Saarland University

Summary

Probabilistic Machine Learning

Winter 2020/2021

Lecturer:
Prof. Dr. Isabel Valera

Documentation:
Christian Schmidt

Contents

0.1 Introduction

REVIEW

doi:10.1038/nature14541

Probabilistic machine learning and artificial intelligence

Zoubin Ghahramani[1]

How can a machine learn from experience? Probabilistic modelling provides a framework for understanding what learning is, and has therefore emerged as one of the principal theoretical and practical approaches for designing machines that learn from data acquired through experience. The probabilistic framework, which describes how to represent and manipulate uncertainty about models and predictions, has a central role in scientific data analysis, machine learning, robotics, cognitive science and artificial intelligence. This Review provides an introduction to this framework, and discusses some of the state-of-the-art advances in the field, namely, probabilistic programming, Bayesian optimization, data compression and automatic model discovery.

The key idea behind the probabilistic framework to machine learning is that learning can be thought of as inferring plausible models to explain observed data. A machine can use such models to make predictions about future data, and take decisions that are rational given these predictions. Uncertainty plays a fundamental part in all of this. Observed data can be consistent with many models, and therefore which model is appropriate, given the data, is uncertain. Similarly, predictions about future data and the future consequences of actions are uncertain. Probability theory provides a framework for modelling uncertainty.

This Review starts with an introduction to the probabilistic approach to machine learning and Bayesian inference, and then discusses some of the state-of-the-art advances in the field. Many aspects of learning and intelligence crucially depend on the careful probabilistic representation of uncertainty. Probabilistic approaches have only recently become a mainstream approach to artificial intelligence[1], robotics[2] and machine learning[3,4]. Even now, there is controversy in these fields about how important it is to fully represent uncertainty. For example, advances using deep neural networks to solve challenging pattern-recognition problems such as speech recognition[5], image classification[6,7], and prediction of words in text[8], do not overtly represent the uncertainty in the structure or parameters of those neural networks. However, my focus will not be on these types of pattern-recognition problems, characterized by the availability of large amounts of data, but on problems for which uncertainty is really a key ingredient, for example where a decision may depend on the amount of uncertainty.

I highlight five areas of current research at the frontier of probabilistic machine learning, emphasizing areas that are of broad relevance to scientists across many fields: probabilistic programming, which is a general framework for expressing probabilistic models as computer programs and which could have a major impact on scientific modelling; Bayesian optimization, which is an approach to globally optimizing unknown functions; probabilistic data compression; automating the discovery of plausible and interpretable models from data; and hierarchical modelling for learning many related models, for example for personalized medicine or recommendation. Although considerable challenges remain, the coming decade promises substantial advances in artificial intelligence and machine learning based on the probabilistic framework.

Probabilistic modelling and representing uncertainty

At the most basic level, machine learning seeks to develop methods for computers to improve their performance at certain tasks on the basis of observed data. Typical examples of such tasks might include detecting pedestrians in images taken from an autonomous vehicle, classifying gene-expression patterns from leukaemia patients into subtypes by clinical outcome, or translating English sentences into French. However, as I discuss, the scope of machine-learning tasks is even broader than these pattern classification or mapping tasks, and can include optimization and decision making, compressing data and automatically extracting interpretable models from data.

Data are the key ingredients of all machine-learning systems. But data, even so-called big data, are useless on their own until one extracts knowledge or inferences from them. Almost all machine-learning tasks can be formulated as making inferences about missing or latent data from the observed data — I will variously use the terms inference, prediction or forecasting to refer to this general task. Elaborating the example mentioned, consider classifying people with leukaemia into one of the four main subtypes of this disease on the basis of each person's measured gene-expression patterns. Here, the observed data are pairs of gene-expression patterns and labelled subtypes, and the unobserved or missing data to be inferred are the subtypes for new patients. To make inferences about unobserved data from the observed data, the learning system needs to make some assumptions; taken together these assumptions constitute a model. A model can be very simple and rigid, such as a classic statistical linear regression model, or complex and flexible, such as a large and deep neural network, or even a model with infinitely many parameters. I return to this point in the next section. A model is considered to be well defined if it can make forecasts or predictions about unobserved data having been trained on observed data (otherwise, if the model cannot make predictions it cannot be falsified, in the sense of the philosopher Karl Popper's proposal for evaluating hypotheses, or as the theoretical physicist Wolfgang Pauli said the model is "not even wrong"). For example, in the classification setting, a well-defined model should be able to provide predictions of class labels for new patients. Since any sensible model will be uncertain when predicting unobserved data, uncertainty plays a fundamental part in modelling.

There are many forms of uncertainty in modelling. At the lowest level, model uncertainty is introduced from measurement noise, for example, pixel noise or blur in images. At higher levels, a model may have many parameters, such as the coefficients of a linear regression, and there is uncertainty about which values of these parameters will be good at predicting new data. Finally, at the highest levels, there is

[1]Department of Engineering, University of Cambridge, Trumpington Street, Cambridge CB2 1PZ, UK.

often uncertainty about even the general structure of the model: is linear regression or a neural network appropriate, if the latter, how many layers should it have, and so on.

The probabilistic approach to modelling uses probability theory to express all forms of uncertainty[9]. Probability theory is the mathematical language for representing and manipulating uncertainty[10], in much the same way as calculus is the language for representing and manipulating rates of change. Fortunately, the probabilistic approach to modelling is conceptually very simple: probability distributions are used to represent all the uncertain unobserved quantities in a model (including structural, parametric and noise-related) and how they relate to the data. Then the basic rules of probability theory are used to infer the unobserved quantities given the observed data. Learning from data occurs through the transformation of the prior probability distributions (defined before observing the data), into posterior distributions (after observing data). The application of probability theory to learning from data is called Bayesian learning (Box 1).

Apart from its conceptual simplicity, there are several appealing properties of the probabilistic framework for machine intelligence. Simple probability distributions over single or a few variables can be composed to form the building blocks of larger, more complex models. The dominant paradigm in machine learning over the past two decades for representing such compositional probabilistic models has been graphical models[11], with variants including directed graphs (also known as Bayesian networks and belief networks), undirected graphs (also known as Markov networks and random fields), and mixed graphs with both directed and undirected edges (Fig. 1). As discussed later, probabilistic programming offers an elegant way of generalizing graphical models, allowing a much richer representation of models. The compositionality of probabilistic models means that the behaviour of these building blocks in the context of the larger model is often much easier to understand than, say, what will happen if one couples a non-linear dynamical system (for example, a recurrent neural network) to another. In particular, for a well-defined probabilistic model, it is always possible to generate data from the model; such 'imaginary' data provide a window into the 'mind' of the probabilistic model, helping us to understand both the initial prior assumptions and what the model has learned at any later stage.

Probabilistic modelling also has some conceptual advantages over alternatives because it is a normative theory for learning in artificially intelligent systems. How should an artificially intelligent system represent and update its beliefs about the world in light of data? The Cox axioms define some desiderata for representing beliefs; a consequence of these axioms is that 'degrees of belief', ranging from 'impossible' to 'absolutely certain', must follow all the rules of probability theory[10,12,13]. This justifies the use of subjective Bayesian probabilistic representations in artificial intelligence. An argument for Bayesian representations in artificial intelligence that is motivated by decision theory is given by the Dutch book theorem. The argument rests on the idea that the strength of beliefs of an agent can be assessed by asking the agent whether it would be willing to accept bets at various odds (ratios of payoffs). The Dutch book theorem states that unless an artificial intelligence system's (or human's, for that matter) degrees of beliefs are consistent with the rules of probability it will be willing to accept bets that are guaranteed to lose money[14]. Because of the force of these and many other arguments on the importance of a principled handling of uncertainty for intelligence, Bayesian probabilistic modelling has emerged not only as the theoretical foundation for rationality in artificial intelligence systems, but also as a model for normative behaviour in humans and animals[15–18] (but see refs 19, 20 for a discussion), and much research is devoted to understanding how neural circuitry may be implementing Bayesian inference[21,22].

Although conceptually simple, a fully probabilistic approach to machine learning poses a number of computational and modelling challenges. Computationally, the main challenge is that learning involves marginalizing (summing out) all the variables in the model except for the variables of interest (Box 1). Such high-dimensional sums and integrals are generally computationally hard, in the sense that for many models

BOX 1
Bayesian machine learning

There are two simple rules that underlie probability theory: the sum rule:

$$P(x) = \sum_{y \in Y} P(x,y)$$

and the product rule:

$$P(x,y) = P(x)P(y \mid x).$$

Here x and y correspond to observed or uncertain quantities, taking values in some sets X and Y, respectively. For example, x and y might relate to the weather in Cambridge and London, respectively, both taking values in the set $X=Y=\{\text{rainy,cloudy,sunny}\}$. $P(x)$ corresponds to the probability of x, which can be either a statement about the frequency of observing a particular value, or a subjective belief about it. $P(x,y)$ is the joint probability of observing x and y, and $P(y|x)$ is the probability of y conditioned on observing the value of x. The sum rule states that the marginal of x is obtained by summing (or integrating for continuous variables) the joint over y. The product rule states that the joint can be decomposed as the product of the marginal and the conditional. Bayes rule is a corollary of these two rules:

$$P(y \mid x) = \frac{P(x \mid y)P(y)}{P(x)} = \frac{P(x \mid y)P(y)}{\sum_{y \in Y} P(x,y)}$$

We can apply probability theory to machine learning by replacing the symbols above: we replace x by D to denote the observed data, we replace y by θ to denote the unknown parameters of a model, and we condition all terms on m, the class of probabilistic models we are considering. For learning, we thus get:

$$P(\theta \mid D,m) = \frac{P(D \mid \theta,m)P(\theta \mid m)}{P(D \mid m)}$$

where $P(D|\theta,m)$ is the likelihood of parameters θ in model m, $P(\theta|m)$ is the prior probability of θ and $P(\theta|D, m)$ is the posterior of θ given data D.

