portion of the data to test the fidelity of the model. Nevertheless, one may wish to monitor a quantity such as the test error to check if f is capable of modeling the data well enough to produce the kind of test error performance we might hope for.

25.1.3 Model Selection

In the Bayesian framework, the quantity that we need in order to assess the fidelity of a model M is it's likelihood, $p(\mathbf{t}|M)$. This is also sometimes called the "evidence". It will be simpler to explain how to use such quantities in the context of a specific model, and we defer this discussion till section (25.2.4)

25.2 Generalised Linear Models

Generalised linear models have the form

$$y^l = \sum_{i=1}^{k} w_i \phi_i(x^l) \tag{25.2.1}$$

[3] This is readily shown by considering $\left\langle (\mathbf{f} - \mathbf{t})^2 \right\rangle = \left\langle (\mathbf{f} - \langle \mathbf{f} \rangle + \langle \mathbf{f} \rangle - \mathbf{t})^2 \right\rangle = \left\langle (\mathbf{f} - \langle \mathbf{f} \rangle)^2 \right\rangle + (\langle \mathbf{f} \rangle - \mathbf{t})^2$, which has a minimum at $\mathbf{f} = \langle \mathbf{f} \rangle$.

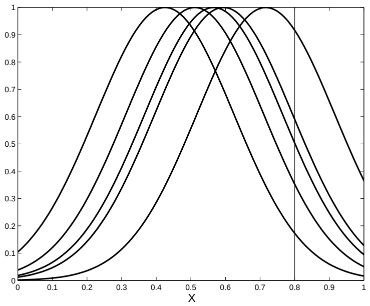


Figure 25.1: A set of 5 Gaussian basis functions. The model output for a particular x is given by a linear combination of the basis function values at that x, here given by the intersection of the basis curves with the line $x = 0.8$.

That is, the model is linear in the parameters w, although the output y depends non-linearly on the input x. If we have several points that we wish to make a prediction at, $x^1 \dots x^l$, then we write the prediction vector $\boldsymbol{(y^1 \dots y^l)^T = y = \Phi w}$ where we have defined the *design matrix* $\mathbf{\Phi}_{ji} = \phi_i(x^j)$. Note that an upper index refers to the datapoint number and the lower to a basis component. In fig(25.1) we plot 5 Gaussian basis functions. The models value for say $x = 0.8$ is then a linear combination of the basis function values at that input (given here by the intersection of the vertical line with $x = 0.8$).

We shall assume that, in addition to w, there may be some tunable (hyper)parameters, such as the width of the basis functions and our belief about the noise level.

25.2.1 Understanding the Prior

The **Gaussian** basis functions have the form $\phi(x) = \exp\left(-0.5\,(x-c)^2/(2\lambda^2)\right)$ which are local in the sense that they decay (quickly) to zero as we move away from the centre c.

The **Log** basis functions have a non-local form, extending to infinity at infinity: $\phi(x) = \exp\left(-0.5\,(x-c)^2/\lambda^2\right) + (x-c)^2\log\left((x-c)^2 + 0.0001\right)$. However, they retain some "bump" Gaussian shape close to the centre c. Such non-local forms can be useful if our beliefs about the function only apply in a limited region. Non-local basis functions (as we shall see) typically give rise to large error bars away from the training data, and do not necessarily affect the quality of predictions close to the training data.

One needs to cover the input region of interest sufficiently well so that the functional form of the output distribution $p(f)$ is expressive enough to capture the kinds of models that we are interested in finding.

25.2.2 Sampling the function space prior

We assume that our prior belief about the function f can be expressed by a belief about the distribution of the parameters w. Here we assume that this takes the form of a Gaussian distribution, with zero mean, and a user specified variance:

$$\boldsymbol{p(w) = \mathbf{normal}(0, \alpha^{-1}\mathbf{I}) \propto \exp\left(-\frac{\alpha}{2}w^2\right)} \tag{25.2.2}$$

Using this distribution, we can draw (say 6) random weight vectors $w_1 \dots w_6$ and plot the corresponding functions $w_1 \cdot \boldsymbol{\phi} \dots w_6 \cdot \boldsymbol{\phi}$, where $\boldsymbol{\phi} = \{\phi_1(x) \dots \phi_k(x)\}$ is a vector of basis function values.

25.2.3 Understanding the Posterior

The posterior for the weights of the GLM is

$$\boldsymbol{p(w|\mathbf{t}) \propto p(\mathbf{t}|w)p(w) \propto \exp -\frac{1}{2}\left(\beta\left(\Phi w - \mathbf{t}\right)^2 + \alpha w^2\right)} \tag{25.2.3}$$

For convenience we define $\beta = 1/\sigma^2$, the reciprocal of our noise belief. It is not difficult to see that $p(w|\mathbf{t})$ is a Gaussian distribution for the weights (since the exponent is quadratic in the weights). In fact it is the distribution

$$\boldsymbol{p(w|\mathbf{t}) = \mathbf{normal}(\beta C^{-1}\Phi^T\mathbf{t}, C^{-1})} \tag{25.2.4}$$

where $C = \alpha\mathrm{I} + \beta\boldsymbol{\Phi}^T\boldsymbol{\Phi}$. That is, $p(w|\mathbf{t})$ is a multivariate Gaussian distribution with mean $\langle w\rangle = \beta C^{-1}\boldsymbol{\Phi}^T\mathbf{t}$ and covariance matrix $\left\langle(w - \langle w\rangle)(w - \langle w\rangle)^T\right\rangle = C^{-1}$. For the linear model, the mean predictions over the posterior are simply given by

$$\langle f(x)\rangle = \langle w\rangle^T \boldsymbol{\phi}(x) \tag{25.2.5}$$

Similarly, the variance is given by

$$\mathrm{var}(f(x)) = \boldsymbol{\phi}(x)^T C^{-1}\boldsymbol{\phi}(x) \tag{25.2.6}$$

The predictions becoming more confident towards the edges in the Gaussian basis function case is simply a consequence of the form of the basis functions. This is an important point - you only get out from the method answers consistent with your model assumptions.

What is the posterior distribution $p(y(x^1) \dots y(x^l)|\mathbf{t})$ for a set of chosen $x^1 \dots x^l$?, induced by the Gaussian weight posterior equation (25.2.4)?

25.2.4 Understanding Model Selection Issues

As we have seen, the distribution of the weights w of the GLM are determined automatically through the Bayesian procedure. The only parameters that are left to the user to control are the width of the basis functions, the noise belief, the scale α and the number and type of the basis functions. Let's denote such parameters by Γ. It may be that we would like to carry out a Bayesian analysis for these parameters too, so that we can assess the relevance of different model parameter settings in light of the observed data.

In principle, this can be viewed as just another level in a hierarchy of models. The determined Bayesian would assign a (hyper)prior to these parameters $p(\Gamma)$ and perform model averaging over them (just as we did in the weights w case),

$$\boldsymbol{\langle f(x)\rangle = \int f(x|w)p(w,\Gamma|\mathbf{t})dwd\Gamma = \int\left\{\int f(x|w)p(w|\Gamma,\mathbf{t})dw\right\}p(\Gamma|\mathbf{t})d\Gamma}$$

(25.2.7)

Where $p(\Gamma|\mathbf{t}) = p(\mathbf{t}|\Gamma)p(\Gamma)/p(\mathbf{t})$ and $p(\Gamma)$ is our prior belief about the (hyper)parameters. The "evidence" $p(\mathbf{t}|\Gamma)$ is obtained by integrating over the weights, $p(\mathbf{t}|\Gamma) = \int p(\mathbf{t}|w)p(w|\Gamma)dw$. Typically, the integrations in equation (25.2.7) are extremely difficult to carry out (even if $p(\Gamma|\mathbf{t})$ is tractable) and one needs to resort to techniques such as Monte Carlo.

A simpler alternative is to consider using those Γ that correspond to a maximum of the model posterior $p(\Gamma|\mathbf{t})$. Provided that the posterior $p(\Gamma|\mathbf{t})$ is sharply peaked around it's optimum value, this may still give a faithful value for the average in equation (25.2.7). Assuming a flat prior on Γ, this corresponds to using the Γ that maximize the likelihood $p(\mathbf{t}|\Gamma)$.

In the linear case here, and with the Gaussian noise model assumption, calculating the model likelihood involves only Gaussian integrals, giving

$$\log p(\mathbf{t}|\Gamma) = -\frac{\beta}{2}\mathbf{t}^2 + \frac{1}{2}\beta^2 \mathbf{t}^T \mathbf{\Phi}^T C^{-1} \mathbf{\Phi}\mathbf{t} - \frac{1}{2}\log\det(C) + \frac{k}{2}\log\alpha - \frac{P}{2}\log(2\pi/\beta) \tag{25.2.8}$$

GLMs can be very flexible regression models and one advantage from the Bayesian point of view is that the model likelihood $p(\mathbf{t}|\Gamma)$ can be calculated exactly. This makes combining models which have say different numbers of basis functions easy to do – we just use equation (25.2.7).

25.3 Gaussian Processes

Gaussian Processes are (in a certain sense) formally identical to GLMs, but differ in their computational implementation. The main idea is to go back to the general Bayesian framework section (25.1), using the Gaussian noise model, but now to specify the form of the prior on the functions directly.

25.3.1 Specifying the Prior

From equation (25.1.4) we see that, since we already have a (Gaussian) definition for the likelihood $p(f|\mathbf{t})$, all we need to do is specify a prior distribution $p(f)$ on the function space f to complete the model specification. The most natural choice is to specify a Gaussian distribution here, since that will mean that the posterior is also Gaussian.

Imagine that we are given a set of inputs $x^1 \ldots x^l$. Consider a particular x^i and it's corresponding possible function value y^i. If we have a space of possible functions, then they will pass through different y^i for the same x^i (see say x^1 in fig(25.2)). Indeed, we can *construct* the prior on functions so that the distribution of these values should be Gaussian, centered around some mean value (we will take this to be zero for simplicity) with a certain variance.

Consider now two inputs, x^i and x^j and their separation, $|x^i - x^j|$. Note that y^i and y^j fluctuate as different functions are sampled from some function space prior. How can we incorporate ideas of smoothness? If $|x^i - x^j|$ is small, we may expect that a set of values at y^i and a set at y^j should be highly correlated (as in fig(25.2) for x^1 and x^2) . This means that we might well think that the output values y^i and

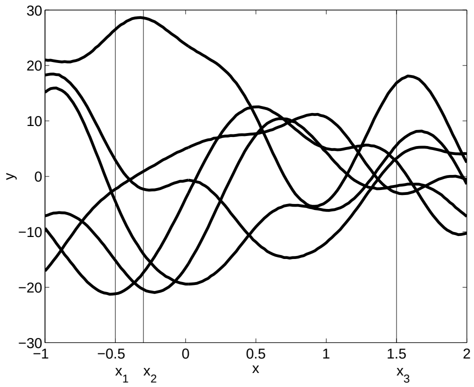


Figure 25.2: Samples functions from a Gaussian Process Prior. The correlation between y^i and y^j decreases with the distance $|x^i - x^j|$.

y^j should be highly correlated if $|x^i - x^j|$ is small. Conversely, if $|x^i - x^j|$ is large, we (probably) do not expect that y^i and y^j will be at all correlated (as for values at x^1 and x^3 in fig(25.2)). We can express these beliefs about the correlations of the components of a vector of values $y = (y^1 \dots y^l)$ at inputs $x^1 \dots x^l$ as a multivariate Gaussian distribution

$$\boldsymbol{p(y) \propto \exp\left(-\frac{1}{2} y^T \mathrm{K}^{-1} y\right)} \tag{25.3.1}$$

where K is the covariance matrix of the outputs y. The elements of K are specified by the covariance function $c(x_i, x_j)$. As we argued above, we might expect that the correlation between y_i and y_j decreases the further apart x_i and x_j are.

In the **Gaussian** case, the covariance function is $c(x_i, x_j) = \alpha \exp\left(-0.5\lambda\left(x_i - x_j\right)^2\right)$. Note that the shape of this function is smooth at zero.

In the **Ornstein Uhlenbeck** case, the covariance function is $c(x_i, x_j) = \alpha \exp\left(-0.5\lambda|x_i - x_j|\right)$.

Note how the Ornstein Uhlenbeck process gives rise to much less smooth functions than those formed with the Gaussian covariance function.

□ What is the relationship between the derivative of the covariance function at the origin and the smoothness of the function space?

Changing the length scale of the covariance function affects the range over which the functions are correlated. See how changing α alters the scale of the outputs.

25.3.2 Making Predictions

Imagine that we have some new inputs x^* and we wish to make some predictions for their outputs y^*. According to the Bayesian philosophy, we need to specify a likelihood and a prior. We already specified a prior in section (25.3.1),

$$p(\mathbf{y}^*, y) = \text{normal}(\mathbf{0}, K) \tag{25.3.2}$$

where K can be partitioned into matrices $\text{K}, K_{x^*x^*}, K_{xx^*}, K_{xx^*}^T$. K has elements $c(x^i, x^j)$, and K_{xx^*} elements $c(x^i, x^*)$ *etc.*. The likelihood is

$$\boldsymbol{p(\mathbf{t}|\mathbf{y}^*, y) = p(\mathbf{t}|y) = \textbf{normal}(y, \sigma^2\mathbf{I})} \tag{25.3.3}$$

Since the prior and likelihood are Gaussian, it is clear that the posterior $p(\mathbf{y}^*, y|\mathbf{t}) \propto p(\mathbf{t}|\mathbf{y}^*, y)p(\mathbf{y}^*, y)$ is also Gaussian in $\mathbf{y}^*, y$. The marginal distribution $p(\mathbf{y}^*|\mathbf{t})$ is therefore also Gaussian. You might like to convince yourselves in your own time that it takes the form

$$p(\mathbf{y}^*|\mathbf{t}) = \text{normal}(K_{xx^*}\left(\text{K} + \sigma^2\right)^{-1}\mathbf{t}, K_{x^*x^*} - K_{xx^*}^T\left(\text{K} + \sigma^2\text{I}\right)^{-1}K_{xx^*}) \tag{25.3.4}$$

First we see predictions for one training point. The red curve is the mean prediction and the green curve are the error bars (one standard deviation). The blue crosses are the training data points.

We can now try to understand the posterior as in the case of GLMs. In the same way, we alter the **noise belief** and **actual noise** and see what happens to the predictions.

Note how the error bars collapse onto the data for a single datapoint. See how this also happens for two datapoints as well.

Can you observe any differences between the GP predictions and the GLM predictions?
What do you think could be the connection between GPs and GLMs?

25.3.3 Model Selection

It is straightforward to show that the likelihood for a GP model is

$$\log p(\mathbf{t}|\Gamma) = \frac{1}{2}\log(\det(\text{K} + \sigma^2)) - \frac{1}{2}\mathbf{t}^T(\text{K} + \sigma^2\text{I})\mathbf{t} - \frac{1}{2}P\log(2\pi) \tag{25.3.5}$$

25.3.4 Classification problems

There are two basic methods for making predictions in classification problems (see, e.g Ripley, 1996); (i) the *sampling paradigm*, where a class-conditional density $p(x|k)$ and a prior are created for each class k, and Bayes' theorem is used to determine $p(k|x)$ given a new input x, or (ii) the *diagnostic paradigm*, where the aim is to predict $p(k|x)$ directly via some function of the input. As $p(k|x)$ must lie in $[0, 1]$, this condition is usually achieved by using an output (or transfer) function which enforces the constraint. For the two class problem a common choice is the logistic function $\sigma(y) = 1/(1 + e^{-y})$. For a $k > 2$ class problem a simple generalization of the logistic function, the softmax function, is frequently used.

We will follow the diagnostic paradigm and use the logistic function, an approach also used widely in the neural networks literature. In the simplest method of this kind, logistic regression, the input to the sigmoid function y is simply computed as a linear combination of the inputs, plus a bias, i.e. $\boldsymbol{y = w^T x + b}$. Neural networks and other flexible methods allow y to be a non-linear function of the inputs.

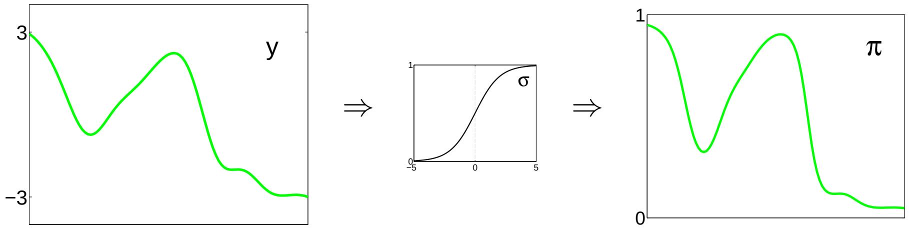


Figure 25.3: $\pi(x)$ is obtained from $y(x)$ by "squashing" it through the sigmoid function σ.

25.4 Gaussian Processes for Classification

By using the logistic transfer function σ to produce an output which can be interpreted as $\pi(x)$, the probability of the input x belonging to class 1, the job of specifying a prior over functions π can be transformed into that of specifying a prior over the input to the transfer function, which we shall call the *activation*, and denote by y, with $\pi(x) = \sigma(y(x))$ (see figure 25.3). For input x_i, we will denote the corresponding probability and activation by π_i and y_i respectively.

Given that the GP contains adjustable hyperparameters, how should they be adapted given the data ? Maximum likelihood or (generalized) cross-validation methods are often used, but we will prefer a Bayesian solution. A prior distribution over the hyperparameters $P(\boldsymbol{\theta})$ is modified using the training data to obtain the posterior distribution $P(\boldsymbol{\theta}|\mathbf{t}) \propto P(\mathbf{t}|\boldsymbol{\theta})P(\boldsymbol{\theta})$.

To make predictions we then integrate over the posterior; for example, the mean value $\overline{\pi}(x_*)$ for test input x_* is given by

$$\overline{\pi}(x_*) = \int \left\{ \int \pi(\mathbf{x}_*) P(\pi(x_*)|\mathbf{t}, \boldsymbol{\theta}) d\pi \right\} P(\boldsymbol{\theta}|\mathbf{t}) d\boldsymbol{\theta}. \tag{25.4.1}$$

We show in section 25.4.3 how to perform the integral in 25.4.1 over the hyperparameters $P(\boldsymbol{\theta}|\mathbf{t})$. Here we consider the hyperparameters to be fixed, and are interested in the posterior distribution $P(\pi_*|\mathbf{t}) = P(\pi(x_*)|\mathbf{t})$ for a new input x_*. This can be calculated by finding the distribution $P(y_*|\mathbf{t})$ (y_* is the activation of π_*) and then using the appropriate Jacobian to transform the distribution. Formally the equations for obtaining $P(y_*|\mathbf{t})$ are identical to equation **??**. However, even if we use a GP prior so that $P(y_*, y)$ is Gaussian, the usual expression for $P(\mathbf{t}|y) = \prod_i \pi_i^{t_i}(1-\pi_i)^{1-t_i}$ for classification data (where the t's take on values of 0 or 1), means that the average over π in equation 25.4.1 is no longer exactly analytically tractable.

After transforming equation 25.4.1 to an integral over activations, we will employ Laplace's approximation, i.e. we shall approximate the integrand $P(y_*, y|\mathbf{t})$ by a Gaussian distribution centred at a maximum of this function with respect to y_*, y with an inverse covariance matrix given by $-\nabla\nabla \log P(y_*, y|\mathbf{t})$. The necessary integrations (marginalization) can then be carried out analytically (see, e.g. Green and Silverman (1994) §5.3) and, we provide a derivation in the following section. The averages over the hyperparameters will be carried out using Monte Carlo techniques, which we describe in section 25.4.3.

25.4.1 Maximizing $P(y_*, y|\mathbf{t})$

Let y_+ denote (y_*, y), the complete set of activations. By Bayes' theorem $\log P(y_+|\mathbf{t}) = \log P(\mathbf{t}|y) + \log P(y_+) - \log P(\mathbf{t})$, and let $\Psi_+ = \log P(\mathbf{t}|y) + \log P(y_+)$. As $P(\mathbf{t})$ does not depend on y_+ (it is just a normalizing factor), the maximum of $P(y_+|\mathbf{t})$ is found by maximizing Ψ_+ with respect to y_+. We define Ψ similarly in relation to $P(y|\mathbf{t})$. Using $\log P(t_i|y_i) = t_i y_i - \log(1 + e^{y_i})$, we obtain

$$\Psi_+ = \mathbf{t}^T y - \sum_{i=1}^{n} \log(1 + e^{y_i}) - \frac{1}{2} y_+^T K_+^{-1} y_+ - \frac{1}{2} \log |K_+| - \frac{n+1}{2} \log 2\pi \qquad (25.4.2)$$

$$\Psi = \mathbf{t}^T y - \sum_{i=1}^{n} \log(1 + e^{y_i}) - \frac{1}{2} y^T K^{-1} y - \frac{1}{2} \log |K| - \frac{n}{2} \log 2\pi \qquad (25.4.3)$$

where K_+ is the covariance matrix of the GP evaluated at $x_1, \ldots x_n, x_*$. K_+ can be partitioned in terms of an $n \times n$ matrix K, a $n \times 1$ vector $\boldsymbol{k}$ and a scalar k_*, viz.

$$K_+ = \begin{pmatrix} K & \boldsymbol{k} \\ \boldsymbol{k}^T & k_* \end{pmatrix} \qquad (25.4.4)$$

As y_* only enters into equation 25.4.2 in the quadratic prior term and has no data point associated with it, maximizing Ψ_+ with respect to y_+ can be achieved by first maximizing Ψ with respect to y and then doing the further quadratic optimization to determine y_*. To find a maximum of Ψ we use the Newton-Raphson (or Fisher scoring) iteration $y^{new} = y - (\nabla\nabla\Psi)^{-1}\nabla\Psi$. Differentiating equation 25.4.3 with respect to y we find

$$\nabla\Psi = (\mathbf{t} - \boldsymbol{\pi}) - K^{-1} y \qquad (25.4.5)$$

$$\nabla\nabla\Psi = -K^{-1} - N \qquad (25.4.6)$$

where the 'noise' matrix is given by $N = \mathrm{diag}(\pi_1(1-\pi_1), .., \pi_n(1-\pi_n))$. This results in the iterative equation,

$$\boldsymbol{y^{new} = (K^{-1} + N)^{-1} N (y + N^{-1}(\mathbf{t} - \pi))} \qquad (25.4.7)$$

To avoid unnecessary inversions, it is usually more convenient to rewrite this in the form

$$\boldsymbol{y^{new} = K(I + KN)^{-1}(Ny + (\mathbf{t} - \pi))} \qquad (25.4.8)$$

Given a converged solution $\tilde{y}$ for y, y_* can easily be found using $y_* = \boldsymbol{k}^T K^{-1} \tilde{y} = \boldsymbol{k}^T(\mathbf{t} - \tilde{\boldsymbol{\pi}})$. $var(y_*)$ is given by $(K_+^{-1} + N_+)^{-1}_{(n+1)(n+1)}$, where N_+ is the N matrix with a zero appended in the $(n+1)$th diagonal position. Given the mean and variance of y_* it is then easy to find $\hat{\pi}_* = \int \pi_* P(\pi_*|\mathbf{t}) d\pi_*$, the mean of the distribution of $P(\pi_*|\mathbf{t})$(see equation 25.4.1). In order to calculate the Gaussian integral over the logistic sigmoid function, we employ an approximation based on the expansion of the sigmoid function in terms of the error function. As the Gaussian integral of an error function is another error function, this approximation is fast to compute. Specifically, we use a basis set of 5 scaled error functions to interpolate the logistic sigmoid at chosen points. This gives an accurate approximation (to 10^{-4}) to the desired integral with a small computational cost.

The justification of Laplace's approximation in our case is somewhat different from the argument usually put forward, e.g. for asymptotic normality of the maximum likelihood estimator for a model with a finite number of parameters. This is because the dimension of the problem grows with the number of data points. However, if we consider the "infill asymptotics", where the number of data points in a *bounded* region increases, then a local average of the training data at any point x will provide a tightly localized estimate for $\pi(x)$ and hence $y(x)$, so we would expect the distribution $P(y)$ to become more Gaussian with increasing data.

25.4.2 Parameterizing the covariance function

There are many reasonable choices for the covariance function. Formally, we are required to specify functions which will generate a non-negative definite covariance matrix for any set of points $(x_1, \ldots, x_k)$. From a modelling point of view we wish to specify covariances so that points with nearby inputs will give rise to similar predictions. We find that the following covariance function works well:

$$C(x, x') = v_0 \exp\{-\frac{1}{2}\sum_{l=1}^{d} w_l (x_l - x'_l)^2\} \tag{25.4.9}$$

where x_l is the lth component of x and $\boldsymbol{\theta} = \log(v_0, w_1, \ldots, w_d)$ plays the role of hyperparameters[4]. We define the hyperparameters to be the log of the variables in equation (25.4.9) since these are positive scale-parameters. This covariance function can be obtained from a network of Gaussian radial basis functions in the limit of an infinite number of hidden units (Williams, 1996).

The w_l parameters in equation 25.4.9 allow a different length scale on each input dimension. For irrelevant inputs, the corresponding w_l will become small, and the model will ignore that input. This is closely related to the Automatic Relevance Determination (ARD) idea of MacKay and Neal (Neal, 1995). The v_0 variable gives the overall scale of the prior; in the classification case, this specifies if the π values will typically be pushed to 0 or 1, or will hover around 0.5.

25.4.3 Integration over the hyperparameters

To make predictions we integrate the predicted probabilities over the posterior $P(\boldsymbol{\theta}|\mathbf{t}) \propto P(\mathbf{t}|\boldsymbol{\theta})P(\theta)$, as given by equation 25.4.1. For the regression problem $P(\mathbf{t}|\boldsymbol{\theta})$ can be calculated exactly using $\boldsymbol{P(\mathbf{t}|\theta) = \int P(\mathbf{t}|y)P(y|\theta)dy}$, but this integral is not analytically tractable for the classification problem. Again we use Laplace's approximation and obtain

$$\log P(\mathbf{t}|\boldsymbol{\theta}) \simeq \Psi(\tilde{y}) + \frac{1}{2}|K^{-1} + N| + \frac{n}{2}\log 2\pi \tag{25.4.10}$$

where $\tilde{y}$ is the converged iterate of equation 25.5.1, and we denote the right-hand side of this equation by $\log P_a(\mathbf{t}|\boldsymbol{\theta})$ (where a stands for approximate).

The integration over $\boldsymbol{\theta}$-space also cannot be done analytically, and we employ a Markov Chain Monte Carlo method. We have used the Hybrid Monte Carlo (HMC) method of Duane *et al* (1987), with broad Gaussian hyperpriors on the parameters.

HMC works by creating a fictitious dynamical system in which the hyperparameters are regarded as position variables, and augmenting these with momentum

[4] We call $\boldsymbol{\theta}$ the hyperparameters rather than parameters as they correspond closely to hyperparameters in neural networks; in effect the weight parameters have been integrated out exactly.

variables $\boldsymbol{p}$. The purpose of the dynamical system is to give the hyperparameters "inertia" so that random-walk behaviour in $\boldsymbol{\theta}$-space can be avoided. The total energy, H, of the system is the sum of the kinetic energy, $K = \boldsymbol{p}^T\boldsymbol{p}/2$ and the potential energy, E. The potential energy is defined such that $p(\boldsymbol{\theta}|D) \propto \exp(-E)$, i.e. $E = -\log P(\mathbf{t}|\boldsymbol{\theta}) - \log P(\boldsymbol{\theta})$. In practice $\log P_a(\mathbf{t}|\boldsymbol{\theta})$ is used instead of $\log P(\mathbf{t}|\boldsymbol{\theta})$. We sample from the joint distribution for $\boldsymbol{\theta}$ and $\boldsymbol{p}$ given by $P(\boldsymbol{\theta}, \boldsymbol{p}) \propto \exp(-E-K)$; the marginal of this distribution for $\boldsymbol{\theta}$ is the required posterior. A sample of hyperparameters from the posterior can therefore be obtained by simply ignoring the momenta.

Sampling from the joint distribution is achieved by two steps: (i) finding new points in phase space with near-identical energies H by simulating the dynamical system using a discretised approximation to Hamiltonian dynamics, and (ii) changing the energy H by Gibbs sampling the momentum variables.

Hamilton's first order differential equations for H are approximated using the leapfrog method which requires the derivatives of E with respect to $\boldsymbol{\theta}$. Given a Gaussian prior on $\boldsymbol{\theta}$, $\log P(\boldsymbol{\theta})$ is straightforward to differentiate. The derivative of $\log P_a(\boldsymbol{\theta})$ is also straightforward, although implicit dependencies of $\tilde{\boldsymbol{y}}$ (and hence $\tilde{\boldsymbol{\pi}}$) on $\boldsymbol{\theta}$ must be taken into account by using equation 25.4.5 at the maximum point to obtain, $\tilde{\boldsymbol{y}}' = I + KN^{-1}K'(\mathbf{t} - \boldsymbol{\pi})$. The calculation of the energy can be quite expensive as for each new $\boldsymbol{\theta}$, we need to perform the maximization required for Laplace's approximation, equation 25.4.10. The Newton-Raphson was initialized each time with $\boldsymbol{\pi} = 0.5$, and iterated until the mean relative difference of the elements of N between consecutive iterations was less than 10^{-4}.

The same step size ε is used for all hyperparameters, and should be as large as possible while keeping the rejection rate low. We have used a trajectory made up of $L = 20$ leapfrog steps, which gave a low correlation between successive states[5]. This proposed state is then accepted or rejected using the Metropolis rule depending on the final energy H^* (which is not necessarily equal to the initial energy H because of the discretization).

The priors over hyperparameters were set to be Gaussian with a mean of -3 and a standard deviation of 3. In all our simulations a step size $\varepsilon = 0.1$ produced a very low rejection rate ($< 5\%$). The hyperparameters corresponding to the w_l's were initialized to -2 and that for v_0 to 0. The sampling procedure was run for 200 iterations, and the first third of the run was discarded; this "burn-in" is intended to give the hyperparameters time to come close to their equilibrium distribution.

25.5 Multiple classes

The extension of the preceding framework to multiple classes is essentially straightforward, although notationally more complex.

We shall throughout employ a one-of-m class coding scheme[6], and use the multi-class analogue of the logistic function - the softmax function - to describe the class probabilities. The probability that an instance labelled by n is in class m is denoted by π_m^n, so that an upper index to denotes the example number, and a lower index the class label. Similarly, the activations associated with the probabilities

[5] In our experiments $\boldsymbol{\theta}$ is only 7 or 8 dimensional, so the trajectory length needed is much shorter than that for neural network HMC implementations.

[6] That is, the class is represented by a vector of length m with zero entries everywhere except for the mth component which contains 1

are denoted by y_m^n. Formally, the softmax link function relates the activations and probabilities through

$$\pi_m^n = \frac{\exp y_m^n}{\Sigma_{m'} \exp y_{m'}^n}$$

which automatically enforces the constraint $\sum_m \pi_m^n = 1$. The targets are similarly represented by t_m^n, which are specified using a one-of-m coding.

The log likelihood takes the form $\mathcal{L} = \sum_{n,m} t_m^n \ln \pi_m^n$, which for the softmax link function gives

$$\mathcal{L} = \sum_{n,m} t_m^n \left(y_m^n - \ln \sum_{m'} \exp \pi_{m'}^n \right) \tag{25.5.1}$$

As for the two class case, we shall assume that the GP prior operates in activation space; that is we specify the correlations between the activations y_m^n.

One important assumption we make is that our prior knowledge is restricted to correlations between the activations of a particular class. Whilst there is no difficulty in extending the framework to include inter-class correlations, we have not yet encountered a situation where we felt able to specify such correlations. Formally, the activation correlations take the form,

$$\langle y_m^n y_{m'}^{n'} \rangle = \delta_{m,m'} K_m^{n,n'} \tag{25.5.2}$$

where $K_m^{n,n'}$ is the n, n' element of the covariance matrix for the mth class. Each individual correlation matrix K_i has the form given by equation 25.4.9 for the two class case). We shall make use of the same intraclass correlation structure as that given in equation 25.4.9 with a separate set of hyperparameters for each class.

For simplicity, we introduce the augmented vector notation,

$$y_+ = \left(y_1^1, ...y_1^n, y_1^*, y_2^1, ...y_2^n, y_2^*, ...y_m^1, ...y_m^n, y_m^* \right)$$

where, as in the two class case, y_i^* denotes the target activation for class i; this notation is also used to define $\mathbf{t}_+$ and $\boldsymbol{\pi}_+$. In a similar manner, we define y, $\mathbf{t}$ and $\boldsymbol{\pi}$ by excluding the corresponding target values, denoted by a '*' index.

With this definition of the augmented vectors, the GP prior takes the form,

$$\boldsymbol{P(y_+) \propto \exp \left\{ -\frac{1}{2} y_+^T K^+ y_+ \right\}} \tag{25.5.3}$$

where, from equation 25.5.2, the covariance matrix K^+ is block diagonal in the matrices, $K_1^+, ..., K_m^+$. Each individual matrix K_i^+ expresses the correlations of activations within class i, with covariance function given by equation 25.4.9, as for the two class case.

25.5.1 Finding the mode of the distribution

The GP prior and likelihood, defined by equations 25.5.3, 25.5.1 respectively, define the posterior distribution of activations, $P(y_+|\mathbf{t})$. Again, as in section 25.4.1 we are interested in a Laplace approximation to this posterior, and therefore need to find the mode with respect to y_+. Dropping unnecessary constants, the multi-class analogue of equation 25.4.2 for terms involving y_+ in the exponent of the posterior are:

$$\Psi_+ = -\frac{1}{2} y_+^T K_+^{-1} y_+ + \mathbf{t}^T y - \sum_n \ln \sum_n \exp y_m^n$$

By the same principle as in section 25.4.1, we define Ψ by analogy with equation 25.4.3, and first optimize Ψ with respect to y, afterwards performing the quadratic optimization of Ψ_+ with respect to y_*.

In order to optimize Ψ with respect to y_+, we make use of the Hessian given by,

$$\nabla\nabla\Psi = -K^{-1} - N$$

Although this is in the same form as for the two class case, equation (25.4.6), there is a slight change in the definition of the 'noise' matrix, N. A convenient way to define N is by introducing the matrix Π which is an $(m * n_+) \times (n_+)$ matrix of the form $\Pi = (diag(\pi_1^1..\pi_1^{n_+}), .., diag(\pi_m^1..\pi_m^{n_+}))$. Using this notation, we can write the noise matrix in the form of a diagonal matrix and an outer product,

$$N = -diag(\pi_1^1..\pi_1^{n_+}, .., \pi_m^1..\pi_m^{n_+}) + \Pi\Pi^{\mathsf{T}} \tag{25.5.4}$$

The update equation for iterative optimization of Ψ with respect to the activation y then follow the same form as that given by equation . The advantage of the representation of the noise matrix in equation 25.5.4 is that we can then invert matrices and find their determinants using the identities,

$$(A + HH^{\mathsf{T}})^{-1} = A^{-1} - A^{-1}H\left(I + H^{\mathsf{T}}A^{-1}H\right)^{-1}H^{\mathsf{T}}A^{-1} \tag{25.5.5}$$

and

$$\det(A + HH^{\mathsf{T}}) = \det(A)\det(I + H^{\mathsf{T}}A^{-1}H) \tag{25.5.6}$$

where $A = K + diag(\pi_1^1..\pi_1^{n_+})$. Thus, rather than requiring determinants and inverses of $(m*n_+) \times (m*n_+)$ matrices, we only need to carry out expensive matrix computations on $(n_+) \times (n_+)$ matrices. Some care must be taken in manipulating the noise matrix N as this is singular due to the linear constraint imposed upon its elements by $\sum_m \pi_m^n = 1$. The resulting update equations for y are then of the same form as given in equation 25.5.1, where the noise matrix and covariance matrices are now in their multiple class form.