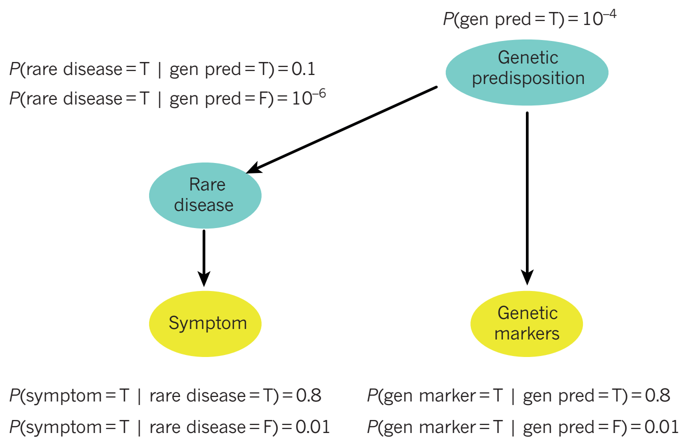
For example, the data D might be a time series of hourly observations of the weather in Cambridge and London, and the model might attempt to capture the joint weather patterns at both locations over successive hours, with parameters θ modelling correlations over time and space. Learning is the transformation of prior knowledge or assumptions about the parameters $P(\theta|m)$, through the data D, into posterior knowledge about the parameters, $P(\theta|D,m)$. This posterior is now the prior to be used for future data. A learned model can be used to predict or forecast new unseen test data, D_{test}, by simply applying the sum and product rule to get the prediction:

$$P(D_{test} \mid D,m) = \int P(D_{test} \mid \theta, D, m)P(\theta \mid D,m)d\theta$$

Finally, different models can be compared by applying Bayes rule at the level of m:

$$P(m \mid D) = \frac{P(D \mid m)P(m)}{P(D)}$$
$$P(D \mid m) = \int P(D \mid \theta,m)P(\theta \mid m)d\theta$$

The term $P(D|m)$ is the marginal likelihood or model evidence, and implements a preference for simpler models known as Bayesian Ockham's razor[78,96,97].



P(gen pred = T) = 10^−4
Genetic predisposition
P(rare disease = T | gen pred = T) = 0.1
P(rare disease = T | gen pred = F) = 10^−6
Rare disease
Symptom
Genetic markers
P(symptom = T | rare disease = T) = 0.8
P(symptom = T | rare disease = F) = 0.01
P(gen marker = T | gen pred = T) = 0.8
P(gen marker = T | gen pred = F) = 0.01

Figure 1 | Bayesian inference. A simple example of Bayesian inference applied to a medical diagnosis problem. Here the problem is diagnosing a rare disease using information from the patient's symptoms and, potentially, the patient's genetic marker measurements, which indicate predisposition (gen pred) to this disease. In this example, all variables are assumed to be binary. T, true; F, false. The relationships between variables are indicated by directed arrows and the probability of each variable given other variables they directly depend on is also shown. Yellow nodes denote measurable variables, whereas green nodes denote hidden variables. Using the sum rule (Box 1), the prior probability of the patient having the rare disease is: $P(\text{rare disease} = \text{T}) = P(\text{rare disease} = \text{T}|\text{gen pred} = \text{T})\, p(\text{gen pred} = \text{T}) + p(\text{rare disease} = \text{T}|\text{gen pred} = \text{F})\, p(\text{gen pred} = \text{F}) = 1.1 \times 10^{-5}$. Applying Bayes rule we find that for a patient observed to have the symptom, the probability of the rare disease is: $p(\text{rare disease} = \text{T}|\text{symptom} = \text{T}) = 8.8 \times 10^{-4}$, whereas for a patient observed to have the genetic marker (gen marker) it is $p(\text{rare disease} = \text{T}|\text{gen marker} = \text{T}) = 7.9 \times 10^{-4}$. Assuming that the patient has both the symptom and the genetic marker the probability of the rare disease increases to $p(\text{rare disease} = \text{T}|\text{symptom} = \text{T}, \text{gen marker} = \text{T}) = 0.06$. Here, we have shown fixed, known model parameters, that is, the numbers $\theta = (10^{-4}, 0.1, 10^{-6}, 0.8, 0.01, 0.8, 0.01)$. However, both these parameters and the structure of the model (the presence or absence of arrows and additional hidden variables) could be learned from a data set of patient records using the methods in Box 1.

there is no known polynomial time algorithm for performing them exactly. Fortunately, a number of approximate integration algorithms have been developed, including Markov chain Monte Carlo (MCMC) methods, variational approximations, expectation propagation and sequential Monte Carlo[23–26]. It is worth noting that computational techniques are one area in which Bayesian machine learning differs from much of the rest of machine learning: for Bayesian researchers the main computational problem is integration, whereas for much of the rest of the community the focus is on optimization of model parameters. However, this dichotomy is not as stark as it appears: many gradient-based optimization methods can be turned into integration methods through the use of Langevin and Hamiltonian Monte Carlo methods[27,28], while integration problems can be turned into optimization problems through the use of variational approximations[24]. I revisit optimization in a later section.

The main modelling challenge for probabilistic machine learning is that the model should be flexible enough to capture all the properties of the data required to achieve the prediction task of interest. One approach to addressing this challenge is to develop a prior distribution that encompasses an open-ended universe of models that can adapt in complexity to the data. The key statistical concept underlying flexible models that grow in complexity with the data is non-parametrics.

Flexibility through non-parametrics

One of the lessons of modern machine learning is that the best predictive performance is often obtained from highly flexible learning systems, especially when learning from large data sets. Flexible models can make better predictions because to a greater extent they allow data to 'speak for themselves'. (But note that all predictions involve assumptions and therefore the data are never exclusively 'speaking for themselves'.) There are essentially two ways of achieving flexibility. The model could have a large number of parameters compared with the data set (for example, the neural network recently used to achieve translations of English and French sentences that approached the accuracy of state-of-the-art methods is a probabilistic model with 384 million parameters[29]). Alternatively, the model can be defined using non-parametric components.

The best way to understand non-parametric models is through comparison to parametric ones. In a parametric model, there are a fixed, finite number of parameters, and no matter how much training data are observed, all the data can do is set these finitely many parameters that control future predictions. By contrast, non-parametric approaches have predictions that grow in complexity with the amount of training data, either by considering a nested sequence of parametric models with increasing numbers of parameters or by starting out with a model with infinitely many parameters. For example, in a classification problem, whereas a linear (parametric) classifier will always make predictions using a linear boundary between classes, a non-parametric classifier can learn a non-linear boundary whose shape becomes more complex with more data. Many non-parametric models can be derived starting from a parametric model and considering what happens as the model grows to the limit of infinitely many parameters[30]. Clearly, fitting a model with infinitely many parameters to finite training data would result in 'overfitting', in the sense that the model's predictions might reflect quirks of the training data rather than regularities that can be generalized to test data. Fortunately, Bayesian approaches are not prone to this kind of overfitting since they average over, rather than fit, the parameters (Box 1). Moreover, for many applications we have such huge data sets that the main concern is underfitting from the choice of an overly simplistic parametric model, rather than overfitting.

A full discussion of Bayesian non-parametrics is outside the scope of this Review (see refs 9, 31, 32 for this), but it is worth mentioning a few of the key models. Gaussian processes are a very flexible non-parametric model for unknown functions, and are widely used for regression, classification, and many other applications that require inference on functions[33]. Consider learning a function that relates the dose of some chemical to the response of an organism to that chemical. Instead of modelling this relationship with, say, a linear parametric function, a Gaussian process could be used to learn directly a non-parametric distribution of non-linear functions consistent with the data. A notable example of a recent application of Gaussian processes is GaussianFace, a state-of-the-art approach to face recognition that outperforms humans and deep-learning methods[34]. Dirichlet processes are a non-parametric model with a long history in statistics[35] and are used for density estimation, clustering, time-series analysis and modelling the topics of documents[36]. To illustrate Dirichlet processes, consider an application to modelling friendships in a social network, where each person can belong to one of many communities. A Dirichlet process makes it possible to have a model whereby the number of inferred communities (that is, clusters) grows with the number of people[37]. Dirichlet processes have also been used for clustering gene-expression patterns[38,39]. The Indian buffet process (IBP)[40] is a non-parametric model that can be used for latent feature modelling, learning overlapping clusters, sparse matrix factorization, or to non-parametrically learn the structure of a deep network[41]. Elaborating the social network modelling example, an IBP-based model allows each person to belong to some subset of a large number of potential communities (for example, as defined by different families, workplaces, schools, hobbies, and so on) rather than a single community, and the probability of friendship between two people depends on the number of overlapping communities they have[42]. In this case, the latent features of each person correspond to the communities, which are not assumed to be observed directly. The IBP can be thought of as a way of endowing Bayesian non-parametric models with 'distributed representations', as popularized in the neural network literature[43]. An interesting link between Bayesian non-parametrics and neural networks is that, under fairly general conditions, a neural network with

```
statesmean = [-1, 1, 0]  # Emission parameters.
initial    = Categorical([1.0/3, 1.0/3, 1.0/3]) # Prob distr of state[1].
trans      = [Categorical([0.1, 0.5, 0.4]), Categorical([0.2, 0.2, 0.6]),
              Categorical([0.15, 0.15, 0.7])]   # Trans distr for each state.
data       = [Nil, 0.9, 0.8, 0.7, 0, -0.025, -5, -2, -0.1, 0, 0.13]

@model hmm begin # Define a model hmm.
 states = Array(Int, length(data))
 @assume(states[1] ~ initial)
 for i = 2:length(data)
   @assume(states[i] ~ trans[states[i-1]])
   @observe(data[i]  ~ Normal(statesmean[states[i]], 0.4))
 end
 @predict states
end
```