Essentially, these are all the results needed to generalise to the multiple class method. Although, as we mentioned above, the time complexity of the problem does not scale with the m^3, but rather m (due to the identities (25.5.5,25.5.6)), calculating the function and its gradient is still rather expensive. We experimented with several methods of mode finding for the Laplace approximation. The advantage of the Newton iteration method is its fast quadratic convergence. An integral part of each Newton step is the calculation of the inverse of a matrix M acting upon a vector, *i.e.,*$M^{-1}\mathbf{b}$. In order to speed up this particular step, we used a conjugate gradient method to solve the corresponding linear system $Mx = \mathbf{b}$.

As for the 2 class case, after approximating the posterior by a Gaussian, we average the softmax (sigmoid for 2 classes) output over the Gaussian approximation to the posterior. At present, we simply sample this integral using 1000 draws from a Gaussian random vector generator.

25.6 Discussion

One should always bear in mind that *all models are wrong!* (If we knew the correct model, we wouldn't need to bother with this whole business). Also, there is no such thing as assumption free predictions, or a "universal" method that will always predict well, regardless of the problem. In particular, there is no way that one can simply look at data and determine what is signal and what is noise. The separation of a signal into such components is done on the basis of *belief* about the noise/signal process.

As far as the Bayesian is concerned, the best thing to do is to incorporate as much of the knowledge we have about possible solutions, quantifying our beliefs about which models are more or less likely. These beliefs are then updated in light of the observed data, giving rise to a principled method of model selection. Also, concepts such as model averaging are intrinsically inherent in this framework. That predictions are based on such subjective prior belief is in no sense a drawback. Indeed, the insistence of this framework in making the user specify his/her model assumptions explicitly greatly enhances different scientists ability to evaluate each others work.

Note that our stated aim in this practical was to find a good regression model and *not* to try to interpret the data. This is an important difference and should be kept in mind. It may well be that using a non-linear model, we can (also) fit the data well using far fewer adjustable parameters. In that case, we may be able to place more emphasis on interpreting such lower dimensional representations and perform *feature extraction* (as potentially in neural networks). However, linear models are generally easier to work with and are a useful starting place in our search for a good regression model. Coupled with the Gaussian noise assumption, using a Gaussian prior on the weights of a linear model defines a Gaussian Process in the output space. In this sense, generalised linear models *are* Gaussian Process with a particular covariance function. Once this is realised, one is free to directly *specify* the form of the covariance function, as we did in the latter half of the practical, and this obviates the need for a weight space. This is in some cases convenient since it therefore also deals with the problem of the curse of dimensionality. As far as the Bayesian is concerned in this regression context, without any explicit belief about the data generating process, the only requirements/prior belief one has are typically expressed in terms of the smoothness of the *function* itself. That is, the question of parameter complexity is irrelevant - the Bayesian is perfectly happy to use a model with a billion parameters or one with 10 parameters. Whichever model most aptly captures his/her belief about the data generating *function* is the preferred choice.

IV. Approximate Inference Methods

26 Sampling

Sampling is a big topic. An excellent introduction to this area is *Neal, R. M. (1993) Probabilistic Inference Using Markov Chain Monte Carlo Methods, Technical Report CRG-TR-93-1, Dept. of Computer Science, University of Toronto.*

See also related articles at
`http://www.cs.toronto.edu/~radford/publications.html`

Readers are also invited to read the chapter on sampling methods methods in David MacKay's book.

26.1 Introduction

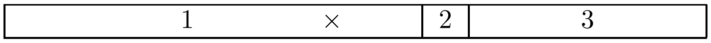
Consider the distribution $p(x)$. Sampling is the process of generating a vector x from the distribution $p(x)$, with probability given by $p(x)$. One way to view this is that if we have a procedure $\mathcal{S}(p)$ from which we can generate a set of P samples $x^1, \ldots, x^P$, then, in the limit of $P \to \infty$, the relative frequency that the sample value x occurs tends to $p(x)$. (In the continuous distribution case, this can be defined as the limiting case of the relative frequency $x \in \Delta$ tending to $\int_{x \in \Delta} p(x)$.

In both cases, sampling simply means drawing examples from the distribution with the correct frequency.

One dimensional Discrete distribution

In the sequel, we assume that a random number generator exists which is able to produce a value uniformly at random from the unit interval $[0, 1]$. We will make use of this uniform random number generator to draw samples from non-uniform distributions.

As an example of the above, consider the one dimensional discrete distribution $p(x)$ where x can be in any of the states 1 to K. To be specific, consider the three state distribution $p(x = 1) = 0.6$, $p(x = 2) = 0.1$, $p(x = 3) = 0.3$.



This represents a partitioning of the unit interval $[0, 1]$ in which the interval $[0, 0.6]$ has been labelled as state 1, $[0.6, 0.7]$ as state 2, and $[0.7, 1.0]$ as state 3. If we were to drop a point × anywhere at random, uniformly in the interval $[0, 1]$, the chance that × would land in interval 1 is 0.6, and the chance that it would be in interval 2 is 0.2 and similarly, for interval 3, 0.3. This therefore defines for us a valid sampling procedure for discrete one-dimensional distributions:

Cumulant Let p_i, $i = 1, K$ label the K state probabilities. Calculate the so-called cumulant, $c_i = \sum_{j \leq i} p_j$, and set $c_0 = 0$. (In the above, we have $(c_0, c_1, c_2, c_3) = (0, 0.6, 0.7, 1)$). Draw a value u uniformly at random from the unit interval $[0, 1]$. Find that i for which $c_{i-1} \leq u \leq c_i$. The sampled state is then i. In our example, we may have sampled $u = 0.66$. Then the sampled x state would be state $x = 2$, since this is in the interval $[c_1, c_2]$.

Continuous Case

Intuitively, the generalisation of the discrete case to the continuous case is clear. First we calculate the cumulant density function

$$C(y) = \int_{-\infty}^{y} p(x)dx$$

Then we generate a random u uniformly from $[0, 1]$, and then obtain the corresponding sample value x by solving $C(x) = u$. For some special distributions, such as Gaussians, very efficient equivalent ways to achieve this are usually employed.

Multi-variate discrete distributions

One way to generalise the one dimensional case to a higher dimensional case $p(x_1, \ldots, x_n)$ would be to translate the higher dimensional case into an equivalent one-dimensional distribution. We can enumerate all the possible joint states $(x_1, \ldots, x_n)$, giving each a unique integer y from 1 to the total number of states accessible. This then transforms the multi-dimensional distribution into an equivalent one-dimensional distribution, and sampling can be achieved as before. Of course, in high dimensional distributions, we would have, in general, exponentially many states if x, and an explicit enumeration would be impractical.

An alternative exact approach would be to capitalise on the relation

$$p(x_1, x_2) = p(x_2|x_1)p(x_1)$$

This suggests that we can sample from the joint distribution $p(x_1, x_2)$ by first sampling a value for x_1 from the one-dimensional $p(x_1)$, and then, with x_1 clamped to this value, sampling a value for x_2 from the one-dimensional $p(x_2|x_1)$.

It is clear how to generalise this to say three or more variables by using

$$p(x_1, x_2, x_3) = p(x_3|x_2, x_1)p(x_2|x_1)p(x_1)$$

In order to calculate $p(x_1)$ we need to marginalise the joint distribution $p(x_1, \ldots, x_n)$ over all the other variables (and similar calculations are required for the other conditional distributions). Such marginals, in general, will require the summation over an exponential number of states, and will, except for small n, generally also be impractical.

Belief Networks

Here we specify the joint probability by factors of lower dimensional conditional distributions.

$$p(x) = \prod_i p(x_i|\mathrm{pa}\,(x_i))$$

For example

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

as shown below.

By making a so-called *ancestral ordering* (in which parents always come before children), as in the equation above, one can sample first from those nodes that do not have any parents (here, x_1 and x_2). Given these values, one can then sample x_3, and then x_4,x_5 and finally x_6. Hence, despite the presence of loops in the graph, such a *forward sampling* procedure is straightforward. Any quantity of interest, for example, a marginal $p(x_5)$, is approximated by counting the relative number of times that x_5 is in a certain state in the samples.

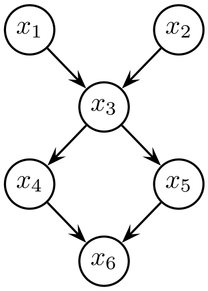


Figure 26.1: A Belief Network without any clamped variables.

How can we sample from a distribution in which certain variables are clamped in evidential values? One approach would be to proceed as above with forward sampling, and then discard any samples which do not match the evidential states. This can be extremely inefficient, and is not recommended.

Gibbs Sampling

One of the simplest ways to more effectively account for evidence is to employ a recursive procedure. One way to motivate the procedure is to assume that someone has presented you with an sample x^1 from the distribution $p(x)$. (For the moment, we leave aside the issue of evidence). We then consider a particular variable, x_i. We may write

$$p(x) = p(x_i|x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)p(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$$

(One may view this decomposition as x_i given all its parents, mutliplied by the probability of the parents). Since we assume that someone has already provided us with a sample x^1, from which we can readoff the ‘parental’ state $x^1_1, \dots, x^1_{i-1}, x^1_{i+1}, \dots, x^1_n$, we can then draw a sample from

$$p(x_i|x^1_1, \dots, x^1_{i-1}, x^1_{i+1}, \dots, x^1_n)$$

This distribution is usually easy to sample from since it is one-dimensional (this holds for both discrete and continuous distribtions). We call this new sample (in which only x_i has been updated) x^2. One then selects another variable x_j to sample and, by this procedure, one generate a set $x^1, \dots, x^P$ of ‘samples’.

There are a couple of important remarks about this procedure. Clearly, if the initial sample x^1 is not representative – that is, it is fact a part of the state space that is relatively extremely unlikely, then we should not really expect that the samples we draw will initially be very representative either. This motivates the so-called 'burn in' stage in which, perhaps 1/3 of the samples are discarded. Another remark is that it is clear there will be a high degree of correlation in any two successive samples, since only one variable is updated. What we would really like is that each sample x is simply drawn 'at random' from $p(x)$ – clearly, in general, such random samples will not possess the same degree of correlation as those from Gibbs sampling. This motivates so-called subsampling, in which, say, every 10^{th}, sample $x^K, x^{K+10}, x^{K+20}, \ldots$, is taken, and the rest discarded.

Gibbs sampling is reminiscent of MinConflicts type procedures in Computer Science, in which a single variable is updated to see if this is a better or worse solution to a minimisation problem. Essentially, Gibbs sampling is the generalisation of this to the stochastic case.

Evidence

Evidence is easy to deal with in the Gibbs sampling procedure. One simply clamps for all time those variables that are evidential into their evidential states. There is also no need to sample for these variables, since their states are known.

Despite its simplicity, Gibbs sampling is one of the most useful and popular sampling methods, especially in discrete cases. However, one should bear in mind that convergence is a major issue – that is, answering questions such as 'how many samples are needed to be reasonably sure that my sample estimate $p(x_5)$ is accurate?', is, to a large extent, an unknown. Despite many mathematical results in this area, little is really known about these issues, and general rules of thumb, and sensible awareness on behalf of the user are rather required. (Indeed, if one were able to answer such questions, one would understand the distribution well enough that usually some exact technique would be preferable).

Caution

As with most sampling schemes, a word of caution is required. Whilst there are some formal results that show that Gibbs sampling (under certain restrictions) is a correct sampling procedure, one can easily construct cases where it will fail. In fig(26.2), we show such a case in which the two dimensional continuous distribution has mass only in the lower left and upper right regions. In that case, if we start in the lower left region, we will always remain there, and never explore the upper right region. This problem occurs essentially because there are two regions which are not connected by a path which is reachable by Gibbs sampling. Such multi-modality is the scourge of sampling in general, and is very difficult to address.

Importance Sampling

The aim here is to replace sampling with respect to the intractable distribution $p(x)$, and instead sample from a tractable, simpler distribution $q(x)$. We need to in someway adjust/reweight the samples from $q(x)$ such that, in the limit of a large

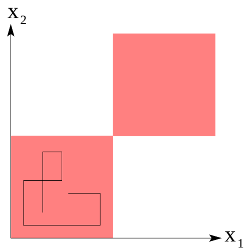


Figure 26.2: A two dimensional distribution for which Gibbs sampling fails. The upper right region is never explored. This is a case where the sampler is non-ergodic. For an ergodic sampler there is a non-zero chance any region of the space will be visited.

number of samples, the correct results will be obtained. Consider the average

$$\int f(x)p(x) = \frac{\int f(x)p^*(x)}{\int p^*(x)}dx \tag{26.1.1}$$

$$= \frac{\int f(x)\frac{p^*(x)}{q(x)}q(x)dx}{\int \frac{p^*(x)}{q(x)}q(x)dx} \tag{26.1.2}$$

Let $x^1, \ldots x^P$ be samples from $q(x)$, then we can approximate the above by

$$\int f(x)p(x) \approx \frac{\sum_{\mu=1}^P f(x^\mu)\frac{p^*(x^\mu)}{q(x^\mu)}}{\sum_{\mu=1}^P \frac{p^*(x^\mu)}{q(x^\mu)}} = \sum_{\mu=1}^P f(x^\mu)r^\mu$$

where

$$r^\mu = \frac{\frac{p^*(x^\mu)}{q(x^\mu)}}{\sum_{\mu=1}^P \frac{p^*(x^\mu)}{q(x^\mu)}}$$

Hence, in principle, this reweighting of the samples from q will give the correct result. In high dimensional spaces x, however, the r^μ will tend to have only one dominant value close to 1, and the rest will be zero, particularly if the sampling distribution q is not well matched to p, since then the ratio q/p will not be close to unity. However, in a moderate number of dimensions, perhaps less than 10 or so, this method can produce reasonable results. Indeed, it forms the basis for a simple class of algorithms called *particle filters*, which are essentially importance sampling for temporal Belief Networks (eg non-linear Kalman Filters), in which one forward samples from a proposal distribution q, and one can exploit the simplified Markov structure to recursively define reweighting factors.
See `http://www-sigproc.eng.cam.ac.uk/smc/index.html` for references.

26.2 Markov Chain Monte Carlo (MCMC)

Let's restate the problem of sampling from a distribution $p(x)$. Let's write

$$p(x) = \frac{1}{Z}p^*(x)$$

where Z is the normalisation constant of the distribution. Usually, we will be able to evaluate $p^*(x)$, but not Z, since $Z = \int_x p^*(x)$ is an intractable high dimensional summation/integration.

Gibbs sampling is a special case of a more general procedure, in which, given a current sample x^{μ}, we wish to generate a new sample $x^{\mu+1}$ such that together, the sequence $x^1, \ldots, x^P$ represents a representative set of samples of $p(x)$.

Understanding MCMC

Consider the conditional distribution $p(x_{t+1}|x_t)$. If we are given an initial sample x_1, then we can recursively generate samples $x_1, x_2, \ldots, x_t$. After a long time $t >> 1$, we can plot the samples x_t. Are the samples x_t samples from some distribution and, if so, *which* distribution? The answer to this is (generally), yes, they are samples from the *stationary* distribution $p_\infty(x)$ which is defined as

$$p_\infty(x') = \int_x p(x'|x)p_\infty(x)$$

This equation defines the stationary distribution, from which we see that the stationary distribution is equal to the eigenfunction with unit eigenvalue of the transition kernel. Under some mild properties (STATE!! ergodicity usually required), *every* transition distribution has a stationary distribution. This is also *unique* (STATE conditions required).

The idea in MCMC is to reverse this process. If we are given the distribution $p(x)$, can we find a transition $p(x'|x)$ which has $p(x)$ as its stationary distribution? If we can, then we can draw samples from the Markov Chain, and use these as samples from $p(x)$.

Note that whilst (usually) every Markov transition $p(x'|x)$ has a unique stationary distribution, every distribution $p(x)$ has a great many different transitions $p(x'|x)$ with $p(x)$ as their equilibrium distributions. (This is why there are very many different MCMC sampling methods for the same distribution).

Detailed Balance

How do we construct transitions $p(x'|x)$ with given $p(x)$ as their stationary distributions? One convenient trick is to assume detailed balance. This is the assumption

$$p(x'|x)p(x) = p(x|x')p(x')$$

which is required to hold for all x and x'.

If $p(x'|x)$ and $p(x|x')$ satisfy detailed balance, then

$$\int_x p(x'|x)p(x) = \int_x p(x|x')p(x') = p(x')$$

That is, detailed balance is a sufficient condition for stationarity. It is not, however, a necessary condition.

For example, consider drawing samples from the uniform distribution $U[0, 1]$. One (rather silly!) way to do this would be to draw x' as follows. Draw a random number y from a small interval uniform distribution $y \sim U[0, \epsilon]$ where, say $\epsilon = 0.5$. Then take x' to be the value $x + y$ with wrapped boundary conditions. That is, a point $x' = 1 + \delta$ gets mapped to δ. Clearly, under this scheme, we will eventually sample correctly from the uniform distribution $U[0, 1]$ however, in a left to right manner. This clear irreversibility of the chain shows that detailed balance is not a necessary criterion for correct MCMC sampling.