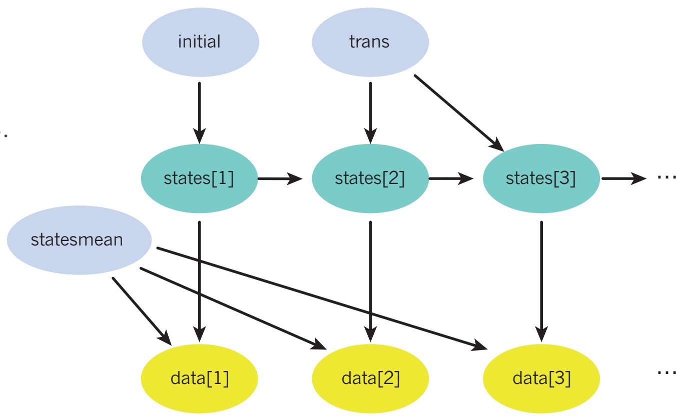


Figure 2 | Probabilistic programming. A probabilistic program in Julia (left) defining a simple three-state hidden Markov model (HMM), inspired by an example in ref. 62. The HMM is a widely used probabilistic model for sequential and time-series data, which assumes the data were obtained by transitioning stochastically between a discrete number of hidden states[98]. The first four lines define the model parameters and the data. Here 'trans' is the 3×3 state-transition matrix, 'initial' is the initial state distribution, and 'statesmean' are the mean observations for each of the three states; actual observations are assumed to be noisy versions of this mean with Gaussian noise. The function hmm starts the definition of the HMM, drawing the sequence of states with the @assume statements, and conditioning on the observed data with the @observe statements. Finally @predict states that we wish to infer the states and data; this inference is done automatically by the universal inference engine, which reasons over the configurations of this computer program. It would be trivial to modify this program so that the HMM parameters are unknown rather than fixed. A graphical model (right) corresponding to the HMM probabilistic program showing dependencies between the parameters (blue), hidden state variables (green) and observed data (yellow). This graphical model highlights the compositional nature of probabilistic models.

infinitely many hidden units is equivalent to a Gaussian process[44]. Note that the above non-parametric components should be thought of again as building blocks, which can be composed into more complex models as described earlier. The next section describes an even more powerful way of composing models — through probabilistic programming.

Probabilistic programming

The basic idea in probabilistic programming is to use computer programs to represent probabilistic models (http://probabilistic-programming.org)[45–47]. One way to do this is for the computer program to define a generator for data from the probabilistic model, that is, a simulator (Fig. 2). This simulator makes calls to a random number generator in such a way that repeated runs from the simulator would sample different possible data sets from the model. This simulation framework is more general than the graphical model framework described previously since computer programs can allow constructs such as recursion (functions calling themselves) and control flow statements (for example, 'if' statements that result in multiple paths a program can follow), which are difficult or impossible to represent in a finite graph. In fact, for many of the recent probabilistic programming languages that are based on extending Turing-complete languages (a class that includes almost all commonly used languages), it is possible to represent any computable probability distribution as a probabilistic program[48].

The full potential of probabilistic programming comes from automating the process of inferring unobserved variables in the model conditioned on the observed data (Box 1). Conceptually, conditioning needs to compute input states of the program that generate data matching the observed data. Whereas normally we think of programs running from inputs to outputs, conditioning involves solving the inverse problem of inferring the inputs (in particular the random number calls) that match a certain program output. Such conditioning is performed by a 'universal inference engine', usually implemented by Monte Carlo sampling over possible executions of the simulator program that are consistent with the observed data. The fact that defining such universal inference algorithms for computer programs is even possible is somewhat surprising, but it is related to the generality of certain key ideas from sampling such as rejection sampling, sequential Monte Carlo methods[25] and 'approximate Bayesian computation'[49].

As an example, imagine you write a probabilistic program that simulates a gene regulatory model that relates unmeasured transcription factors to the expression levels of certain genes. Your uncertainty in each part of the model would be represented by the probability distributions used in the simulator. The universal inference engine can then condition the output of this program on the measured expression levels, and automatically infer the activity of the unmeasured transcription factors and other uncertain model parameters. Another application of probabilistic programming implements a computer vision system as the inverse of a computer graphics program[50].

There are several reasons why probabilistic programming could prove to be revolutionary for machine intelligence and scientific modelling (its potential has been noticed by US Defense Advanced Research Projects Agency, which is currently funding a major programme called Probabilistic Programming for Advancing Machine Learning). First, the universal inference engine obviates the need to manually derive inference methods for models. Since deriving and implementing inference methods is generally the most rate-limiting and bug-prone step in modelling, often taking months, automating this step so that it takes minutes or seconds will greatly accelerate the deployment of machine learning systems. Second, probabilistic programming could be potentially transformative for the sciences, since it allows for rapid prototyping and testing of different models of data. Probabilistic programming languages create a very clear separation between the model and the inference procedures, encouraging model-based thinking[51].

There are a growing number of probabilistic programming languages. BUGS[52], Stan[53], AutoBayes[54] and Infer.NET[55] allow only a restrictive class of models to be represented compared with systems based on Turing-complete languages. In return for this restriction, inference in such languages can be much faster than for the more general languages[56], such as IBAL[57], BLOG[58], Church[59], Figaro[60], Venture[61], and Anglican[62]. A major emphasis of recent work is on fast inference in general languages (see for example ref. 63). Nearly all approaches to probabilistic programming are Bayesian since it is hard to create other coherent frameworks for automated reasoning about uncertainty. Notable exceptions are systems such as Theano, which is not itself a probabilistic programming language but uses symbolic differentiation to speed up and automate optimization of parameters of neural networks and other probabilistic models[64].

Although parameter optimization is commonly used to improve probabilistic models, in the next section I will describe recent work on how probabilistic modelling can be used to improve optimization.

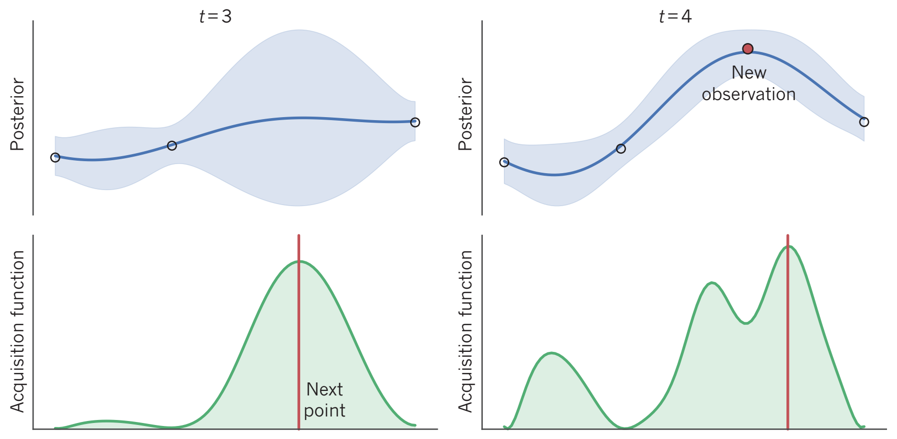


Figure 3 | Bayesian optimization. A simple illustration of Bayesian optimization in one dimension. The goal is to maximize some true unknown function *f* (not shown). Information about this function is gained by making observations (circles, top), which are evaluations of the function at specific *x* values. These observations are used to infer a posterior distribution over the function values (shown as mean, blue line; and standard deviations, blue shaded area) representing the distribution of possible functions; note that uncertainty grows away from the observations. On the basis of this distribution over functions, an acquisition function is computed (green shaded area, bottom panels), which represents the gain from evaluating the unknown function *f* at different *x* values; note that the acquisition function is high where the posterior over *f* has both high mean and large uncertainty. Different acquisition functions can be used such as 'expected improvement' or 'information gain'. The peak of the acquisition function (red line) is the best next point to evaluate, and is therefore chosen for evaluation (red dot, new observation). The left and right panels show an example of what could happen after three and four function evaluations, respectively.

Bayesian optimization

Consider the very general problem of finding the global maximum of an unknown function that is expensive to evaluate (say, evaluating the function requires performing lots of computation or conducting an experiment). Mathematically, for a function *f* on a domain *X*, the goal is to find a global maximizer x^*:

$$x^* = \arg\max_{x \in X} f(x).$$

Bayesian optimization poses this as a problem in sequential decision theory: where should one evaluate next so as most quickly to maximize *f*, taking into account the gain in information about the unknown function *f* (refs 65,66)? For example, having evaluated at three points, measuring the corresponding values of the function at those points, $[(x_1, f(x_1)), (x_2, f(x_2)), (x_3, f(x_3))]$, which point *x* should the algorithm evaluate next, and where does it believe the maximum to be? This is a classic machine-intelligence problem with a wide range of applications in science and engineering, from drug design (where the function could be the drug's efficacy) to robotics (where the function could be the speed of a robot's gait). It can be applied to any problem involving the optimization of expensive functions; the qualifier 'expensive' comes because Bayesian optimization might use substantial computational resources to decide where to evaluate next, and a trade-off for these resources has to be made with the cost of function evaluations.

The current best-performing global optimization methods maintain a Bayesian representation of the probability distribution over the uncertain function *f* being optimized, and use this uncertainty to decide where (in *X*) to query next[67–69]. In continuous spaces, most Bayesian optimization methods (Fig. 3) use Gaussian processes (as described in the section on non-parametrics) to model the unknown function. A recent high-impact application has been the optimization of the training process for machine-learning models, including deep neural networks[70]. This and related recent work[71] are further examples of the application of machine intelligence to improve machine intelligence.

There are interesting links between Bayesian optimization and reinforcement learning. Specifically, Bayesian optimization is a sequential decision problem where the decisions (choices of *x* to evaluate) do not affect the state of the system (the actual function *f*). Such state-less sequential decision problems fall under the rubric of multi-arm bandits[72], a subclass of reinforcement-learning problems. More broadly, important recent work takes a Bayesian approach to learning to control uncertain systems[73] (for a review see ref. 74). Faithfully representing uncertainty about the future outcome of actions is particularly important in decision and control problems. Good decisions rely on good representations of the probability of different outcomes and their relative payoffs.

More generally, Bayesian optimization is a special case of Bayesian numerical computation[75,76], which is re-emerging as a very active area of research (http://www.probabilistic-numerics.org), and includes topics such as solving ordinary differential equations and numerical integration. In all these cases, probability theory is being used to represent computational uncertainty; that is, the uncertainty that one has about the outcome of a deterministic computation.

Data compression

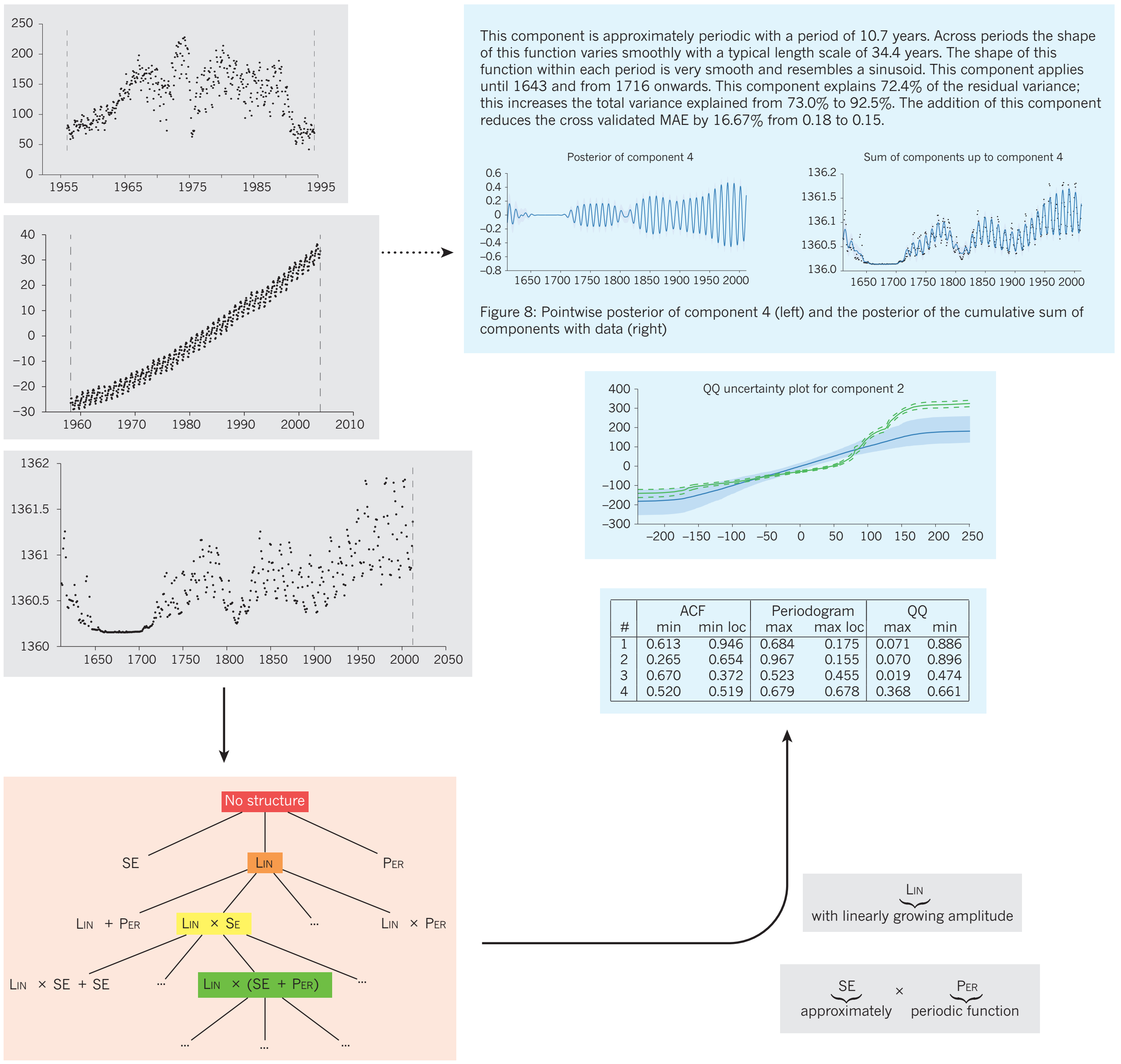
Consider the problem of compressing data so as to communicate them or store them in as few bits as possible in such a manner that the original data can be recovered exactly from the compressed data. Methods for such lossless data compression are ubiquitous in information technology, from computer hard drives to data transfer over the internet. Data compression and probabilistic modelling are two sides of the same coin, and Bayesian machine-learning methods are increasingly advancing the state-of-the-art in compression. The connection between compression and probabilistic modelling was established in the mathematician Claude Shannon's seminal work on the source coding theorem[77], which states that the number of bits required to compress data in a lossless manner is bounded by the entropy of the probability distribution of the data. All commonly used lossless data compression algorithms (for example, gzip) can be viewed as probabilistic models of sequences of symbols.

The link to Bayesian machine learning is that the better the probabilistic model one learns, the higher the compression rate can be[78]. These models need to be flexible and adaptive, since different kinds of sequences have very different statistical patterns (say, Shakespeare's plays or computer source code). It turns out that some of the world's best compression algorithms (for example, Sequence Memoizer[79] and PPM with dynamic parameter updates[80]) are equivalent to Bayesian non-parametric models of sequences, and improvements to compression are being made through a better understanding of how to learn the statistical structure of sequences. Future advances in compression will come with advances in probabilistic machine learning, including special compression methods for non-sequence data such as images, graphs and other structured objects.

Automatic discovery of interpretable models from data

One of the grand challenges of machine learning is to fully automate the process of learning and explaining statistical models from data. This is the goal of the Automatic Statistician (http://www.automaticstatistician.com), a system that can automatically discover plausible models from data, and explain what it has discovered in plain English[81]. This could be useful to almost any field of endeavour that is reliant on extracting knowledge from data. In contrast to the methods described in much of the machine-learning literature, which have been focused on extracting increasing performance improvements on pattern-recognition problems using techniques such as kernel methods, random forests or deep learning, the Automatic Statistician builds models that are composed of interpretable components, and has a principled way of representing uncertainty about model structures given the data. It also gives reasonable answers not just for big data sets, but also for small ones. Bayesian approaches provide an elegant way of trading off the complexity of the model and the complexity of the data, and probabilistic models are compositional and interpretable, as already described.

A prototype version of the Automatic Statistician takes in time-series data and automatically generates 5–15 page reports describing the model it has discovered (Fig. 4). This system is based on the idea that probabilistic building blocks can be combined through a grammar to build



This component is approximately periodic with a period of 10.7 years. Across periods the shape of this function varies smoothly with a typical length scale of 34.4 years. The shape of this function within each period is very smooth and resembles a sinusoid. This component applies until 1643 and from 1716 onwards. This component explains 72.4% of the residual variance; this increases the total variance explained from 73.0% to 92.5%. The addition of this component reduces the cross validated MAE by 16.67% from 0.18 to 0.15.

Figure 8: Pointwise posterior of component 4 (left) and the posterior of the cumulative sum of components with data (right)

#	ACF		Periodogram		QQ	
	min	min loc	max	max loc	max	min
1	0.613	0.946	0.684	0.175	0.071	0.886
2	0.265	0.654	0.967	0.155	0.070	0.896
3	0.670	0.372	0.523	0.455	0.019	0.474
4	0.520	0.519	0.679	0.678	0.368	0.661

Posterior of component 4
Sum of components up to component 4
QQ uncertainty plot for component 2
No structure
SE
LIN
PER
LIN + PER
LIN × SE
LIN × PER
LIN × SE + SE
LIN × (SE + PER)
LIN with linearly growing amplitude
SE approximately × PER periodic function

Figure 4 | The Automatic Statistician. A flow diagram describing the Automatic Statistician. The input to the system is data, in this case represented as time series (top left). The system searches over a grammar of models to discover a good interpretation of the data (bottom left), using Bayesian inference to score the models (Box 1). Components of the model discovered are translated into English phrases (bottom right). The end result is a report with text, figures and tables, describing in detail what has been inferred about the data, including a section on model checking and criticism (top right)[99,100]. Data and the report are for illustrative purposes only.

an open-ended language of models[82]. In contrast to work on equation learning (see for example ref. 83), the models attempt to capture general properties of functions (for example, smoothness, periodicity or trends) rather than a precise equation. Handling uncertainty is at the core of the Automatic Statistician; it makes use of Bayesian non-parametrics to give it the flexibility to obtain state-of-the-art predictive performance, and uses the metric marginal likelihood (Box 1) to search the space of models.