Metropolis/Hastings Sampling

The detailed balance criterion can be written

$$\frac{p(x'|x)}{p(x|x')} = \frac{p(x')}{p(x)}, \forall x, x'$$

This can make the process of constructing a suitable Markov Chain easier since only the relative value of $p(x')$ to $p(x)$ is required, and not the absolute value of $p(x)$ or $p(x')$, in the criterion.

For example, in the continuous domain, we might postulate that[1]

$$p(x'|x) = q(x'|x)f(x',x) + \delta(x'|x)\left(1 - \int_{x''} q(x''|x)f(x'',x)\right)$$

would be a candidate transition. The reader may verify that this is indeed a distribution since

$$\int_{x'} p(x'|x) = \int_{x'} q(x'|x)f(x',x) + 1 - \int_{x''} q(x''|x)f(x'',x) = 1$$

The above transition clearly splits into two cases, namely when $x' = x$ and $x' \neq x$. When $x' = x$, then clearly detailed balance trivially holds. In the case $x' \neq x$, then

$$p(x'|x) = q(x'|x)f(x',x)$$

Then we require (for $x' \neq x$)

$$\frac{q(x'|x)f(x',x)}{q(x|x')f(x,x')} = \frac{p(x')}{p(x)}$$

or

$$\frac{f(x',x)}{f(x,x')} = \frac{q(x|x')p(x')}{q(x'|x)p(x)}$$

Then consider the function

$$f(x',x) = \min(1, \frac{q(x|x')p(x')}{q(x'|x)p(x)})$$

Then if $q(x|x')p(x') > q(x'|x)p(x)$ $f(x',x) = 1$, and $f(x,x') = q(x'|x)p(x)/q(x|x')p(x')$, and hence

$$\frac{f(x',x)}{f(x,x')} = \frac{1}{q(x'|x)p(x)/q(x|x')p(x')} = \frac{q(x|x')p(x')}{q(x'|x)p(x)}$$

The reader may show that, conversely, if $q(x|x')p(x') \leq q(x'|x)p(x)$, we also get $\frac{f(x',x)}{f(x,x')} = \frac{q(x|x')p(x')}{q(x'|x)p(x)}$. Hence the function $f(x',x)$ as defined above is a suitable function to ensure that $p(x'|x)$ satisfies detailed balance. This function is called the *Metropolis-Hastings acceptance function*. Other acceptance functions may also be derived.

[1] One could contemplate, for example, a normalisation by division style method. However, it is not necessarily easy to sample from this transition distribution. The beauty of the Metropolis method is that this subtractive normalisation results in distribution that is easy to sample from, as we will see.

How do we then sample from $p(x'|x)$? Imagine we draw a candidate sample x' from $q(x'|x)$. If $q(x|x')p(x') > q(x'|x)p(x)$, then $f(x',x) = 1$, and we must have $x' \neq x$ (since, otherwise, $p(x) > p(x)$, which cannot be true) and we simply have $p(x'|x) = q(x'|x)$ – namely we accept the sample x'. Conversely, if $q(x|x')p(x') \leq q(x'|x)p(x)$, then $f(x',x) = \frac{q(x|x')p(x')}{q(x'|x)p(x)}$, and we cannot rule out that $x' = x$. Hence

$$p(x'|x) = q(x'|x)f(x',x) + \delta(x'|x)\left(1 - \int_{x''} q(x''|x)f(x'',x)\right)$$

This can be interpreted as a mixture distribution with two distributions $q(x'|x)$ and $\delta(x'|x)$, and associated mixture coefficients $f(x',x)$ and $1 - \int_{x''} q(x''|x)f(x'',x)$. To sample from this mixture, we sample from the mixture weight $f(x',x)$. We therefore with probability $f(x',x)$ draw a sample from $q(x'|x)$ (that is, we accept the candidate) and otherwise take the sample $x' = x$. A common mistake in MCMC is, when we reject the candidate x', simply to restart the procedure. The correct approach is that, if the candidate x' is rejected, we take the original x as a new sample. Hence, another copy of x is included in the sample set – ie each iteration of the algorithm produces a sample – either a copy of the current sample, or the candidate sample.

The reader may show that Gibbs sampling can be put in the this framework for a suitable proposal $q(x'|x)$.

Whilst all of this is quite cool, the reader should bear in mind a couple of points. Firstly, having a 'correct' transition does not guarantee that the samples will indeed be from $p(x)$. The proposal distribution $q(x'|x)$ may not explore all regions, in which case we have not shown detailed balance holds for all points x and x'. This is what can happen in the example of the Gibbs sampler, which is not ergodic – we locally satisfy detailed balance in a region, but not over all space, hence the samples are not samples from $p(x)$.

Another related point is that, even if we can guarantee ergodicity of the chain, we have no clue as to how long we need to wait until we have drawn a representative sample from $p(x)$. The reason for this is essentially that, if the chain is ergodic, then indeed, eventually, we will be drawing samples from the stationary distribution – but *when?*. Assessing convergence is a major headache in MCMC, and in some sense, just as difficult as sampling from $p(x)$ itself, since we need to have some global idea of the distribution in order to know how long before we are likely to have reached a representative point of $p(x)$. In practice, there are some heuristics....

Gaussian Proposal distribution

If we use

$$q(x'|x) \propto e^{-\frac{1}{2\sigma^2}(x'-x)^2}$$

Then $q(x'|x) = q(x|x')$, and the acceptance criterion is simply

$$a = \frac{p^*(x')}{p^*(x)}$$

Hence, if the unnormalised probability of the candidate state is higher than the current state, we accept the candidate. Otherwise, if the unnormalised probability of the candidate state is lower than the current state, we accept the candidate only with probability $p^*(x')/p^*(x)$. Otherwise, the candidate is rejected, and the new

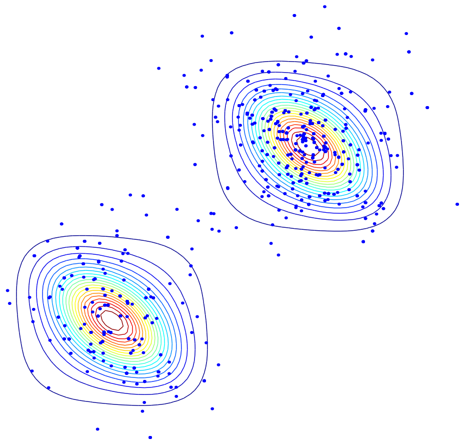


Figure 26.3: One thousand samples from a non-Gaussian distribution. The contours plotted are iso-probability contours. Here, Metropolis sampling was used with a standard deviation of 1 in each dimension.

sample is taken to be x.

The Metropolis algorithm with isotropic Gaussians above is intuitive, and simple to implement. However, it is not necessarily very efficient. Intuitively, we will certainly accept the candidate if the unnormalised probability is higher than the probability at the current state. We attempt to find a higher point on the distribution essentially by making a small jump in a random direction. In high dimensions, it is unlikely that a random direction will result in a value of the probability which is higher than the current value. Because of this, only very small jumps (which will typically result in $a < 1$) will be accepted. However, if only very small jumps are accepted, the speed with which we explore the space x is extremely slow, and a tremendous number of samples would be required.

Assessing convergence

.... a bunch of heuristics that I'll write down sometime....

Auxilliary Variable Methods

General idea is that we wish to sample from $p(x)$. We introduce a distribution $p(y|x)$, and hence have a joint distribution $p(x, y) = p(y|x)p(x)$. We then draw samples (x^μ, y^μ) from this joint distribution. A valid set of samples from $p(x)$ are then given by the x^μ. In order for this to be useful, we do not always simply draw a sample from $p(x)$, and then a sample for y from $p(y|x)$. Repeating this would be senseless. However, if we, for example, where to perform Gibbs sampling, sampling alternately from $p(y|x)$ and $p(x|y)$ (which is found from $p(x|y) \propto p(y|x)p(x)$), the auxiliary variable may enable us to find an easy sampling distribution $p(x|y)$ and consequently, we may be able to mode hop.

Hybrid Monte Carlo

This is a method for continuous systems that aims to make non-trivial jumps in the samples and, in so doing, to jump potentially from one mode to another.

It is customary (though not necessary) to derive Hybrid MCMC in terms of Hamiltonians. We will follow this approach here as well.

Let' define the difficult distribution from which we wish to sample as[2]

$$p(x) = \frac{1}{Z_x} e^{H_x(x)}$$

for some given 'Hamiltonian' $H(x)$. We then define another, 'easy' distribution

$$p(y) = \frac{1}{Z_y} e^{H_y(y)}$$

so that we can define a joint distribution

$$p(x, y) = p(x)p(y) = \frac{1}{Z} e^{H_x(x)+H_y(y)}$$

The algorithm alternates between a Gibbs and a so-called *dynamic* step. In the Gibbs step, we simply draw a new value for y from $p(y)$. In the dynamic step, we draw a sample of $p(x)$ by drawing a sample from $p(x, y)$ and discarding then the sampled y.

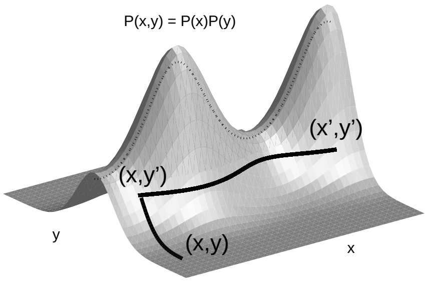


Figure 26.4: Hybrid Monte Carlo. Starting from the point x, y, we first draw a new value for y from the Gaussian $p(y)$. Then we use Hamiltonian dynamics to traverse the distribution at roughly constant energy $H(x, y)$ to reach a point x', y'. We accept this point if $H(x', y') > H(x, y')$. Otherwise this candidate is accepted with probability $\exp(H(x', y') - H(x, y'))$.

In the standard form of the algorithm, we choose a multi-dimensional Gaussian auxiliary distribution, usually with $dim(y) = dim(x)$:

$$H_y(y) = -\frac{1}{2} y^T y$$

The Gibbs step is the trivial since we just sample from the Gaussian $p(y)$.

[2] In physics, the convention is to write $p(x) = \frac{1}{Z_x} e^{-H_x(x)}$ since probable states correspond to low energy (Hamiltonian) states. De gustibus non est disputandum!

The dynamic step is a Metropolis step, with a very special kind of proposal distribution. In the dynamic step, the main idea is to go from one point of the space x, y to a new point x', y' that is a non-trivial distance from x, y and which will be accepted with a high probability. In the basic formulation using a symmetric proposal distribution, we will accept the candidate x', y' if the values $H(x', y')$ is higher than or close to the value $H(x, y')$. How can such a non-trivial distance between x, y and x', y' be accomplished? One way to do this is to use *Hamiltonian dynamics*.

Hamiltonian Dynamics

Imagine that we have a Hamiltonian $H(x, y)$. We wish to make an update $x' = x + \Delta x$, $y' = y + \Delta y$ for small Δx and Δy. Hamiltonian dynamics is defined by the requirement of *energy preservation*. That is,

$$H(x', y') = H(x, y)$$

We can satisfy this (up to first order) by considering the Taylor expansion

$$\begin{aligned} H(x', y') &= H(x + \Delta x, y + \Delta y) \\ &\approx H(x) + \Delta x^T \nabla_x H(x, y) + H(y) + \Delta y^T \nabla_y H(x, y) + O(|\Delta x|^2) + O(|\Delta y|^2) \end{aligned} \tag{26.2.1}$$

The energy preservation requirement is then that

$$\Delta x^T \nabla_x H(x, y) + \Delta y^T \nabla_y H(x, y) = 0$$

This is a single scalar requirement, and there are therefore many different solutions for Δx and Δy that satisfy this single condition. In physics, it is customary to assume isotropy, which limits dramatically the number of possible solutions to essentially just the following:

$$\Delta x = \epsilon \nabla_y H(x, y) \qquad \Delta y = -\epsilon \nabla_x H(x, y)$$

where ϵ is a small value to ensure that the Taylor expansion is accurate. Hence

$$x(t+1) = x(t) + \epsilon \nabla_y H(x, y) \qquad y(t+1) = y(t) - \epsilon \nabla_x H(x, y)$$

Defines Hamiltonian dynamics – a locally energy preserving dynamical system.

Discarding the isotropy requirement opens up the possibility for other dynamics, for example, relativistic dynamics. In practice, however, the simple Hamiltonian dynamics is usually considered adequate.

For the Hybrid MC method, $H(x, y) = H(x) + H(y)$, so that $\nabla_x H(x, y) = \nabla_x H(x)$ and $\nabla_y H(x, y) = \nabla_y H(y)$ For the Gaussian case, $\nabla_y H(x, y) = -y$.

$$x(t+1) = x(t) - \epsilon y \qquad y(t+1) = y(t) - \epsilon \nabla_x H(x)$$

There are specific ways to implement the dynamic equations above (called *Leapfrog* discretization) that are more accurate – see the Radford Neal reference. (Special case of simplectic discretization I think).

We can then follow the Hamiltonian dynamics for many time steps (usually of the order of several hundred) to reach a candidate point x', y'. If the Hamiltonian dynamics was well behaved, $H(x', y')$ will have roughly the same value as $H(x, y)$.

We then do a Metropolis step, and accept the point x', y' if $H(x', y') > H(x, y)$ and otherwise accept it with probability $exp(H(x', y') - H(x, y))$. If rejected, we take the initial point x, y as the sample.

In order to make a symmetric proposal distribution, at the start of the dynamic step, we choose $\epsilon = +\epsilon_0$ or $\epsilon = -\epsilon_0$ uniformly. This means that there is the same chance that we go back to the point x, y starting from x', y', as vice versa.

Combined with the Gibbs step, we then have the general procedure.

1. Start from x, y. Draw a new sample y' from $p(y)$.
2. Then, starting from x, y', choose a random (forwards or backwards) and then perform Hamiltonian dynamics for some time steps until we reach a candidate x', y'. Accept x', y' if $H(x', y') > H(x, y)$, otherwise accept it with probability $\exp(H(x', y') - H(x, y))$. If rejected, we take the sample as x, y'.
3. The above steps are repeated.

One obvious feature of HMC is that we now use, not just the potential $H(x)$ to find define candidate samples, but the *gradient* of $H(x)$ as well. An intuitive reason for the success of the algorithm is that it is less myopic than straightforward Metropolis, since the use of the gradient enables the algorithm to feel it's way to other regions of high probability, by following at all times likely paths in the augmented space. One can also view the auxiliary variables as momentum variables – it is as if the sample has now a momentum. Provided this momentum is high enough, we can escape local minima......more later.

Slice Sampling

Blah

Swendson-Wang

This is a classic algorithm used for discrete variables. The main motivation here is to introduce $p(y|x)$ in such a way that the distribution $p(x|y)$ is easy to sample from.

Originally, the SW method was introduced to alleviate the problems encountered in sampling from Ising Models close to their critical temperature, in which Gibbs sampling completely breaks down.

In it's simplest form, the Ising model with no external interactions on a set of variables $x_1, \ldots, x_n$, $x_i \in \{0, 1\}$ takes the form

$$p(x) = \frac{1}{Z} \prod_{i \sim j} e^{\beta I[x_i = x_j]}$$

which means that this is a pairwise Markov network with a potential contribution e^{β} if neighbouring nodes i and j are in the same state, and a contribution 1 otherwise. We assume that $\beta > 0$ which encourages neighbours to be in the same state. The lattice based neighbourhood structure makes this difficult to sample from, and especially when the inverse temperature encourages large scale islands to form. In that case, the probability of an individual variable being flipped by Gibbs sampling is negligible.

We wish to introduce a distribution $p(y|x)$ so that $p(x|y)$ is easy to sample from. The easiest kind of distributions to sample from are factorised. Let's see if we can make $p(x|y)$ factorised.

$$p(x|y) \propto p(y|x)p(x) \propto p(y|x) \prod_{i\sim j} e^{\beta I[x_i=x_j]}$$

It's clear that we need to employ $p(y|x)$ to cancel the terms $e^{\beta I[x_i=x_j]}$. We can do this by making

$$p(y|x) = \prod_{i\sim j} p(y_{ij}|x_i, x_j) = \prod_{i\sim j} \frac{1}{z_{ij}} I[0 < y_{ij} < e^{\beta I[x_i=x_j]}]$$

where $I[0 < y_{ij} < e^{\beta I[x_i=x_j]}]$ denotes a uniform distribution between 0 and $e^{\beta I[x_i=x_j]}$. z_{ij} is the normalisation constant $z_{ij} = e^{\beta I[x_i=x_j]}$. Hence

$$\begin{aligned}
p(x|y) &\propto p(y|x)p(x) && (26.2.2)\\
&\propto \prod_{i\sim j} \frac{1}{e^{\beta I[x_i=x_j]}} I[0 < y_{ij} < e^{\beta I[x_i=x_j]}] e^{\beta I[x_i=x_j]} && (26.2.3)\\
&\propto \prod_{i\sim j} I[0 < y_{ij} < e^{\beta I[x_i=x_j]}] && (26.2.4)
\end{aligned}$$

Let's assume that we have a sample y_{ij}. If $y_{ij} > 1$ then to draw a sample from $p(x|y)$, we must have $1 < e^{\beta I[x_i=x_j]}$, which means that x_i and x_j are in the same state. Otherwise, if $y_{ij} < 1$, then what constraint does this place on what the x can be? None! Hence, wherever $y_{ij} > 1$, we bond x_i and x_j to be in the same state. The probability

$$y_{ij} > 1 = \int_{y_{ij}=1}^{\infty} \frac{1}{z_{ij}} I[0 < y_{ij} < e^{\beta I[x_i=x_j]}] \frac{e^{\beta} - 1}{e^{\beta}} = 1 - e^{-\beta}$$

Hence, if $x_i = x_j$, we bind x_i and x_j to be in the same state with probability $1 - e^{-\beta}$. After doing this for all the x_i and x_j pairs, we will end up with a graph in which we have clusters of like-state bonded variables. Which state each cluster is equal. Hence the algorithm simply chooses a random state for each cluster.