Important earlier work includes statistical expert systems[84,85] and the Robot Scientist, which integrates machine learning and scientific discovery in a closed loop with an experimental platform in microbiology to automate the design and execution of new experiments[86]. Auto-WEKA is a recent project that automates learning classifiers, making heavy use of the Bayesian optimization techniques already described[71]. Efforts to automate the application of machine-learning methods to data have recently gained momentum, and may ultimately result in artificial intelligence systems for data science.

Perspective

The information revolution has resulted in the availability of ever larger collections of data. What is the role of uncertainty in modelling such big data? Classic statistical results state that under certain regularity conditions, in the limit of large data sets the posterior distribution of the parameters for Bayesian parametric models converges to a single point around the maximum likelihood estimate. Does this mean that Bayesian probabilistic modelling of uncertainty is unnecessary if you have a lot of data?

There are at least two reasons this is not the case[87]. First, as we have seen, Bayesian non-parametric models have essentially infinitely many parameters, so no matter how many data one has, their capacity to learn should not saturate, and their predictions should continue to improve.

Second, many large data sets are in fact large collections of small data sets. For example, in areas such as personalized medicine and recommendation systems, there might be a large amount of data, but there is still a relatively small amount of data for each patient or client, respectively. To customize predictions for each person it becomes necessary to build a model for each person — with its inherent uncertainties — and to couple these models together in a hierarchy so that information can be borrowed from other similar people. We call this the personalization of models, and it is naturally implemented using hierarchical Bayesian approaches such as hierarchical Dirichlet processes[36], and Bayesian multitask learning[88,89].

Probabilistic approaches to machine learning and intelligence are a very active area of research with wide-ranging impact beyond conventional pattern-recognition problems. As I have outlined, these problems include data compression, optimization, decision making, scientific model discovery and interpretation, and personalization. The key distinction between problems in which a probabilistic approach is important and problems that can be solved using non-probabilistic machine-learning approaches is whether uncertainty has a central role. Moreover, most conventional optimization-based machine-learning approaches have probabilistic analogues that handle uncertainty in a more principled manner. For example, Bayesian neural networks represent the parameter uncertainty in neural networks[44], and mixture models are a probabilistic analogue for clustering methods[78]. Although probabilistic machine learning often defines how to solve a problem in principle, the central challenge in the field is finding how to do so in practice in a computationally efficient manner[90,91]. There are many approaches to the efficient approximation of computationally hard inference problems. Modern inference methods have made it possible to scale to millions of data points, making probabilistic methods computationally competitive with conventional methods[92–95]. Ultimately, intelligence relies on understanding and acting in an imperfectly sensed and uncertain world. Probabilistic modelling will continue to play a central part in the development of ever more powerful machine learning and artificial intelligence systems. ■

Received 12 February; accepted 21 April 2015.

1. Russell, S. & Norvig, P. *Artificial Intelligence: a Modern Approach* (Prentice–Hall, 1995).
2. Thrun, S., Burgard, W. & Fox, D. *Probabilistic Robotics* (MIT Press, 2006).
3. Bishop, C. M. *Pattern Recognition and Machine Learning* (Springer, 2006).
4. Murphy, K. P. *Machine Learning: A Probabilistic Perspective* (MIT Press, 2012).
5. Hinton, G. *et al*. Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. *IEEE Signal Process. Mag.* **29,** 82–97 (2012).
6. Krizhevsky, A., Sutskever, I. & Hinton, G. E. ImageNet classification with deep convolutional neural networks. In *Proc. Advances in Neural Information Processing Systems 25* 1097–1105 (2012).
7. Sermanet, P. *et al*. Overfeat: integrated recognition, localization and detection using convolutional networks. In *Proc. International Conference on Learning Representations* http://arxiv.org/abs/1312.6229 (2014).
8. Bengio, Y., Ducharme, R., Vincent, P. & Janvin, C. A neural probabilistic language model. *J. Mach. Learn. Res.* **3,** 1137–1155 (2003).
9. Ghahramani, Z. Bayesian nonparametrics and the probabilistic approach to modelling. *Phil. Trans. R. Soc. A* **371,** 20110553 (2013).
A review of Bayesian non-parametric modelling written for a general scientific audience.
10. Jaynes, E. T. *Probability Theory: the Logic of Science* (Cambridge Univ. Press, 2003).
11. Koller, D. & Friedman, N. *Probabilistic Graphical Models: Principles and Techniques* (MIT Press, 2009).
This is an encyclopaedic text on probabilistic graphical models spanning many key topics.
12. Cox, R. T. *The Algebra of Probable Inference* (Johns Hopkins Univ. Press, 1961).
13. Van Horn, K. S. Constructing a logic of plausible inference: a guide to Cox's theorem. *Int. J. Approx. Reason.* **34,** 3–24 (2003).
14. De Finetti, B. La prévision: ses lois logiques, ses sources subjectives. In *Annales de l'institut Henri Poincaré* [in French] **7,** 1–68 (1937).
15. Knill, D. & Richards, W. Perception as Bayesian inference (Cambridge Univ. Press, 1996).
16. Griffiths, T. L. & Tenenbaum, J. B. Optimal predictions in everyday cognition. *Psychol. Sci.* **17,** 767–773 (2006).
17. Wolpert, D. M., Ghahramani, Z. & Jordan, M. I. An internal model for sensorimotor integration. *Science* **269,** 1880–1882 (1995).
18. Tenenbaum, J. B., Kemp, C., Griffiths, T. L. & Goodman, N. D. How to grow a mind: statistics, structure, and abstraction. *Science* **331,** 1279–1285 (2011).
19. Marcus, G. F. & Davis, E. How robust are probabilistic models of higher-level cognition? *Psychol. Sci.* **24,** 2351–2360 (2013).
20. Goodman, N. D. *et al*. Relevant and robust a response to Marcus and Davis (2013). *Psychol. Sci.* **26,** 539–541 (2015).
21. Doya, K., Ishii, S., Pouget, A. & Rao, R. P. N. *Bayesian Brain: Probabilistic Approaches to Neural Coding* (MIT Press, 2007).
22. Deneve, S. Bayesian spiking neurons I: inference. *Neural Comput.* **20,** 91–117 (2008).
23. Neal, R. M. *Probabilistic Inference Using Markov Chain Monte Carlo Methods*. Report No. CRG-TR-93–1 http://www.cs.toronto.edu/~radford/review.abstract.html (Univ. Toronto, 1993).
24. Jordan, M., Ghahramani, Z., Jaakkola, T. & Saul, L. An introduction to variational methods in graphical models. *Mach. Learn.* **37,** 183–233 (1999).
25. Doucet, A., de Freitas, J. F. G. & Gordon, N. J. *Sequential Monte Carlo Methods in Practice* (Springer, 2000).
26. Minka, T. P. Expectation propagation for approximate Bayesian inference. In *Proc. Uncertainty in Artificial Intelligence 17* 362–369 (2001).
27. Neal, R. M. In *Handbook of Markov Chain Monte Carlo* (eds Brooks, S., Gelman, A., Jones, G. & Meng, X.-L.) (Chapman & Hall/CRC, 2010).
28. Girolami, M. & Calderhead, B. Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *J. R. Stat. Soc. Series B Stat. Methodol.* **73,** 123–214 (2011).
29. Sutskever, I., Vinyals, O. & Le, Q. V. Sequence to sequence learning with neural networks. In *Proc. Advances in Neural Information Processing Systems 27*, 3104–3112 (2014).
30. Neal, R. M. in *Maximum Entropy and Bayesian Methods* 197–211 (Springer, 1992).
31. Orbanz, P. & Teh, Y. W. in *Encyclopedia of Machine Learning* 81–89 (Springer, 2010).
32. Hjort, N., Holmes, C., Müller, P. & Walker, S. (eds). *Bayesian Nonparametrics* (Cambridge Univ. Press, 2010).
33. Rasmussen, C. E. & Williams, C. K. I. *Gaussian Processes for Machine Learning* (MIT Press, 2006).
This is a classic monograph on Gaussian processes, relating them to kernel methods and other areas of machine learning.
34. Lu, C. & Tang, X. Surpassing human-level face verification performance on LFW with GaussianFace. In *Proc. 29th AAAI Conference on Artificial Intelligence* http://arxiv.org/abs/1404.3840 (2015).
35. Ferguson, T. S. A Bayesian analysis of some nonparametric problems. *Ann. Stat.* **1,** 209–230 (1973).
36. Teh, Y. W., Jordan, M. I., Beal, M. J. & Blei, D. M. Hierarchical Dirichlet processes. *J. Am. Stat. Assoc.* **101,** 1566–1581 (2006).
37. Kemp, C., Tenenbaum, J. B., Griffiths, T. L., Yamada, T. & Ueda, N. Learning systems of concepts with an infinite relational model. In *Proc. 21st National Conference on Artificial Intelligence* 381–388 (2006).
38. Medvedovic, M. & Sivaganesan, S. Bayesian infinite mixture model based clustering of gene expression profiles. *Bioinformatics* **18,** 1194–1206 (2002).
39. Rasmussen, C. E., De la Cruz, B. J., Ghahramani, Z. & Wild, D. L. Modeling and visualizing uncertainty in gene expression clusters using Dirichlet process mixtures. *Trans. Comput. Biol. Bioinform.* **6,** 615–628 (2009).

40. Griffiths, T. L. & Ghahramani, Z. The Indian buffet process: an introduction and review. *J. Mach. Learn. Res.* **12,** 1185–1224 (2011).
This article introduced a new class of Bayesian non-parametric models for latent feature modelling.
41. Adams, R. P., Wallach, H. & Ghahramani, Z. Learning the structure of deep sparse graphical models. In *Proc. 13th International Conference on Artificial Intelligence and Statistics* (eds Teh, Y. W. & Titterington, M.) 1–8 (2010).
42. Miller, K., Jordan, M. I. & Griffiths, T. L. Nonparametric latent feature models for link prediction. In *Proc. Advances in Neural Information Processing Systems* 1276–1284 (2009).
43. Hinton, G. E., McClelland, J. L. & Rumelhart, D. E. in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*: *Foundations* 77–109 (MIT Press, 1986).
44. Neal, R. M. *Bayesian Learning for Neural Networks* (Springer, 1996).
This text derived MCMC-based Bayesian inference in neural networks and drew important links to Gaussian processes.
45. Koller, D., McAllester, D. & Pfeffer, A. Effective Bayesian inference for stochastic programs. In *Proc. 14th National Conference on Artificial Intelligence* 740–747 (1997).
46. Goodman, N. D. & Stuhlmüller, A. *The Design and Implementation of Probabilistic Programming Languages.* Available at http://dippl.org (2015).
47. Pfeffer, A. *Practical Probabilistic Programming* (Manning, 2015).
48. Freer, C., Roy, D. & Tenenbaum, J. B. in *Turing's Legacy* (ed. Downey, R.) 195–252 (2014).
49. Marjoram, P., Molitor, J., Plagnol, V. & Tavaré, S. Markov chain Monte Carlo without likelihoods. *Proc. Natl Acad. Sci. USA* **100,** 15324–15328 (2003).
50. Mansinghka, V., Kulkarni, T. D., Perov, Y. N. & Tenenbaum, J. Approximate Bayesian image interpretation using generative probabilistic graphics programs. In *Proc. Advances in Neural Information Processing Systems 26* 1520–1528 (2013).
51. Bishop, C. M. Model-based machine learning. *Phil. Trans. R. Soc. A* **371,** 20120222 (2013).
This article is a very clear tutorial exposition of probabilistic modelling.
52. Lunn, D. J., Thomas, A., Best, N. & Spiegelhalter, D. WinBUGS — a Bayesian modelling framework: concepts, structure, and extensibility. *Stat. Comput.* **10,** 325–337 (2000).
This reports an early probabilistic programming framework widely used in statistics.
53. Stan Development Team. *Stan Modeling Language Users Guide and Reference Manual, Version 2.5.0*. http://mc-stan.org/ (2014).
54. Fischer, B. & Schumann, J. AutoBayes: a system for generating data analysis programs from statistical models. *J. Funct. Program*. **13,** 483–508 (2003).
55. Minka, T. P., Winn, J. M., Guiver, J. P. & Knowles, D. A. *Infer.NET 2.4*. http://research.microsoft.com/infernet (Microsoft Research, 2010).
56. Wingate, D., Stuhlmüller, A. & Goodman, N. D. Lightweight implementations of probabilistic programming languages via transformational compilation. In *Proc. International Conference on Artificial Intelligence and Statistics* 770–778 (2011).
57. Pfeffer, A. IBAL: a probabilistic rational programming language. In *Proc. International Joint Conference on Artificial Intelligence* 733–740 (2001).
58. Milch, B. *et al.* BLOG: probabilistic models with unknown objects. In *Proc. 19th International Joint Conference on Artificial Intelligence* 1352–1359 (2005).
59. Goodman, N., Mansinghka, V., Roy, D., Bonawitz, K. & Tenenbaum, J. Church: a language for generative models. In *Proc. Uncertainty in Artificial Intelligence 22* 23 (2008).
This is an influential paper introducing the Turing-complete probabilistic programming language Church.
60. Pfeffer, A. *Figaro: An Object-Oriented Probabilistic Programming Language*. Tech. Rep. (Charles River Analytics, 2009).
61. Mansinghka, V., Selsam, D. & Perov, Y. Venture: a higher-order probabilistic programming platform with programmable inference. Preprint at http://arxiv.org/abs/1404.0099 (2014).
62. Wood, F., van de Meent, J. W. & Mansinghka, V. A new approach to probabilistic programming inference. In *Proc. 17th International Conference on Artificial Intelligence and Statistics* 1024–1032 (2014).
63. Li, L., Wu, Y. & Russell, S. J. *SWIFT: Compiled Inference for Probabilistic Programs*. Report No. UCB/EECS-2015–12 (Univ. California, Berkeley, 2015).
64. Bergstra, J. *et al*. Theano: a CPU and GPU math expression compiler. In *Proc. 9th Python in Science Conference* http://conference.scipy.org/proceedings/scipy2010/ (2010).
65. Kushner, H. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *J. Basic Eng.* **86,** 97–106 (1964).
66. Jones, D. R., Schonlau, M. & Welch, W. J. Efficient global optimization of expensive black-box functions. *J. Glob. Optim*. **13,** 455–492 (1998).
67. Brochu, E., Cora, V. M. & de Freitas, N. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. Preprint at http://arXiv.org/abs/1012.2599 (2010).
68. Hennig, P. & Schuler, C. J. Entropy search for information-efficient global optimization. *J. Mach. Learn. Res*. **13,** 1809–1837 (2012).
69. Hernández-Lobato, J. M., Hoffman, M. W. & Ghahramani, Z. Predictive entropy search for efficient global optimization of black-box functions. In *Proc. Advances in Neural Information Processing Systems* 918–926 (2014).
70. Snoek, J., Larochelle, H. & Adams, R. P. Practical Bayesian optimization of machine learning algorithms. In *Proc. Advances in Neural Information Processing Systems* 2960–2968 (2012).
71. Thornton, C., Hutter, F., Hoos, H. H. & Leyton-Brown, K. Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In *Proc. 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 847–855 (2013).
72. Robbins, H. Some aspects of the sequential design of experiments. *Bull. Amer. Math. Soc.* **55,** 527–535 (1952).
73. Deisenroth, M. P. & Rasmussen, C. E. PILCO: a model-based and data-efficient approach to policy search. In *Proc. 28th International Conference on Machine Learning* 465–472 (2011).
74. Poupart, P. in *Encyclopedia of Machine Learning* 90–93 (Springer, 2010).
75. Diaconis, P. in *Statistical Decision Theory and Related Topics IV* 163–175 (Springer, 1988).
76. O'Hagan, A. Bayes-Hermite quadrature. *J. Statist. Plann. Inference* **29,** 245–260 (1991).
77. Shannon, C. & Weaver, W. *The Mathematical Theory of Communication* (Univ. Illinois Press, 1949).
78. MacKay, D. J. C. *Information Theory, Inference, and Learning Algorithms* (Cambridge Univ. Press, 2003).
79. Wood, F., Gasthaus, J., Archambeau, C., James, L. & Teh, Y. W. The sequence memoizer. *Commun. ACM* **54,** 91–98 (2011).
This article derives a state-of-the-art data compression scheme based on Bayesian nonparametric models.
80. Steinruecken, C., Ghahramani, Z. & MacKay, D. J. C. Improving PPM with dynamic parameter updates. In *Proc. Data Compression Conference* (in the press).
81. Lloyd, J. R., Duvenaud, D., Grosse, R., Tenenbaum, J. B. & Ghahramani, Z. Automatic construction and natural-language description of nonparametric regression models. In *Proc. 28th AAAI Conference on Artificial Intelligence* Preprint at: http://arxiv.org/abs/1402.4304 (2014).
Introduces the Automatic Statistician, translating learned probabilistic models into reports about data.
82. Grosse, R. B., Salakhutdinov, R. & Tenenbaum, J. B. Exploiting compositionality to explore a large space of model structures. In *Proc. Conference on Uncertainty in Artificial Intelligence* 306–315 (2012).
83. Schmidt, M. & Lipson, H. Distilling free-form natural laws from experimental data. *Science* **324,** 81–85 (2009).
84. Wolstenholme, D. E., O'Brien, C. M. & Nelder, J. A. GLIMPSE: a knowledge-based front end for statistical analysis. *Knowl. Base. Syst.* **1,** 173–178 (1988).
85. Hand, D. J. Patterns in statistical strategy. In *Artificial Intelligence and Statistics* (ed Gale, W. A.) (Addison-Wesley Longman, 1986).
86. King, R. D. *et al*. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature* **427,** 247–252 (2004).
87. Welling, M. *et al*. Bayesian inference with big data: a snapshot from a workshop. *ISBA Bulletin* **21,** https://bayesian.org/sites/default/files/fm/bulletins/1412.pdf (2014).
88. Bakker, B. & Heskes, T. Task clustering and gating for Bayesian multitask learning. *J. Mach. Learn. Res*. **4,** 83–99 (2003).
89. Houlsby, N., Hernández-Lobato, J. M., Huszár, F. & Ghahramani, Z. Collaborative Gaussian processes for preference learning. In *Proc. Advances in Neural Information Processing Systems 26* 2096–2104 (2012).
90. Russell, S. J. & Wefald, E. *Do the Right Thing: Studies in Limited Rationality* (MIT Press, 1991).
91. Jordan, M. I. On statistics, computation and scalability. *Bernoulli* **19,** 1378–1390 (2013).
92. Hoffman, M., Blei, D., Paisley, J. & Wang, C. Stochastic variational inference. *J. Mach. Learn. Res*. **14,** 1303–1347 (2013).
93. Hensman, J., Fusi, N. & Lawrence, N. D. Gaussian processes for big data. In *Proc. Conference on Uncertainty in Artificial Intelligence* 244 (UAI, 2013).
94. Korattikara, A., Chen, Y. & Welling, M. Austerity in MCMC land: cutting the Metropolis-Hastings budget. In *Proc. 31th International Conference on Machine Learning* 181–189 (2014).
95. Paige, B., Wood, F., Doucet, A. & Teh, Y. W. Asynchronous anytime sequential Monte Carlo. In *Proc. Advances in Neural Information Processing Systems 27* 3410–3418 (2014).
96. Jefferys, W. H. & Berger, J. O. Ockham's Razor and Bayesian Analysis. *Am. Sci.* **80,** 64–72 (1992).
97. Rasmussen, C. E. & Ghahramani, Z. Occam's Razor. In *Neural Information Processing Systems 13* (eds Leen, T. K., Dietterich, T. G., & Tresp, V.) 294–300 (2001).
98. Rabiner, L. R. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* **77,** 257–286 (1989).
99. Gelman, A. *et al. Bayesian Data Analysis* 3rd edn (Chapman & Hall/CRC, 2013).
100. Lloyd, J. R. & Ghahramani, Z. Statistical model criticism using kernel two sample tests (2015).