The algorithm then does the following:

1. If $x_i = x_j$, we bond variables x_i and x_j with probability $1 - e^{-\beta}$. Repeat this for all variables.
2. For each cluster formed from the above, set the state of the cluster uniformly at random.
3. Repeat the above steps.

Pictures from Higdon. Matlab code.

Temporal Distributions

Many applications involve temporal distributions of the generic form

$$p(v_{1:T}, h_{1:T}) = p(v_1|h_1)p(h_1) \prod_{t=2}^{T} p(v_t|h_t)p(h_t|h_{t-1})$$

We encountered a few already, namely the Kalman Filter, Hidden Markov Model and Switching Kalman Filter. Our interest here will be in the calculation of $p(h_t|v_{1:T})$.

In the mentioned models, we have used either exact inference methods, or developed (in the SKF case) approximate inference methods. However, there are cases where the transitions are such that it may not be clear how to form an appropriate analytic approximation procedure (although, in my experience, such situations are rare), and more general numerical approximations are sought.

It should be born in mind that tayloring the approximation method to the model at hand is usually vital for reasonable performance. Nevertheless, we'll discuss below some fairly general sampling procedures that may be brought to bear, and have proved popular, mainly due to their implementational simplicity.

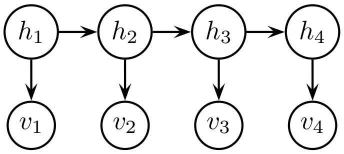


Figure 26.5: A Switching Kalman Filter. The variables h and v are Gaussian distributed. The Switch variables s are discrete, and control the means and variances of the Gaussian transitions.

Particle Filters

Despite our interest in $p(h_t|v_{1:T})$, PFs make the assumption that the so-called 'filtered estimate' $p(h_t|v_{1:t})$ would be a reasonable approximation or, at least, a quantity of interest.

The traditional viewpoint of a Particle Filter is as a recursive importance sampler. Here, we show how it can be viewed as a (somewhat severe) approximation of the Forward Pass in Belief Propagation.

$$\rho(h_t) \propto p(v_t|h_t) \int_{h_{t-1}} p(h_t|h_{t-1})\rho(h_{t-1}) \tag{26.2.5}$$

where the ρ message has the interpretation

$$\rho(h_t) \propto p(h_t|v_{1:t})$$

A PF can be viewed as an approximation of equation (26.2.5) in which the message $\rho(h_{t-1})$ has been approximated by a sum of δ-peaks:

$$\rho(h_{t-1}) = \sum_{i=1}^{L} w_{t-1}(i)\delta\left(h_{t-1} - h_{t-1}(i)\right) \tag{26.2.6}$$

where $w_{t-1}(i)$ are mixture components $\sum_{i=1}^{L} w_{t-1}(i) = 1$, and the points $h_{t-1}(i)$ are given values. In other words, the ρ message is represented as a weighted mixture of delta-spikes where the weight and position of the spikes are the parameters of the distribution.

Using equation (26.2.6) in equation (26.2.5), we have

$$\rho(h_t) = kp(v_t|h_t) \sum_{i=1}^{L} p(h_t|h_{t-1}(i))w_{t-1}(i) \tag{26.2.7}$$

The constant k is used to make $\rho(h_t)$ a distribution. Although $\rho(h_{t-1})$ was a simple sum of delta peaks, in general $\rho(h_t)$ will not be – the peaks get 'broadened' by the hidden-to-hidden and hidden-to-observation factors. One can think of many ways to approximate $\rho(h_t)$.

In PFs, we make another approximation of $\rho(h_t)$ in the form of a weighted sum of delta-peaks. There are many ways to do this, but the simplest is to just sample a set of points from the (unnormalised) distribution equation (26.2.7). There are many ways we could carry out this sampling. One simple way is to that equation (26.2.7) represents a mixture distribution.

Sampling from an unnormalised mixture using Importance Sampling

Consider

$$p(h) = \frac{1}{Z}\sum_{i=1}^{I} w_i \phi_i(h)$$

How can we sample from this distribution? Clearly, there are many approaches. A simple idea is to use importance sampling.

$$\langle f(h) \rangle = \frac{1}{Z}\sum_i w_i \int_h q_i(h) f(h) \frac{\phi_i(h)}{q_i(h)} \tag{26.2.8}$$

$$\approx \frac{1}{Z}\sum_i w_i \sum_\mu f(h_i^\mu)\frac{\phi_i(h_i^\mu)}{q_i(h_i^\mu)} \tag{26.2.9}$$

$$\approx \frac{\sum_i w_i \sum_\mu f(h_i^\mu)\frac{\phi_i(h_i^\mu)}{q_i(h_i^\mu)}}{\sum_i w_i \sum_\mu \frac{\phi_i(h_i^\mu)}{q_i(h_i^\mu)}} \tag{26.2.10}$$

$$\approx \frac{\sum_{i,\mu} r_i^\mu f(h_i^\mu)}{\sum_{i,\mu} r_i^\mu} \tag{26.2.11}$$

where $r_i^\mu \equiv w_i \frac{\phi_i(h_i^\mu)}{q_i(h_i^\mu)}$ If, say for each mixture component i, we generate a set of P samples $h_i^\mu, \mu = 1, \ldots, P$, then we will have $I \times P$ weights r_i^μ. We then need to *select* from this set, a smaller set (usually of size I again) points r_i^μ. This can either be done by discarding small r_i^μ, or sampling from the unnormalised distribution r_i^μ. One done, we have a set of retained r_i^μ, from which a new set of mixture weights w_i^* can be found by normalising the selected weights. Heuristics are usually required since, as is nearly always the case with naive IS, only a few of the weights will have significant value – exponential dominance problem. In practice, repopulation heuristics are usually employed to get around this. And other hacks.....

A better sampling approach

In my humble opinion, there is little advantage in using the (very poor) importance sampler. Rather, it is better to look again at the equation

$$\rho(h_t) \propto p(v_t|h_t) \int_{h_{t-1}} p(h_t|h_{t-1})\rho(h_{t-1}) \tag{26.2.12}$$

This can be interpreted as the marginal of the two time potential

$$\rho(h_t, h_{t-1}) \propto p(v_t|h_t)p(h_t|h_{t-1})\rho(h_{t-1})$$

Assuming that we have a sample h_{t-1}^{μ} from $\rho(h_{t-1})$, we can then draw a sample h_t from $\rho(h_t, h_{t-1})$ by Gibbs sampling, ie by sampling from the unnormalised conditional $\rho(h_t, h_{t-1} = h_{t-1}^{\mu})$. For each sample, we can then proceed to the next time step. This will then generate a single sample path $h_1^{\mu}, h_2^{\mu}, \dots h_T^{\mu}$. We repeat this procedure to get a set of sample paths (this can, of course, be also done in parallel, so that we generate at each time step, a set of sample $h_t^{\mu}, \mu = 1, \dots P$.

The advantage of this approach is that any of the more powerful sampling methods developed over the last 50 years can be used, and one is not hampered by the miserable performance of importance sampling.

```
% DEMO for sampling from a 2D distribution
close all;  hold on

x(:,1)=randn(2,1); % intial sample
s=1; % width of Metropolis candidate distribution

yj=0; xi=0; xxx=-6:0.1:6; yyy=-6:0.1:6; for yy=yyy
  yj=yj+1; xi=0;
  for xx=xxx
    xi=xi+1;
    z(xi,yj)=exp(logp([xx yy]));
  end
end
contour(xxx,yyy,z,20); % iso-probability contours

for mu=2:1000
  x(:,mu)=metropolis(x(:,mu-1),s,’logp’);
  plot(x(1,mu),x(2,mu),’.’);
  if mod(mu,100)==1
    drawnow
  end
end
```

```
function xnew=metropolis(x,s,logp)

% Get a new sample xnew from distribution exp(logp) using
% metropolis sampling with standard deviation s, and current sample x

  xcand=x+s*randn(size(x));
  loga=feval(logp,xcand)-feval(logp,x);
  if loga>0
    xnew=xcand;
  else
    r=rand;
    if log(r)<loga
      xnew=xcand;
    else
      xnew=x;
    end
  end
```

```
function l=logp(x)

  % make a non-Gaussian distribution.
  % larger f makes the distribution more bimodal.

  l1 = exp(- (x(1)^2+x(2)^2+sin(x(1)+x(2))^2));
  f=3;
  l2 = exp(- ((x(1)-f)^2+(x(2)-f)^2+sin(x(1)+x(2)-2*f)^2));
  l=log(l1+l2);
```

Rao-Blackwellisation

Explain why this is very often a red-herring since it assumes that one has a good sampler (which indeed is the whole problem of sampling!). Give a picture where it's easy to sample if high dimensions, but more multi-modal in the lower projected dimension, compounding the difficulty of sampling. (Rao-Blackwellisation says that the variance of the sampler will always be higher in the higher space – but that is in fact a good thing in many practical cases.) My feeling is that RB is just another inappropriate piece of theory that misses the point.

Appendix A Mathematics

NEED TO ADD LAGRANGE MULTIPLIERS

A.0.1 Vectors

The course assumes that you are familiar with the basics of vectors and vector calculations. Let $\mathbf{x}$ denote the n-dimensional vector with components

$$(x_1, x_2, \cdots, x_n)$$

Then $|\mathbf{x}|$ denotes the length of this vector, using the usual Euclidian definition:

$$|\mathbf{x}| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

The inner product $\mathbf{w} \cdot \mathbf{x}$ is defined as:

$$\mathbf{w} \cdot \mathbf{x} = \sum_{i=1}^{n} w_i x_i$$

and has a natural geometric interpretation as:

$$\mathbf{w} \cdot \mathbf{x} = |\mathbf{w}|\, |\mathbf{x}| \cos(\theta)$$

where θ is the angle between the two vectors. Thus if the lengths of two vectors are fixed their inner product is largest when $\theta = 0$, whereupon one is just some constant multiple of the other.

A.0.2 Matrices

The course assumes some familiarity with matrices, which are shown as upper-case bold letters such as $\mathbf{A}$. If the element of the i-th row and j-th column is a_{ij}, then $\mathbf{A}^T$ denotes the matrix that has a_{ji} there instead - the *transpose* of $\mathbf{A}$. So, for example if $\mathbf{A}$ is a 3×3 matrix:

$$\mathbf{A} = \begin{pmatrix} 2 & 3 & 4 \\ 4 & 5 & 9 \\ 6 & 7 & 1 \end{pmatrix}$$

then the transpose (written $\mathbf{A}^T$) is:

$$\mathbf{A}^T = \begin{pmatrix} 2 & 4 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 1 \end{pmatrix}$$

The product of two matrices $\mathbf{A}$ and $\mathbf{B}$ has $\sum_k a_{ik} b_{kj}$ in the i-th row and j-th column.

The matrix $\mathbf{I}$ is the identity or unit matrix, necessarily square, with 1s on the diagonal and 0s everywhere else. If $\det(\mathbf{A})$ denotes the determinant of a square matrix $\mathbf{A}$ then the equation

$$\det(\mathbf{A} - \lambda\mathbf{I}) = 0$$

is called the *characteristic polynomial* of $\mathbf{A}$. Using the example above, the characteristic polynomial would be:

$$\begin{vmatrix} 2-\lambda & 3 & 4 \\ 4 & 5-\lambda & 9 \\ 6 & 7 & 1-\lambda \end{vmatrix} = 0$$

which is

$$(2-\lambda)((5-\lambda)(1-\lambda) - 63) - 3(4(1-\lambda) - 54) + 4(28 - 6(5-\lambda)) = 0$$

which simplifies to:

$$-\lambda^3 + 8\lambda^2 + 82\lambda + 26 = 0$$

Note that a square matrix must satisfy its own characteristic polynomial, by definition of the polynomial, so (pre- or post-multiplying through by $\mathbf{A}^{-1}$) it provides a way to calculate the inverse of a matrix using only matrix multiplication, if that inverse exists. Clearly the inverse exists if and only if the matrix is square and $\det(\mathbf{A}) \neq 0$ (note that $\det(\mathbf{A})$ is the constant term in the characteristic polynomial).

The roots of the characteristic polynomial are called the *eigenvalues* of the matrix. Note that if $\mathbf{A}$ is an $m \times n$ matrix and $\mathbf{x}$ is an n-dimensional (column) vector, then

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

represents a linear map into an m-dimensional space. If $\mathbf{A}$ happens to be a square matrix then any vector which is transformed by the linear map into a scalar multiple of itself is called an *eigenvector* of that matrix. Obviously, in that case $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ for some λ. The eigenvectors can be found by finding the eigenvalues and then solving the linear equation set:

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = 0$$

An *orthogonal matrix* is a square matrix $\mathbf{A}$ such that $\mathbf{A}^T = \mathbf{A}^{-1}$. Such matrices represent a mapping from one rectangular co-ordinate system to another. For such a matrix,

$$\mathbf{A}\mathbf{A}^T = \mathbf{I}$$

- the inner product of any two different rows is 0 and the inner product of any row with itself is 1.

A.0.3 Basic combinatorics

The number of ways of selecting k items from a collection of n items is

$$\left(\begin{array}{c} n \\ k \end{array} \right) = \frac{n!}{k!(n-k)!}$$

if the ordering of the selection doesn't matter. This quantity is also the coefficient of x^k in the expansion of $(1+x)^n$. Stirling's formula provides a useful approximation for dealing with large factorials:

$$n! \approx n^n e^{-n} \sqrt{2\pi n}$$

There are a huge number of formulae involving combinations. For example, since $(1+x)^{n+1} = (1+x)^n(1+x)$ it is clear that

$$\left(\begin{array}{c} n \\ k \end{array} \right) + \left(\begin{array}{c} n \\ k+1 \end{array} \right) = \left(\begin{array}{c} n+1 \\ k+1 \end{array} \right)$$

and so on.

A.0.4 Basic probability and distributions

A random variable X is a variable which, in different experiments carried out under the same conditions, assumes different values x_i, each of which then represents a random event. A discrete random variable can take one of only a finite, or perhaps a countably infinite, set of values. A continuous random variable can take any value in a finite or infinite interval. Random variables are completely characterised by their probability density and distribution functions.

For a discrete random variable, if $p(X = x)$ is the probability that it takes the value x then

$$F(x) = p(X < x)$$

is the distribution function of X. For a continuous random variable, there is a probability density function $f(x)$ such that

$$\int_{-\infty}^{\infty} f(x)\, dx = 1$$

and the distribution function is then:

$$F(x) = \int_{-\infty}^{x} f(t)\, dt$$

For a discrete random variable, the mean value μ is

$$\mu = \sum x_i p(X = x_i)$$

and for a continuous variable it is

$$\mu = \int_{-\infty}^{\infty} t f(t)\, dt$$

The variance σ^2 is, for a discrete variable:

$$\sigma^2 = \sum (x_i - \mu)^2 p(X = x_i)$$

and for a continuous variable:

$$\sigma^2 = \int_{-\infty}^{\infty} (t - \mu)^2 f(t)\, dt$$

There are several widely-occurring distributions that are worth knowing about. Suppose that some event will happen with fixed probability p. Then the probability that it will happen exactly k times in n trials is

$$\left(\begin{array}{c} n \\ k \end{array} \right) p^k (1-p)^{n-k}$$

and this is the binomial distribution. It has mean np and variance $np(1-p)$. If one lets $n \to \infty$ one gets the Gaussian or normal distribution, typically parameterised by two constants a and b; it has density function

$$\frac{1}{a\sqrt{2\pi}} e^{-(x-b)^2/(2a^2)}$$

with mean b and variance a^2. If one starts with the binomial distribution and lets $n \to \infty$ and $p \to 0$ with the extra assumption that $np = a$, where a is some constant, then one gets the Poisson distribution with density function

We will not use the Poisson distribution in this course

$$\frac{a^k e^{-a}}{k!}$$

with mean and variance both a.

A.0.5 Partial differentiation

If $z = f(x_1, x_2, \cdots, x_n)$ is a function of n independent variables then one can form the partial derivative of the function with respect to one variable (say x_i),

$$\frac{\partial f}{\partial x_i}$$

by treating all other variables as constant. For example, if

$$f = xy + y^3$$

then

$$\frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x + 3y^2$$

The geometric significance of a quantity such as $\frac{\partial f}{\partial x}$ is as follows. If the function f is plotted and represents some suitably well-behaved surface, then this partial derivative represents the slope of the surface in a direction parallel to the x-axis at any given point (x, y). The total derivative dz is given by

$$dz = \sum_i \frac{\partial z}{\partial x_i} dx_i$$

and clearly, if all the x_i are functions of one variable t then

$$\frac{dz}{dt} = \sum_i \frac{\partial z}{\partial x_i} \frac{dx_i}{dt}$$

There is a directly analogous version of this ‘chain rule’ for the case where the x_i are each functions of several variables and you wish to find the partial derivative of z with respect to one of those variables.

Exercise: Find the partial derivatives of the function

$$f(x, y, z) = (x + 2y)^2 \sin(xy)$$

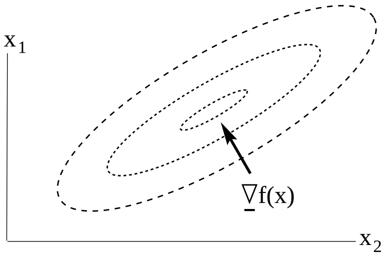


Figure A.1: Interpreting the gradient. The ellipses are contours of constant function value, $f = const$. At any point x, the gradient vector $\nabla f(x)$ points along the direction of maximal increase of the function.