Acknowledgements I acknowledge an EPSRC grant EP/I036575/1, the DARPA PPAML programme, a Google Focused Research Award for the Automatic Statistician and support from Microsoft Research. I am also grateful for valuable input from D. Duvenaud, H. Ge, M. W. Hoffman, J. R. Lloyd, A. Ścibior, M. Welling and D. Wolpert.

Author Information Reprints and permissions information is available at www.nature.com/reprints. The author declares competing financial interests: see go.nature.com/hyx7hh for details. Readers are welcome to comment on the online version of this paper at go.nature.com/hyx7hh. Correspondence should be addressed to Z.G. (zoubin@eng.cam.ac.uk).

0.2 Lecture 1

0.2.1 Notations

0.2.2 One Variable

0.2.2.1 Gaussian

0.2.2.2 Maximum Likelihood Estimation

0.2.2.3 Posterior and Maximum a Posteriori estimates (MAP)

0.2.3 More than one Variable

0.2.3.1 Multivariate Gaussian

0.2.3.2 Conditional and marginal distributions

0.2.4 Latent variables and Generative models

0.2.4.1 Example: Gaussian Factor Analysis

Probabilistic Machine Learning: Lecture 1

Isabel Valera

1 Notations

We try to be consistent with notations throughout the course. Here we give a summary of the main notation we will use. These are the same as in Bishop (2006).

- $\mathbf{x} = (x_1, \ldots, x_D)^T$: a column D-dimensional vector, i.e. lower case bold Roman letter.
- $\mathbf{x}^T$: the row vector transposed of $\mathbf{x}$.
- $\mathbf{x}_1, \ldots, \mathbf{x}_N$: N samples of a D-dimensional vector.
- $\mathbf{X}$: data matrix with n-th row the row vector $\mathbf{x}_n^T$; i.e. the n, i entry of $\mathbf{X}$ is the i-th feature/dimension of the n-th observation $\mathbf{x}_n$.
- $\boldsymbol{\mathsf{x}}$: data matrix for one-dimensional variables, i.e. this is a column vector whose n-th element is x_n.
- $\mathbf{x} \neq \boldsymbol{\mathsf{x}}$: we use two different typefaces. $\boldsymbol{\mathsf{x}}$ denotes a N-dim vector (i.e. N samples of one-dim variables); $\mathbf{x}$ denotes a D-dimensional vector (i.e. 1 sample of a D-dimensional variable).
- $\mathbb{E}_x\left[f(x, y)\right]$: expected value with respect to the random variable x of the function $f(x, y)$.
- $\mathbb{E}\left[x\right]$: expected value of x when there is no ambiguity as to which variable is being averaged over.

2 One variable

Suppose we have one random variable X, one-dimensional.
Assume that it is distributed according to some probability distribution, for instance Gaussian:

$$X \sim \mathcal{N}(\mu, \sigma^2) \quad , \tag{1}$$

where μ and σ^2 are the parameters.

Now, we observe N samples of this variable, $x_1, \ldots, x_N$, and assume they are independent and identically distributed as a Gaussian.

Question: given this sample, how do we estimate μ and σ?

2.1 Maximum Likelihood Estimation

One possibility is to find the pair (μ, σ) that maximizes the likelihood of observing the data. In general, the likelihood of the data $\boldsymbol{\mathsf{x}}$ for a probability distribution of parameters θ is defined as $p(\boldsymbol{\mathsf{x}}|\theta)$. Often it is mathematically more convenient to consider the logarithm of this quantity, also called log-likelihood. This is also useful numerically, as there is less risk of underflow when taking a sum instead of a product of a very small numbers. We denote the log-likelihood as $\mathcal{L}(\mu, \sigma^2)$ and we omit from the notation the explicit dependence on x. We recall here that the (log-) likelihood is a function of the parameters θ, in this case, μ and σ^2, as the observed data $x_1, \ldots, x_N$ are given.

Given that the logarithm is monotonically increasing, the maximum of these two coincide. Then we have:

$$\mu_{MLE}, \sigma^2_{MLE} = \underset{\mu,\sigma^2}{\operatorname{argmax}} \left\{ \mathcal{L}(\mu, \sigma^2) \right\} \tag{2}$$

We have N *independent* observations, denoted as a vector $\mathbf{x}$, so we can sum the log-likelihood of each of them:

$$\begin{aligned} \mathcal{L}(\mu, \sigma^2) = p(\mathbf{x}|\mu, \sigma^2) &= \sum_i^N \log\left(\mathcal{N}(x_i|\mu, \sigma^2)\right) &(3)\\ &= \sum_i^N \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{1}{2}\frac{(x_i-\mu)^2}{\sigma^2}\right)\right) &(4)\\ &= -\frac{1}{2}\sum_i^N \log(\sigma^2) - \frac{1}{2\sigma^2}\sum_i^N (x_i-\mu)^2 + const &(5) \end{aligned}$$

where the term *const* does not depend on the parameters. Now we want to extract the maximum of that expressions:

$$\begin{cases} \frac{\partial \mathcal{L}(\mu,\sigma^2)}{\partial \mu} \equiv 0 \\ \frac{\partial \mathcal{L}(\mu,\sigma^2)}{\partial \sigma^2} \equiv 0 \end{cases} \tag{6}$$

After some algebra, we can find the analytical expressions of the Maximum Likelihood estimation:

$$\begin{aligned} \mu_{MLE} &= \frac{1}{N}\sum_{n=1}^N x_n &(7)\\ \sigma^2_{MLE} &= \frac{1}{N}\sum_{n=1}^N \left(x_n - \mu_{ML}\right)^2 &(8) \end{aligned}$$

These correspond to the sample *empirical* mean and variance respectively.

Obs1: this was easy, because we could split the terms inside the sum as the samples were *independent* (and identically) distributed.
Obs2: from this estimation, we do not have idea of any measure on the uncertainty in estimating the parameters, i.e., μ_{MLE} and σ^2_{MLE} are point-estimates but no extra information was obtained from the calculations. In other words, we do not know how the ML estimate of the parameters compares with the true parameters, and thus how well the estimated parameters fit the overall true data distribution.

2.2 Posterior and Maximum a Posteriori estimates (MAP)

To address Obs2, we next treat μ, σ^2 also as random variables, each distributed according to some distribution. For instance, assume that:

$$p(\mu) = \mathcal{N}(\mu|, \mu_0, \sigma_0^2) \quad , \tag{9}$$

and keep σ^2 fixed. $p(\mu)$ is called the *prior* of the parameter μ. This distribution should be chosen in order to use some information (if any) that we know about the parameters. For instance, based on domain knowledge or by looking at the observed data; example: if the samples are all positive, perhaps it is more appropriate to set the prior as Gamma instead of Gaussian.
With this notion in mind and using Bayes' rule, we can now define the *Posterior* of μ as:

$$p(\mu|\mathbf{x}) = \frac{p(\mathbf{x}|\mu)\, p(\mu)}{p(\mathbf{x})} \propto p(\mathbf{x}|\mu)\, p(\mu) \quad , \tag{10}$$

where we made explicit the proportionality of the left-most term because in general the marginal likelihood $p(\mathbf{x})$, a.k.a. evidence, is not accessible (it involves complex calculations of integrals).

Hence, in practice, one estimates $p(\mu|\mathbf{x})$ by writing $p(\mathbf{x}|\mu)\,p(\mu)$ and attempting to then normalize the resulting expression. This is not generally possible analytically, but in this Gaussian example everything is doable. The result, after some algebra, is that the posterior is also Gaussian distributed, $p(\mu|\mathbf{x}) = \mathcal{N}(\mu|\mu_N, \sigma_N^2)$, with parameters:

$$\mu_N = \frac{\sigma^2}{N\,\sigma_0^2 + \sigma^2}\,\mu_0 + \frac{N\sigma_0^2}{N\sigma_0^2 + \sigma^2}\,\mu_{MLE} \tag{11}$$

$$\frac{1}{\sigma_N^2} = \frac{1}{\sigma_0^2} + \frac{N}{\sigma^2} \tag{12}$$

Obs1: μ_N is a combination of the prior's mean μ_0 and the MLE's estimate μ_{MLE}. In particular, when $N = 0$ we recover the prior, i.e. $\mu_N = \mu_0$; on the opposite scenario $N \to \infty$, we get $\mu_N \to \mu_{MLE}$, i.e. the MLE estimate and the mean of the Posterior coincide.
Obs2: $\frac{1}{\sigma_N^2}$, also called *precision*, is additive; which means that for $N \to \infty$ the σ_N^2 decreases, i.e. the more data we observe, the more the posterior variance decreases and the posterior distribution becomes peaked around μ_{MLE}.
Obs3: for *finite* N, if we have $\sigma_0^2 \to \infty$ (prior with infinte variance, also called *non informative* prior), then also in this case $\mu_N \to \mu_{MLE}$, but this time with $\sigma_N^2 = \sigma^2/N$.

These analysis and results can be observed by playing around with the *jupyter* notebook included in Lecture 1 material section.[1]

Being able to characterize the posterior distribution allows not only to measure the variance of the estimated parameters (i.e. σ_N^2), but one can perform other tasks such as sampling from that distribution. However, if one is interested on a point-estimate from the posterior, one common choice is taking the value at the peak of this distribution (also called *mode*), i.e., the maximum a posteriori (MAP) estimate:

$$\mu_{MAP} = \underset{\mu}{\operatorname{argmax}}\, p(\mu|\mathbf{x}) \quad . \tag{13}$$

For the example considered above where the posterior is also Gaussian, we have $\mu_{MAP} \equiv \mu_N$. But this is not the case in general.

Obs1: given that the (usually hard-to-compute) term $p(\mathbf{x})$ in the denominator of the exact posterior does not depend on μ, to calculate μ_{MAP} one can simply consider the maximization over $p(\mathbf{x}|\mu)\,p(\mu)$. This quantity is easier to access because does not require the hard task of normalizing the posterior.

Obs2: All the results of this section can be generalized for D-dimensional variables.

3 More than one variable

Suppose now that we have more than one random variable. For simplicity, we consider the case of having two of them x_1 and x_2 and both of them one-dimensional.

Also in this case, assume that these are distributed according to some probability distribution, for instance Gaussian:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \sigma_1^2 & \sigma_{12}^2 \\ \sigma_{12}^2 & \sigma_2^2 \end{bmatrix}\right) \quad , \tag{14}$$

this is also called *Multivariate Gaussian* with parameters $\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}$ and $\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12}^2 \\ \sigma_{12}^2 & \sigma_2^2 \end{bmatrix}$.
Σ is called *covariance* matrix and to be valid it has to be *positive semi-definite* (all its eigenvalues should be nonnegative) and *symmetric*. In general, x_1 and x_2 are not *independent*, the way they depend to each other is regulated by the cross terms of the covariance matrix σ_{12}^2.

[1] https://cms.sic.saarland/pml/materials/index

3.1 Conditional and marginal distributions

When two variables are correlated with each other, a relevant question is how does one vary when the other takes a given fixed value. This answer is provided by the *conditional* distribution of x_1 *given* x_2:

$$p(x_1|x_2) = \frac{p(x_1, x_2)}{p(x_2)} \tag{15}$$

The term in the denominator is the marginal probability of x_2 and is defined as:

$$p(x_2) = \int p(x_1, x_2)\, dx_1 \quad . \tag{16}$$

Given that this integral is not always easy to calculate, one can instead derive an expression for the conditional distribution rewriting as:

$$p(x_1|x_2) = \frac{p(x_1, x_2)}{Z_1} \tag{17}$$

where Z_1 is a normalization constant that one should find to make $p(x_1|x_2)$ a valid probability distribution. Usually finding Z_1 is easier than calculating the integral in (16). This can be done rewriting the Multivariate distribution isolating the terms depending on x_1:

$$(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu}) \quad = \quad \frac{1}{\sigma_1^2}\left[x_1 - \left(\mu_1 + (x_2-\mu_2)\frac{\sigma_1^2}{\sigma_{12}^2}\right)\right]^2 \tag{18}$$

This means that $p(x_1|x_2)$ is also a Gaussian, i.e. $\mathcal{N}(\mu_{x_1|x_2}, \sigma^2_{x_1|x_2})$, with parameters:

$$\mu_{x_1|x_2} \quad = \quad \mu_1 + (x_2-\mu_2)\frac{\sigma_1^2}{\sigma_{12}^2} \tag{19}$$

$$\sigma^2_{x_1|x_2} \quad = \quad \sigma_1^2 \tag{20}$$

These calculations can be done explicitly, this is a good practice exercise for handling Gaussian distribution (e.g. completing the square). Similar thinking can be applied to derive the expression of the marginal $p(x_1)$, but in this case calculations are much more tedious! Fortunately, in the multivariate Gaussian case, to obtain the marginal distribution over a subset of multivariate normal random variables, one only needs to drop the irrelevant variables (the variables that one wants to marginalize out) from the mean vector and the covariance matrix. In our example, the result is that also $p(x_1)$ is a Gaussian distribution, namely $\mathcal{N}(x_1|\mu_1, \sigma_1^2)$.

You can find interactive plots visualizing these distributions inside the Lecture 1 material section.[2]

Obs1: the number of parameters for the 2-variable example is 2 for the mean plus 4 for the covariance matrix. More generally, it grows as $N + N(N+1)/2$, i.e. quadratically in the number of variables N.

Obs2: the Multivariate Gaussian has a single maximum (it is *unimodal*), therefore it may be too limited to fit data with many modes.

Obs3: if x_1 and x_2 come from different types of distributions, e.g. one is Gaussian and the other is Gamma, and they are correlated, there is no such thing as a Multivariate distribution with mixed types of variables. In these cases, how do we capture correlations? As an example, think of the variables (*salary*, *balance*). These are two correlated variables, if you increase your salary, then it is more likely that also your bank account's balance increases. However, one is always positive (salary), the other can go negative (balance).

These three observations show that a Multivariate Gaussian suffers to major problems: it might not capture complex situations like non-unimodal or mixed-type distributions. And it has too many parameters, growing quadratically with the number of variables.

[2] https://cms.sic.saarland/pml/materials/index

4 Latent variables and Generative models

Question: how do we capture correlations in such complex scenarios?

The answer is given by *latent* variables. The purpose of these is to capture correlations of a complicated distributions via simpler *conditional* distributions. This is formalized by a generalization of the key idea behind De Finetti's theorem, which states that the probability distribution of any infinite exchangeable sequence of Bernoulli random variables is a "mixture" of the probability distributions of independent and identically distributed sequences of Bernoulli random variables.

Consider two D-dimensional variables $\mathbf{x}_1, \mathbf{x}_2$ and a K-dimensional variable $\mathbf{z}$ (our *latent* variable). The we have:

$$p(\mathbf{x}_1, \mathbf{x}_2) = \int p(\mathbf{x}_1|\mathbf{z})\, p(\mathbf{x}_2|\mathbf{z})\, p(\mathbf{z})\, d\mathbf{z} \tag{21}$$

The quantity in the right-hand-side is a joint distribution, in general this can be complicated as generally $p(\mathbf{x}_1, \mathbf{x}_2) \neq p(\mathbf{x}_1)\, p(\mathbf{x}_2)$. The terms inside the integral are instead *factorized*, thanks to the introduction of the auxiliary latent variable $\mathbf{z}$. This is the key point of the theorem.

We can interpret $p(\mathbf{x}_1, \mathbf{x}_2, \mathbf{z})$ as a model expressing the process that generated the observed data, i.e. we call this a *generative model*.

Obs1: even though we can factorize $p(\mathbf{x}_1, \mathbf{x}_2, \mathbf{z}) = p(\mathbf{x}_1|\mathbf{z})\, p(\mathbf{x}_2|\mathbf{z})\, p(\mathbf{z})$, we can still capture complex correlations between $\mathbf{x}_1, \mathbf{x}_2$ because when we marginalize out over $\mathbf{z}$ as in (21), we get a non-factorized joint $p(\mathbf{x}_1, \mathbf{x}_2)$.

4.1 Example: Gaussian Factor Analysis

An example of this is found in the following model (see Chapter 12 of Murphy (2012) for further details), where for simplicity we consider two one-dimensional variables:

$$\begin{aligned} p(x_1|z) &= \mathcal{N}(x_1|w_1 z, \sigma_1^2) & (22)\\ p(x_2|z) &= \mathcal{N}(x_2|w_2 z, \sigma_2^2) & (23)\\ p(z) &= \mathcal{N}(z|\mu_0, \sigma_0^2) & (24) \end{aligned}$$

Let's calculate the joint $p(x_1, x_2) = \int \mathcal{N}(x_1|w_1 z, \sigma_1^2)\, \mathcal{N}(x_2|w_2 z, \sigma_2^2)\, \mathcal{N}(z|\mu_0, \sigma_0^2)\, dz$. Without loss of generality, we can set and $\sigma_0^2 = 1$. Rewriting everything explicitly, we obtain that the marginal distribution of $(x_1, x_2)^T$ is a Multivariate Gaussian distribution with mean and covariance:

$$\begin{aligned} \boldsymbol{\mu} &= (w_1\mu_0, w_2\mu_0)^T & (25)\\ \boldsymbol{\Sigma} &= \begin{bmatrix} w_1^2 & w_1 w_2 \\ w_1 w_2 & w_2^2 \end{bmatrix} + \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix} \quad . & (26) \end{aligned}$$

The expression for $\boldsymbol{\Sigma}$ highlights that we now have correlations between the two variables, thanks to the terms $w_1 w_2$ in the non-diagonal entries.

Obs1: even though x_1 and x_2 were *conditionally* independent given z, their joint distribution is not factorized anymore, they have indeed a non-diagonal covariance matrix Σ.

Obs2: In general models, the integral in $p(x_1, \ldots, x_D) = \int \prod_{d=1}^{D} p(x_d|\mathbf{z})\, p(\mathbf{z})\, d\mathbf{z}$ does not have closed-form solution.

References

C. M. Bishop, *Pattern recognition and machine learning* (Springer, 2006).

K. P. Murphy, *Machine learning: a probabilistic perspective* (MIT press, 2012).