A.0.6 The gradient vector operator

Consider a function $\phi(x)$ that depends on a vector x. We are interested in how the function changes when the vector x changes by a small amount : $x \to x + \boldsymbol{\delta}$, where $\boldsymbol{\delta}$ is a vector whose length is very small. According to a Taylor expansion, the function ϕ will change to

$$\phi\left(x + \boldsymbol{\delta}\right) = \phi(x) + \sum_i \delta_i \frac{\partial \phi}{\partial x_i} + O\left(\boldsymbol{\delta}^2\right) \tag{A.0.1}$$

We can interpret the summation above as the scalar product between the vector $\nabla\phi$ with components $[\nabla\phi]_i = \frac{\partial \phi}{\partial x_i}$ and $\boldsymbol{\delta}$.

$$\phi\left(x + \boldsymbol{\delta}\right) = \phi(x) + (\nabla\phi)^T \boldsymbol{\delta} + O\left(\boldsymbol{\delta}^2\right) \tag{A.0.2}$$

A.0.7 Interpreting the Gradient $\nabla f(x)$

The gradient points along the direction in which the function increases most rapidly. Why?

Consider a direction $\hat{p}$ (a unit legngth vector). Then a displacement, δ units along this direction changes the function value to

$$f(x + \delta\hat{p}) \approx f(x) + \delta \nabla f(x) \cdot \hat{p}$$

The direction $\hat{p}$ for which the function has the largest change is that which maximises the overlap

$$\nabla f(x) \cdot \hat{p} = |\nabla f(x)||\hat{p}| \cos\theta = |\nabla f(x)| \cos\theta$$

where θ is the angle between $\hat{p}$ and $\nabla f(x)$. The overlap is maximised when $\theta = 0$, giving $\hat{p} = \nabla f(x)/|\nabla f(x)|$. Hence, the direction along which the function changes the most rapidly is along $\nabla f(x)$.

A.0.8 Optimization: Lagrange multipliers

Suppose that you wish to find the stationary points (maxima or minima) of some n-argument function $f(\mathbf{x}) = f(x_1, \cdots, x_n)$, subject to the m constraints $g_1(\mathbf{x}) =$

$0, \cdots, g_m(\mathbf{x}) = 0$. Lagrange showed that they could be found as the solution of the $(n+m)$ equations in the $(n+m)$ variables $x_1, \cdots, x_n, \lambda_1, \cdots, \lambda_m$:

$$\begin{aligned}
\frac{\partial f}{\partial x_1} - \sum_{j=1}^{m} \lambda_j \frac{\partial g_j}{\partial x_1} &= 0 \\
&\cdots \\
\frac{\partial f}{\partial x_n} - \sum_{j=1}^{m} \lambda_j \frac{\partial g_j}{\partial x_n} &= 0 \\
g_1(\mathbf{x}) &= 0 \\
&\cdots \\
g_m(\mathbf{x}) &= 0
\end{aligned}$$

where the λ_j are m specially-introduced variables called Lagrange multipliers. This theorem provides a handy way to tackle a range of optimization problems. Notice that the above equations are the $(n+m)$ partial derivatives of the function

$$f - \sum_{j=1}^{m} \lambda_j g_j$$

each set to zero.

For example, to find the maximum of $f(x,y) = x+y$ subject to the constraint $x^2+y^2=1$, solve:

$$\begin{aligned}
1 - 2\lambda x &= 0 \\
1 - 2\lambda y &= 0 \\
x^2 + y^2 - 1 &= 0
\end{aligned}$$

to get $x = y = \lambda = \pm 1/\sqrt{2}$, after which you should then check to determine which of these two solutions is the true maximum.

Exercise: Find the maximum of $y - x$ subject to the constraint $y + x^2 = 4$.

You can find the answer to the same problem experimentally as follows. Plot the graph of $y = 4 - x^2$ and the graph of $y = x + m$, and find the largest value of m such that the two graphs still intersect.

A.1 Optimization methods

We are given a function $E(\mathbf{w})$ which depends on the vector of variables $\mathbf{w}$. We can also calculate the vector of partial derivatives (or gradient) $\mathbf{g}(\mathbf{w})$ where $g_i = \partial E / \partial w_i$. How should be use this information to optimize E? There are two methods that we consider

1. Gradient descent with fixed stepsize
2. Gradient descent with line searches

One of the most powerful general purpose optimisation methods is conjugate-gradients (this is implemented in the NETLAB package for MATLAB), and I would recommend that you use this in any (continuous) optimisation problem. It is based on line search techniques.

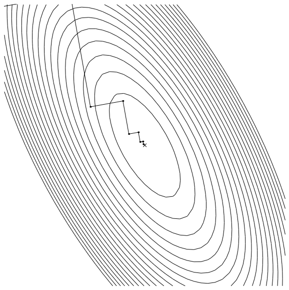


Figure A.2: Optimisation using line search along steepest descent directions. Rushing off following the steepest way downhill from a point (and continuing for a finite time in that direction) doesn't always result in the fastest way to get to the bottom!

A.1.1 Gradient descent with fixed stepsize

Locally, if we are at point $\mathbf{w}$, the quickest way to decrease E is to take a step in the direction $-\mathbf{g}(\mathbf{w})$. If we make the update equation

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta\mathbf{g}(t)$$

then we are doing gradient descent with fixed stepsize η. If η is non-infinitesimal, it is always possible that we will step over the true minimum. Making η very small guards against this, but means that the optimization process will take a very long time to reach a minimum.

A.1.2 Gradient descent with line searches

An obvious extension to the idea of gradient descent is to choose the direction of steepest descent, as indicated by the gradient $\mathbf{g}$, but to calculate the value of the step to take which most reduces the value of E when moving in that direction. This involves solving the one-dimensional problem of minimizing $E(\mathbf{w}(t) - \lambda\mathbf{g}(t))$ with respect to λ, and is known as a *line search.* That step is then taken and the process repeated again.

Finding the size of the step takes a little work; for example, you might find three points along the line such that the error at the intermediate point is less than at the other two, so that there is some minimum along the line lies between the first and second or between the second and third, and some kind of interval-halving approach can then be used to find it. (The minimum found in this way, just as with any sort of gradient-descent algorithm, may not be a global minimum of course.) There are several variants of this theme. Notice that if the step size is chosen to reduce E as much as it can in that direction, then no further improvement in E can be made by moving in that direction for the moment. Thus the next step will have no component in that direction; that is, the next step will be at right angles to the one just taken. This can lead to zig-zag type behaviour in the optimisation, see fig(A.2).

A.1.3 Eigenvalues and Eigenvectors

NEED TO TALK ABOUT SVD

One of the most important concepts in linear algebra is that of eigenvalues and eigenvectors. In a sense, the eigenvectors of the matrix correspond to the natural coordinate system, in which the transformation can be most easily understood.

We call e an eigenvector and λ the corresponding eigenvalue of the matrix A if

$$Ae = \lambda e \tag{A.1.1}$$

Eigenvectors are usually scaled to have unit length, though not always.

Finding Eigenvalues

In general, for an $(n \times n)$ dimensional matrix, there are (possibly including repetitions) n eigenvalues, each with a corresponding eigenvector. We can reform equation (A.1.1) as

$$(A - \lambda \boldsymbol{I})\, e = 0 \tag{A.1.2}$$

This is simply a linear equation, and we can use our knowledge of linear algebra to solve this. We can write equation (A.1.2) as $Be = 0$, where $B = A - \lambda \boldsymbol{I}$. If B has an inverse, then the solution is $e = B^{-1}0 = 0$, since any matrix multiplying the zero vector gives zero. For any non-trivial solution to the problem $Be = 0$, we therefore need B to be non-invertible. This is equivalent to the condition that the B has zero determinant.

In other words, λ is an eigenvalue of A if

$$\det(A - \lambda \boldsymbol{I}) = 0 \tag{A.1.3}$$

This is known as the characteristic equation. This determinant equation will be a polynomial of degree n (the dimension of A), and the resulting equation is known as the characteristic polynomial.

Once we have found an eigenvalue, the corresponding eigenvector can be found by substituting this value for λ in equation (A.1.1) and solving for e.

Find the eigenvalues and eigenvectors of $\begin{pmatrix} 1 & 3 \\ 4 & 2 \end{pmatrix}$

The characteristic equation is

$$\begin{vmatrix} 1-\lambda & 3 \\ 4 & 2-\lambda \end{vmatrix} = 0 \tag{A.1.4}$$

Evaluating the determinant, this is

$$(1-\lambda)(2-\lambda) - 3 \times 4 = 0 \tag{A.1.5}$$

Factorising this quadratic, we have

$$(\lambda - 5)(\lambda + 2) = 0 \tag{A.1.6}$$

so that the solutions are $\lambda = 5$ and $\lambda = -2$. To find the eigenvectors, we consider each eigenvalue separately.
$\lambda = 5$

$$\begin{pmatrix} -4 & 3 \\ 4 & -3 \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{A.1.7}$$

This is

$$-4e_1 + 3e_2 = 0, 4e_1 - 3e_2 = 0 \tag{A.1.8}$$

(Note that the second equation is simply the negative of the first. Such redundancies will always occur in the solution of these eigenvector equations). This means that the eigenvector points along $\begin{pmatrix} 1 \\ \frac{4}{3} \end{pmatrix}$ so that the unit eigenvector is (by normalising this vector)

$$\begin{pmatrix} \frac{3}{5} \\ \frac{4}{5} \end{pmatrix} \tag{A.1.9}$$

Similarly, we obtain for the other eigenvalue $\lambda = -2$

$$\begin{pmatrix} \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \tag{A.1.10}$$

Eigenspaces

Sometimes there is not just a single vector which is an eigenvector for a matrix. It may be that there is a whole space of such vectors.

For example, lets find the eigenvalues and eigenvectors of the matrix:

$$\begin{pmatrix} 0 & 0 & -2 \\ 1 & 2 & 1 \\ 1 & 0 & 3 \end{pmatrix} \tag{A.1.11}$$

First we need to solve the eigenvalue problem

$$\begin{vmatrix} -\lambda & 0 & -2 \\ 1 & 2-\lambda & 1 \\ 1 & 0 & 3-\lambda \end{vmatrix} = 0 \tag{A.1.12}$$

The characteristic polynomial is

$$\lambda^3 - 5\lambda^2 + 8\lambda - 4 = 0 \tag{A.1.13}$$

We need to solve this equation. One good way at guessing a solution is to use the fact that, (as is always true), the product of the roots of a polynomial is equal to the constant in the polynomial. If we guess that the roots are integers then, since their product must be equal to 4, the integer roots could be 1, 2, or 4. Substituting in $\lambda = 1$ in the polynomial, we see this is a solution. Similarly, $\lambda = 2$ is a solution. Since the product of the roots must be equal to four, the only possibility for the other root is that it must also be equal to 2. That is, there is a repeated eigenvalue, namely $\lambda = 2$.

Lets try to find the eigenvectors for $\lambda = 1$:

$$\begin{pmatrix} -1 & 0 & -2 \\ 1 & 1 & 1 \\ 1 & 0 & 2 \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{A.1.14}$$

From the top line, we have $e_1 = -2e_2$. From the second line, $e_3 = -(e_1 + e_2)$. If we set $e_1 = s$, then $e_2 = -s/2$ and $e_3 = -s/2$. This means that the eigenspace is spanned by the basis

$$\begin{pmatrix} 1 \\ -1/2 \\ -1/2 \end{pmatrix} \tag{A.1.15}$$

For $\lambda = 2$, the situation is a little more complicated

$$\begin{pmatrix} -2 & 0 & -2 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{A.1.16}$$

As we can see, if we set $e_1 = s$, then we have the condition $e_3 = -s$. However, e_2 is left free. This means that any vector e in the space

$$s \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} + t \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \tag{A.1.17}$$

is an eigenvector of A with eigenvalue 2. We can characterise this eigenspace by a set of basis vectors that span the space, namely

$$\begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}, \text{and} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \tag{A.1.18}$$

Relating the eigenvalues to the determinant

The determinant of an $n \times n$ matrix A is the product of its eigenvalues,

$$\det A = \prod_{i=1}^{n} \lambda_i \tag{A.1.19}$$

The orthogonality of eigenvectors of a Symmetric Matrix

Consider two distinct eigenvectors of a symmetric matrix A. In otherwords, the eigenvectors of a symmetric matrix form a mutually orthogonal set.

Outer Product Representation of a symmetric matrix

$$A = \sum_{i=1}^{n} \lambda_i e_i e_i^T \tag{A.1.20}$$

where λ_i is the eigenvalue of eigenvector e_i.

Appendix B Graph Terminology:

B.1 Graphs : Basic Concepts and Definitions

Parts of this section are taken from *Expert Systems and Probabilistic Network Models* by E. Castillo, J. Gutierrez, and A. Hadi (Springer, 1997), and also *An introduction to bayesian networks* by F.V.Jensen (Springer 1996). Both are excellent introductions to the field.

B.2 Basic Concepts and Definitions

Graph

A graph $G = (X, L)$ is defined by two sets X and L, where X is a finite set of nodes $X = \{X_1, X_2, \dots X_n\}$ and L is a set of links (also called connections), that is, a subset of all possible ordered pairs of distinct nodes.

For example in fig(B.1a), we have

$$X = \{A, B, C, D, E, F\}, \qquad L = \{L_{AD}, L_{DB}, L_{BC}, L_{FD}, L_{EF}, L_{DE}\}$$

Directed and Undirected Links

Let $G = (X, L)$ be a graph. When $L_{ij} \in L$ and $L_{ij} \in L$, the link $L_{ji} \notin L$ is called a directed link. Otherwise, the link is called undirected.

Directed and Undirected Graphs

A graph in which all the links are directed is called a directed graph and a graph in which all the links are undirected is called an undirected graph. A graph containing both directed and undirected links is called a chain graph.

Adjacency Set

Given a graph $G = (X, L)$ and a node X_i, the adjacency set of X_i is the set of nodes directly attainable from X_i, that is $Adj\,(X_i) = \{X_j | L_{ij} \in L\}$. That is, in a directed graph, the adjacency set of a node G is the set of nodes that G points to. In an undirected graph, it is the set of nodes that G is directly connected to.

The directed graph in fig(B.1a) has adjacency sets

$$\begin{aligned} &Adj\,\{A\} = \{D\}, Adj\,\{B\} = \{C\}, Adj\,\{C\} = \emptyset, \\ &\qquad\qquad Adj\,\{D\} = \{B, E\}, Adj\,\{E\} = \{F\}, Adj\,\{F\} = \{D\}, \end{aligned}$$

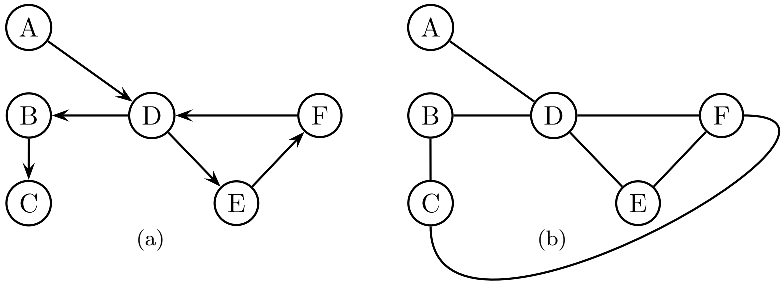


Figure B.1: (a) A directed graph (b) An undirected graph

The undirected graph in fig(B.1b) has adjacency sets

$$Adj\{A\} = \{B, E\}, Adj\{B\} = \{C, D\}, Adj\{C\} = \{B, F\},$$
$$Adj\{D\} = \{A, B, E, F\}, Adj\{E\} = \{D, F\}, Adj\{F\} = \{D, E, C\},$$

B.2.1 Undirected Graphs

Complete set

A subset of nodes S of a graph G is said to be complete if there are links between every pair of nodes in S. Correspondingly, a graph is complete if there is a link between every pair of nodes in the graph.

For example, fig(B.2.1) is a complete graph of 5 nodes.

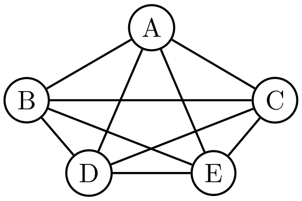


Figure B.2: A complete graph with 5 nodes

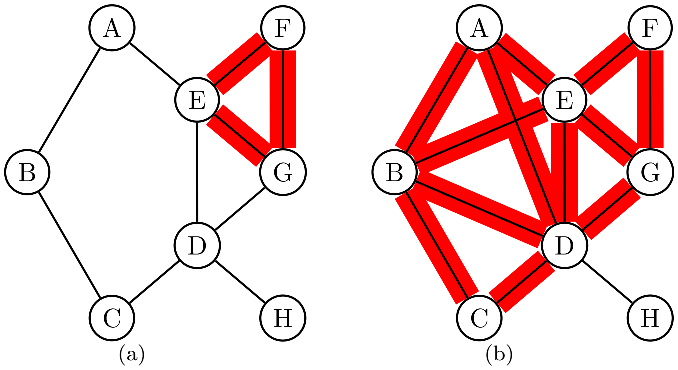


Figure B.3: Two different graphs and their associated cliques. (a) Here there are two cliques of size 3. (b) Here there are several cliques of size three and one clique of size 4. Links belonging to cliques of size two have not been coloured.

Clique

A complete set of nodes C is called a clique if C is not a subset of another complete set. In other words, a clique cannot be expanded since the expanded set would not be complete.

For example, in fig(B.3a), the cliques are

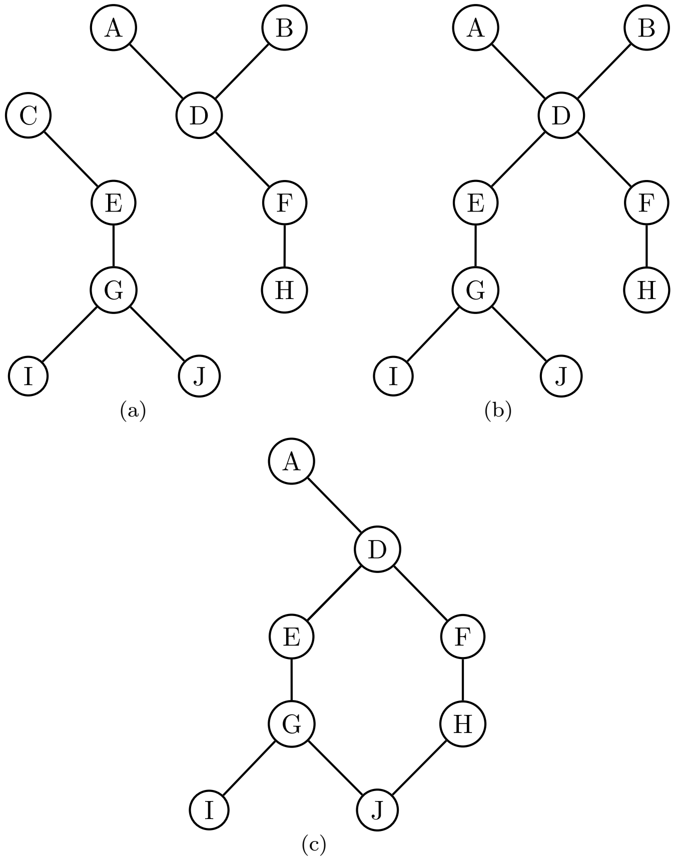
$$\{E, F, G\} \qquad \{D, G, E\} \{A, B\} \{B, C\} \{C, D\} \{D, H\} \{A, E\}$$

$DEFG$ is not a clique since there is no connection between D and F.

In fig(B.3b), the cliques are

$$\{A, B, D, E\}, \{B, C, D\}, \{D, H\}, \{D, E, G\}, \{E, F, G\}$$

ABE is not a clique since this is a complete subset of a larger complete subset, namely $ABED$.



(a)

(b)

(c)

Figure B.4: (a) A disconnected graph. (b) A Tree (c) A loopy graph

Loop

A loop is a closed path (a series of nodes with intersecting adjacency sets) in an undirected graph.

For example, $B - D - F - C - D$ in fig(B.1b) is a loop.

Neighbours of a node

The set of nodes adjacent to a node X_i in an undirected graph is referred to as the neighbours of X_i, $Nbr\left(X_i\right) = \left\{X_j | X_j \in Adj\left(X_i\right)\right\}$

So, in an undirected graph, the neighbours of a node are identical to the adjacency set of that node.

Connected Undirected Graphs

An undirected graph is connected if there is at least one path between every pair on nodes. Otherwise, the graph is disconnected.

For example, fig(B.4a) is an disconnected graph.

Tree

A connected undirected graph is a tree if for every pair of nodes there exists a unique path.

For example fig(B.4b) is a tree.

Multiply-connected or Loopy Graphs

A connected undirected graph is multiply-connected (or loopy) if it contains at least one pair of nodes that are joined by more than one path, or equivalently, if it contains at least one loop.

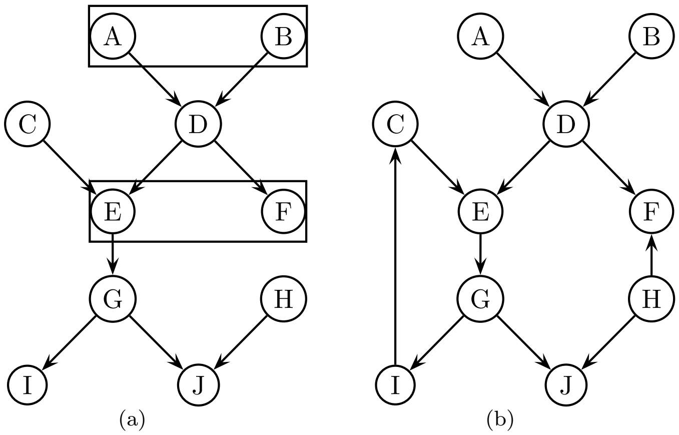


Figure B.5: (a) Parents (A, B) and children (E, F) of node D (b) $C \to E \to G \to I \to C$ is a cycle.

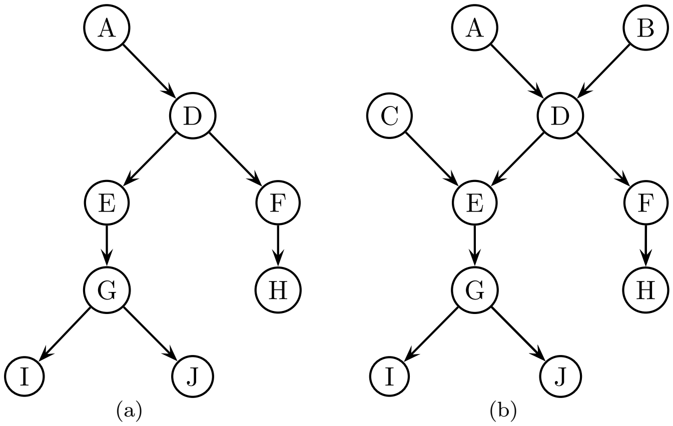


Figure B.6: (a) A Simple Tree (b) A polytree

For example fig(B.4c) is a loopy graph.

B.2.2 Directed Graphs

Parents and Children When there is a directed link from X_i to X_j, then X_i is said to be a parent of X_j, and X_j is a child of X_i.

For example, in fig(B.5a) the parents of node E are C and D. Node E has only one child, node G.

Cycle A cycle is a closed directed path in a directed graph. If a graph contains no cycles, it is called acyclic.

For example C, in fig(B.5b) $C \to E \to G \to I \to C$ is a cycle. The nodes along the path connected D, F, H, J, G, E, D do not form a cycle since the directions of the links are not consistent.

Family The set consisting of a node and its parents is called the family of the node.

Simple Trees and Polytrees A directed tree is called a simple tree if every node has at most one parent. Otherwise it is called a polytree.

For example, in fig(B.6a), the graph is a simple tree, whereas fig(B.6b) is a polytree.

Markov Blanket The Markov blanket of a node A consists of the parents of node A, the children of node A, and the parents of the children of node A. For example, in fig(B.6b), the Markov Blanket of node D is $\{A, B, C, E, F\}$.

Appendix C Some Standard Distributions:

Gamma Distribution

$$p(x) = \frac{1}{\beta\Gamma(\gamma)}\left(\frac{x-x^0}{\beta}\right)^{\gamma-1} e^{-\frac{1}{\beta}\left(x-x^0\right)}, \qquad x \geq x^0, \beta > 0 \qquad \text{(C.0.1)}$$

γ is called the shape parameter, x^0 the location parameter, β is the scale parameter and

$$\Gamma(a) = \int_0^\infty t^{a-1}e^{-t}dt$$

The parameters are related to the mean and variance as follows

$$\gamma = \left(\frac{\mu}{s}\right)^2, \qquad \beta = \frac{s^2}{\mu}$$

where μ is the mean of the distribution and s is the standard deviation. In practice we often encounter a zero location distribution

$$p(x) = \frac{1}{Z}x^{\gamma-1}e^{-x/\beta}, \qquad x \geq 0, \beta > 0$$

Need some plots.... Basically a unimodal distribution with a bump which can be placed close to the location x^0.

Dirichlet Distribution

The Dirichlet distribution is a distribution on probability distributions:

$$p(\boldsymbol{\alpha}) = \frac{1}{Z(u)}\delta\left(\sum_{i=1}^{Q}\alpha_i - 1\right)\prod_{q=1}^{Q}\alpha_q^{u_q-1}$$

where

$$Z(u) = \frac{\prod_{q=1}^{Q}\Gamma(u_q)}{\Gamma\left(\sum_{q=1}^{Q}u_q\right)}$$

It is conventional to denote the distribution as

$$\text{Dirichlet}(\boldsymbol{\alpha}|u)$$

The parameter u controls how strongly the mass of the distribution is pushed to the corners of the simplex. Setting $u_q = 1$ for all q corresponds to a uniform distribution.

In the binary case $Q = 2$, this is also called a Beta distribution.

Appendix D Bounds on Convex Functions

D.1 Kullback Leibler Divergence $KL(q||p)$

The KL diveregence $KL(q||p)$ measures the “difference” between distributions q and p, fig(D.1a). In many ways, this is a natural measure to use, and is well motivated from Information theoretic arguments.

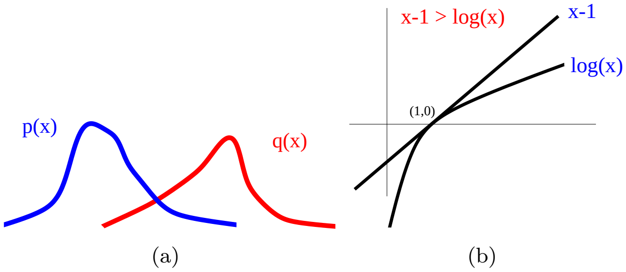


Figure D.1: (a) The probability density functions for two different distributions $p(x)$ and $q(x)$. We would like to numerically characterise the difference between these distributions. (b) A simple linear bound on the logarithm enables us to define a useful distance measure between distributions (see text).

For two distributions $q(x)$ and $p(x)$, it is defined as

$$KL(q||p) \equiv \langle \log q(x) - \log p(x) \rangle_{q(x)}$$

where the notation $\langle f(x) \rangle_{r(x)}$ denotes average of the function $f(x)$ with respect to the distribution $r(x)$. For a continuous variable, this would be $\langle f(x) \rangle_{r(x)} = \int f(x) r(x) dx$, and for a discrete variable, we would have $\langle f(x) \rangle_{r(x)} = \sum_x f(x) r(x)$. The advantage of this notation is that much of the following holds independent of whether the variables are discrete or continuous.

$KL(q||p) \geq 0$ The KL divergence is always ≥ 0. To see this, consider the following simple linear bound on the function $\log(x)$ (see fig(D.1b)):

$$\log(x) \leq x - 1$$

Replacing x by $p(x)/q(x)$ in the above bound

$$\frac{p(x)}{q(x)} - 1 \geq \log \frac{p(x)}{q(x)} \Rightarrow p(x) - q(x) \geq q(x) \log p(x) - q(x) \log q(x)$$

Now integrate (or sum in the case of discrete variables) both sides. Using $\int p(x) dx = 1$, $\int q(x) dx = 1$, and rearranging gives

$$\int \{ q(x) \log q(x) - q(x) \log p(x) \} \, dx \equiv \langle \log q(x) - \log p(x) \rangle_{q(x)} \geq 0$$

Furthermore, one can show that the KL divergence is zero if and only if the two distributions are exactly the same.

D.1.1 Jensen vs Variational Jensen

Imagine we wish to bound the quantity

$$J = \log \int_x p(x) f(x)$$

where $f(x) \geq 0$. Then Jensen's bound is

$$J \geq \int_x p(x) \log f(x)$$

By considering a distribution $r(x) = p(x)f(x)/Z$, and variational distribution $q(x)$, we have

$$J \geq \int_x (-q(x) \log q(x) + q(x) \log p(x) + q(x) \log f(x))$$

Interestingly, one can write this as

$$J \geq -KL(q(x)||p(x)) + \int_x q(x) \log f(x)$$

Furthermore, one can recognise the second term as a KL divergence too:

$$J \geq -KL(q(x)||p(x)) - KL(q(x)||f(x)) - H(q(x))$$

Clearly these two bounds are equal for the setting $q(x) = p(x)$ (this reminds me of the difference in sampling routines, between sampling from the prior and sampling from a distribution which is more optimal for calculating the average). The first term encourages q to be close to p. The second encourages q to be close to f, and the third encourages q to be sharply peaked.

Appendix E Positive Definite Matrices and Kernel Functions:

Outer Product Representation of Matrices

If we have a symmetric matric $A = A^T$, then (a fundamental theorem) then two eigenvectors e^i and e^j of A are orthorgonal $(e^i)^T e^j = 0$ if their eigenvalues λ_i and λ_j are different. This is easy to show by considering:

$$Ae^i = \lambda_i e^i$$

Hence

$$(e^j)^T A e^i = \lambda_i (e^j)^T e^i$$

But, since A is symmetric, the LHS is equivalent to

$$((e^j)^T A)e^i = (Ae^j)^T e^i = \lambda_j (e^j)^T e^i$$

Hence we must have $\lambda_i (e^j)^T e^i = \lambda_j (e^j)^T e^i$. If $\lambda_i \neq \lambda_j$, the only way this condition can be satisfied is if $(e^j)^T e^i = 0$ – ie, the eigenvectors are orthogonal.

This means also that we can represent a symmetric matrix as

$$A = \sum_i \lambda_i e^i (e^i)^T$$

Hermitian Matrices

A square matrix is called Hermitian if

$$A = A^{T*}$$

where $*$ denotes the conjugate operator. The reader can easily show that for Hermitian matrices, the eigenvectors form an orthogonal set, and furthermore that the eigenvalues are *real*.

Positive Definite Matrix

A matrix A is positive definite if and only if

$$y^T A y > 0$$

for any real $y \neq 0$.

Using the above result, this means that

$$y^T A y = \sum_i \lambda_i y^T e^i (e^i)^T y = \sum_i \lambda_i (y^T e^i)^2$$

This is clearly greater than zero if and only if all the eigenvalues are positive. This is therefore an equivalent condition for positive definiteness.

Kernel

Kernels are closely related to the idea of Gaussian Processes. See [63] for an excellent review. Consider a collection of points, $x^1, \ldots, x^P$, and a symmetric function $K(x^i, x^j) = K(x^j, x^i)$. We can use this function to define a symmetric matrix K with elements

$$[K]_{ij} = K(x^i, x^j)$$

The function K is called a Kernel if the corresponding matrix (for all P) is positive definite.

Eigenfunctions

$$\int_x K(x', x)\phi_a(x) = \lambda_a \phi_a(x')$$

By an analogous argument that proves the theorem of linear algebra above, the eigenfunctions are orthogonal:

$$\int_x \phi_a(x)\phi_b^*(x) = \delta_{ab}$$

where $\phi^*(x)$ is the complex conjugate of $\phi(x)$[1]. From the previous results, we know that a symmetric real matrix K must have a decomposition in terms eigenvectors with positive, real eigenvalues. Since this is to be true for any dimension of matrix, it suggests that we need the (real symmetric) kernel function itself to have a decomposition (provided the eigenvalues are countable)

$$K(x^i, x^j) = \sum_\mu \lambda_\mu \phi_\mu(x^i)\phi_\mu^*(x^j)$$

since then

$$\sum_{i,j} y_i K(x^i, x^j) y_j = \sum_{i,j,\mu} \lambda_\mu y_i \phi_\mu(x^i)\phi_\mu^*(x^j) y_j = \sum_\mu \lambda_\mu \underbrace{(\sum_i y_i \phi_\mu(x^i))}_{z_i} \underbrace{(\sum_i y_i \phi_\mu^*(x^i))}_{z_i^*}$$

which is greater than zero if the eigenvalues are all positive (since for complex z, $zz^* \geq 0$).

If the eigenvalues are uncountable (which happens when the domain of the kernel is unbounded), the appropriate decomposition is

$$K(x^i, x^j) = \int \lambda(s)\phi(x^i, s)\phi^*(x^j, s)ds$$

How can we check if K is a Kernel?

It is not always straightforward to check if K is a kernel. If we can show that K can be expressed in the form

$$K(x, x') = \sum_i \psi_i(x)\psi_i(x')$$

[1] This definition of the inner product is useful, and natural particularly in the context of translation invariant kernels. We are free to define the inner product, but this conjugate form is often the most useful.

for real functions $\psi_i(x)$, then K is a Kernel. Note that we do not require the functions to be orthogonal. However, we know that if K is a Kernel, then an alternative expansion in terms of orthogonal functions exists. More generally, K is a Kernel if it has a representation:

$$K(x,x') = \int \psi(x,s)\psi(x',s)ds$$

for real functions $\psi(x,s)$.

A second approach would be to find the eigenfunctions $e_a(x)$ of K,

$$\int k(x,x')e_a(x')dx' = \lambda_a e_a(x)$$

and show that the eigenvalues λ_a are all positive.

Translation Invariance

In the case that $k(x,x') = k(x-x')$ (note that we do not impose symmetry here of the function $k(x,x')$, so that this section holds for more general functions than kernels), the function is called stationary. The eigenproblem becomes

$$\int k(x-x')e(x')dx' = \lambda e(x)$$

In this case, the LHS is in the form of a convolution. It makes sense therefore to take the Fourier Transform:

$$\tilde{k}\tilde{e} = \lambda\tilde{e}$$

This means that $\tilde{e}$ is a delta function, and that therefore the eigenfunctions are e^{isx}.

When k is Hermitian, it has an expansion in terms of eigenfunctions. Note that the form of the conjugate expansion automatically ensures that the Kernel is translation invariant, since $\phi(x+a)\phi^*(x'+a) = e^{is(x+a-x'-a)} = \phi(x)\phi^*(x')$. (Indeed, this shows generally why Fourier representations are central to systems which possess translation invariance). Exercise: what happens if we take the Laplace transform of a translation invariant operator?

Application to the Squared Exponential Kernel

For the squared exponential kernel, the Fourier Transform of the kernel is a Gaussian, and hence $\lambda(s) = e^{-s^2}$. Hence, we have a representation of the kernel as

$$e^{-(x-x')^2} = \int e^{-s^2}e^{isx}e^{-isx'}ds = \int e^{-s^2+is(x-x')}ds \tag{E.0.1}$$

The reader may verify that this is indeed an identity by considering that this is just a rewriting of the Fourier Transform of a Gaussian:

$$e^{-w^2} = \int e^{-x^2}e^{iwx}dx$$

The form of the representation equation (E.0.1) of the kernel verifies that it is indeed a kernel, since the eigenvalues are all positive.

Blochner's Theorem

For a stationary process (the Kernel is translation invariant), we can define the so-called correlation function which, for $K(0) > 0$, is

$$\rho(x) = \frac{K(x)}{K(0)}$$

Blochner's theorem states that $\rho(x)$ is positive semidefinite, if and only if it is the characteristic function of a variable ω,

$$\rho(x) = \int e^{ix\omega} f(\omega) d\omega$$

for a probability distribution $f(\omega)$. This means that we can prove positive semidefiniteness of K by checking that its Fourier Transform is non-negative.

For stationary Kernels which are also isotropic,($K(x) = K(|x|)$), one can show that ρ must be representable as a Hankel transform[63].

Mercer's Theorem

(Not really sure why this is useful!!) For x, x' being in a *bounded* domain of R^N, then any positive (semi) definite Kernel has an expansion of the form

$$K(x, x') = \sum_i \phi_i(x)\phi_i(x')$$

Note that this may possibly be an infinite series. Also, note that there is no requirement that the functions be orthogonal.

Aside : it is interesting to think about the *unbounded* case since, for example, the conditions of Mercer's theorem do not apply to the squared exponential Kernel $e^{-(x-x')^2}$. From Blochner's Theorem, or using simple Fourier Analysis, we can indeed form an expansion of this Kernel in terms of an integral representation of complex orthogonal functions. What is interesting about the squared exponential case is that we can indeed also find a Mercer representation (exercise!), even though the conditions of the theorem do not hold.

Appendix F Approximating Integrals:

The Laplace Approximation

Consider a distribution

$$p(w) = \frac{1}{Z}e^{-E(w)}$$

In many cases (due to some form of the asymptotics), distributions will often tend to be come rather sharply peaked. The Laplace distribution aims to make a Gaussian approximation of $p(w)$. The Laplace approach is a simple *perturbation* expansion, which assumes that the Gaussian is sharply peaked around some value w^*. If we find

$$w^* = \arg\min_w E(w)$$

then a natural approximation is to make a Taylor expansion up to second order:

$$E(w) \approx E(w^*) + (w - w^*) \nabla E|_{w^*} + \frac{1}{2} (w - w^*)^T H (w - w^*)$$

where $H \equiv \nabla\nabla E(w)|_{w^*}$. At the optimum, $\nabla E|_{w^*} = 0$, and an approximation of the distribution is given by the Gaussian

$$p^*(w) = \frac{1}{Z^*}e^{-\frac{1}{2}(w-w^*)^T H(w-w^*)}$$

which has mean w^* and covariance H^{-1}. This means that the normalisation constant is $Z^* = \det 2\pi H^{-1}$. Similarly, we can use the above expansion to estimate the integral

$$\int_w e^{-E(w)} \approx \int_w e^{-E(w^*)-\frac{1}{2}(w-w^*)^T H(w-w^*)}$$

Hence

$$\int_w e^{-E(w)} \approx e^{-E(w^*)}\sqrt{\det 2\pi H^{-1}}$$

Note that although the Laplace approximation essentially fits a Gaussian to a distribution, it is not necessarily the best Gaussian approximation. Other criteria, such as a variational fit, are sometimes more powerful ways of approximating the integral. However, arguably the major benefit of Laplace's method is its speed and simplicity.

Appendix G Inference with Gaussian Random Variables

We have two variables X and Y. Imagine the X models the position of an object in the world (in one dimension) and Y is an observation, say in a camera, of the position of the object in the camera. A camera calibration procedure tells us the relationship between X and Y; in our case we assume

$$Y = 2X + 8 + N_y$$

where N_y is some Gaussian measurement noise with zero mean and variance 1. Thus our model for $P(y|x)$ is

$$P(y|x) = \frac{1}{\sqrt{2\pi}} \exp\{-\frac{1}{2}(y - 2x - 8)^2\}.$$

Also we assume that $x \sim N(0, 1/\alpha)$ so that

$$P(x) = \sqrt{\frac{\alpha}{2\pi}} \exp - \frac{\alpha x^2}{2}.$$

Given this, we want to infer the distribution of X given that $Y = y$. To do this we compute the mean and covariance of $(X, Y)^T$, and then condition on $Y = y$. The mean vector is easily calculated as

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix} = \begin{pmatrix} 0 \\ 8 \end{pmatrix}.$$

For the covariance matrix, we have that $\mathrm{var}(X) = 1/\alpha$. For $\mathrm{var}(Y)$ we find

$$\mathrm{var}(Y) = E[(Y - \mu_y)^2] = E[(2X + N_y)^2] = \frac{4}{\alpha} + 1,$$

and for $\mathrm{covar}(XY)$ we find

$$\mathrm{covar}XY = E[(X - \mu_x)(Y - \mu_y)] = E[X(2X + N_y)] = \frac{2}{\alpha}$$

and thus

$$\Sigma = \begin{pmatrix} 1/\alpha & 2/\alpha \\ 2/\alpha & 4/\alpha + 1 \end{pmatrix}.$$

Given a vector of random variables split into two parts X_1 and X_2 with

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}$$

and

$$\Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}$$

the general expression for obtaining the conditional distribution of X_1 given X_2, is

$$\boldsymbol{\mu}^c_{1|2} = \boldsymbol{\mu}_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \boldsymbol{\mu}_2),$$

$$\Sigma^c_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}.$$

Applying this to the case above, we obtain

$$\mu_{x|y} = 0 + \frac{2}{\alpha}.\frac{\alpha}{4+\alpha}(y-8) = \frac{2}{4+\alpha}(y-8)$$

and

$$\text{var}(x|y) = \frac{1}{\alpha} - \frac{2}{\alpha}.\frac{\alpha}{4+\alpha}.\frac{2}{\alpha} = \frac{1}{4+\alpha}.$$

The obvious estimator of X is $(y-8)/2$, which is obtained from inverting the dependence between y and x on the assumption that the noise is zero. We see that this is obtained in the limit $\alpha \to 0$, which corresponds to an improper prior on X with infinite variance. For non-zero α, the effect is to "shrink" $\mu_{x|y}$ towards zero, which corresponds to the information in the prior on X that zero is its most likely value. Note that if $\alpha \to \infty$, which corresponds to being certain at the outset that $X = 0$, then this information overwhelms the information coming from the observation, and in this limit $\mu_{x|y} = 0$. Notice also that the posterior variance $1/(4+\alpha)$ is smaller than the prior variance $1/\alpha$.

G.1 Gaussian Conditioning

For a joint Gaussian distribution over the vectors x and y with means μ_x, μ_y and covariance elements Σ_{xx},Σ_{xy},Σ_{yy}, the conditional $p(x|y)$ is a Gaussian with mean $\mu_x + \Sigma_{xy}\Sigma_{yy}^{-1}(y - \mu_y)$ and covariance $\Sigma_{xx} - \Sigma_{xy}\Sigma_{yy}^{-1}\Sigma_{yx}$.

G.1.1 Conditioning as a linear system reversal

For a joint distribution $p(x, y)$, consider the conditional $p(x|y)$.

We claim that the statistics of $x|y$ can be written using a linear system of the form

$$x = \overleftarrow{M}y + \overleftarrow{\eta}$$

where $\overleftarrow{\eta} \sim \mathcal{N}\left(\overleftarrow{m}, \overleftarrow{\Sigma}\right)$. To show this, we need to make the statistics of x match.

$$\mu_x = \overleftarrow{M}\mu_y + \overleftarrow{\mu}$$

$$\Sigma_{xx} = \overleftarrow{M}\Sigma_{yy}\overleftarrow{M}^{\mathsf{T}} + \overleftarrow{\Sigma}$$

$$\Sigma_{xy} = \overleftarrow{M}\Sigma_{yy}$$

From the last equation, we have

$$\overleftarrow{M} = \Sigma_{xy}\Sigma_{yy}^{-1}$$

which gives

$$\overleftarrow{\Sigma} = \Sigma_{xx} - \overleftarrow{M}\Sigma_{yy}\overleftarrow{M}^{\mathsf{T}}, \text{or} \overleftarrow{\Sigma} = \Sigma_{xx} - \Sigma_{xy}\Sigma_{yy}^{-1}\Sigma_{yx}$$

and

$$\overleftarrow{\mu} = \mu_x - \overleftarrow{M}\mu_y, \text{or} \overleftarrow{\mu} = \mu_x - \Sigma_{xy}\Sigma_{yy}^{-1}\mu_y$$

This is just a restatement of the previous results, but shows how it may be interpreted as a linear system.

G.2 Gaussian Propagation

Let y be linearly related to x through $y = Mx + \eta$, where $\eta \sim \mathcal{N}(\mu, \Sigma)$, and $x \sim \mathcal{N}(\mu_x, \Sigma_x)$. Then $p(y) = \int_x p(y|x)p(x)$ is a Gaussian with mean $M\mu_x + \mu$ and covariance $M\Sigma_x M^{\mathsf{T}} + \Sigma$.

Bibliography

[1] Y. Takane and M.A. Hunter. Constrained Principal Component Analysis : A Comprehensive Theory. *Applicable Algebra in Engineering, Communication and Computing*, 12(5):391–419, 2001.

[2] G. H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins, 1996.

[3] S. Hawking. Does god play dice? http://www.hawking.org.uk/lectures/dice.html, 2003.

[4] G. Shafer. What is probability? http://www.glennshafer.com/assets/downloads/article4

[5] E.T. Jaynes. *Probability Theory : The Logic of Science*. Cambridge University Press, 2003.

[6] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of Royal Statistical Society, Series B*, 50(2):157–224, 1988.

[7] J. Whittaker. *Graphical Models in Applied Multivariate Statistics*. Wiley: Chichester, 1990.

[8] J. Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[9] M. I. Jordan. *Learning in Graphical Models*. MIT Press, 1998.

[10] F. V. Jensen. *Bayesian Networks and Decision Graphs*. Springer Verlag, 2001.

[11] S. L. Lauritzen. *Graphical Models*. Oxford University Press, 1996.

[12] R. M. Neal. Connectionist Learning of Belief Networks. *Artificial Intelligence*, 56:71–113, 1992.

[13] J. Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, 2000.

[14] K. Binder. Spin Glasses: Experimental Facts, Theoretical Concepts and Open Questions . *Rev. Mod. Phys*, 58:901, 1986.

[15] J. Besag. Spatial Interactions and the Statistical Analysis of Lattice Systems. *Journal of the Royal Statistical Society, Series B*, 36(2):192–236, 1974.

[16] J. Moussouris. Gibbs and Markov Random Systems with Constraints. *Journal of Statistical Physics*, 10:11–33, 1974.

[17] R. Dechter. Bucket Elimination: A unifying framework for probabilistic inference algorithms. *Uncertainty in Artificial Intelligence*, 1996.

[18] J. M. Gutierrez E. Castillo and A. S. Hadi. *Expert Systems and Probabilistic Network Models*. Springer Verlag, 1997.

[19] M. A. Srinvas and R. J. McEliece. The Generalised Distributive Law. *IEEE Transactions of Information Theory*, 46(2):325–343, 2000.

[20] R. E. Bellman. *Dynamic Programming.* Dover Publications, Incorporated, 2003.

[21] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984.

[22] F.V. Jensen and F. Jensen. Optimal Junction Trees. *Uncertainty in Artificial Intelligence*, 1994.

[23] V. Lepar and P.P. Shenoy. A Comparison of Lauritzen-Spiegelhalter, Hugin, and Shenoy-Shafer Architectures for Computing Marginals of Probability Distributions. *Uncertainty in Artificial Intelligence*, 1998.

[24] R. M. Neal and G. E. Hinton. A View of the EM Algorithm That Justifies Incremental, Sparse, and Other Variants. In M.J. Jordan, editor, *Learning in Graphical Models*, chapter 1. MIT Press, 1998.

[25] G. McLachlan and T. Krishnan. *The EM Algorithm and Extensions.* John Wiley and Sons, 1997.

[26] R. Salakhutdinov, S. Roweis, and Z. Ghahramani. Optimization with em and expectation-conjugate-gradient. *Intl. Conf. on Machine Learning ICML*, 2003.

[27] W. Wiegerinck and T. Heskes. IPF for discrete chain factor graphs. *Uncertainty in Artificial Intelligence (UAI 2002)*, 2002.

[28] T. Jebara and A. Pentland. On reversing Jensen's inequality. *Advances in Neural Information Processing Systems (NIPS 13)*, 2000.

[29] T. Minka. A comparison of numerical optimizers for logistic regression. Technical report, Microsoft Research, 2003.

[30] J. O. Berger. *Statistical Decision Theory and Bayesian Analysis.* Springer, Second edition, 1985.

[31] D. J. C. MacKay. *Information Theory, Inference and Learning Algorithms.* Cambridge University Press, 2003.

[32] G.E.P. Box and G.C. Tiao. *Bayesian Inference in Statistical Analysis.* Addison–Wesley, Reading, MA, 1973.

[33] D. J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 1992.

[34] T. J. Loredo. From Laplace To Supernova Sn 1987A: Bayesian Inference In Astrophysics. In P.F. Fougere, editor, *Maximum Entropy and Bayesian Methods*, pages 81–142. Kluwer, 1990.

[35] D.J.C. MacKay. *Information Theory, Inference and Learning Algorithms.* Cambridge University Press, 2003.

[36] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[37] M. E. Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211:244, 2001.

[38] M. E. Tipping and C.M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society, Series B*, 61(3):611–622, 1999.

[39] M. Tipping and C.M. Bishop. Mixtures of probabilistic principal component analysers. *Neural Computation*, 11(2):443482, 1999.

[40] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. of the IEEE*, 77(2), 1989.

[41] T. Heskes and O. Zoeter. Expectation Propagation for approximate inference in dynamic Bayesian networks. In A. Darwiche and N. Friedman, editors, *Uncertainty in Artificial Intelligence*, pages 216–223, 2002.

[42] B. Deylon. Remarks on Linear and Nonlinear Filtering. *IEEE Transactions on Information Theory*, 41(1):317–322, 1995.

[43] H. E. Rauch, G. Tung, and C. T. Striebel. Maximum Likelihood estimates of linear dynamic systems. *American Institute of Aeronautics and Astronautics Journal (AIAAJ)*, 3(8):1445–1450, 1965.

[44] Y. Bar-Shalom and Xiao-Rong Li. *Estimation and Tracking : Principles, Techniques and Software*. Artech House, Norwood, MA, 1998.

[45] Z. Ghahramani and G.E. Hinton. Variational learning for switching state-space models. *Neural Computation*, 12(4):963–996, 1998.

[46] U. Lerner, R. Parr, D. Koller, and G. Biswas. Bayesian Fault Detection and Diagnosis in Dynamic Systems. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AIII-00)*, pages 531–537, 2000.

[47] G. Kitagawa. The Two-Filter Formula for Smoothing and an implementation of the Gaussian-sum smoother. *Annals of the Institute of Statistical Mathematics*, 46(4):605–623, 1994.

[48] C-J. Kim and C. R. Nelson. *State-Space models with regime switching*. MIT Press, 1999.

[49] V. Pavlovic, J.M. Rehg, and J. MacCormick. Learning switching linear models of human motion. In *Advances in Neural Information Processing systems (NIPS 13)*, pages 981–987, 2001.

[50] A. T. Cemgil, B. Kappen, and D. Barber. A Generative Model for Music Transcription. *IEEE Transactions on Audio, Speech and Language Processing*, 14(2):679 – 694, 2006.

[51] S. Chib and M. Dueker. Non-Markovian Regime Switching with Endogenous States and Time-Varying State Strengths. *Econometric Society 2004 North American Summer Meetings 600*, 2004.

[52] U. N. Lerner. *Hybrid Bayesian Networks for Reasoning about Complex Systems*. PhD thesis, Stanford University, 2002.

[53] O. Zoeter. *Monitoring non-linear and switching dynamical systems*. PhD thesis, Radboud University Nijmegen, 2005.

[54] T. Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, MIT Media Lab, 2001.

[55] D. L. Alspach and H. W. Sorenson. Nonlinear Bayesian Estimation Using Gaussian Sum Approximations. *IEEE Transactions on Automatic Control*, 17(4):439–448, 1972.

[56] D. M. Titterington, A. F. M. Smith, and U. E. Makov. *Statistical analysis of finite mixture distributions*. Wiley, 1985.

[57] M. Verhaegen and P. Van Dooren. Numerical Aspects of Different Kalman Filter Implementations. *IEEE Transactions of Automatic Control*, 31(10):907–917, 1986.

[58] C-J. Kim. Dynamic linear models with Markov-switching. *Journal of Econometrics*, 60:1–22, 1994.

[59] Y. Bar-Shalom and T. E. Fortmann. *Tracking and Data Association*. Academic Press, 1988.

[60] E. B. Sudderth, A. T. Ihler, and A. S. Freeman, W. T.and Willsky. Nonparametric belief propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 605–612, 2003.

[61] A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[62] B. Mesot and D. Barber. Switching linear dynamical systems for noise robust speech recognition. IDIAP-RR 08, 2006.

[63] M. Seeger. Gaussian Processes for Machine Learning. *International Journal of Neural Systems*, 14(2):69–106, 2004.