Bayesian Statistics and Machine Learning

(for scientists and applied mathematicians)

Alvin J. K. Chua

National University of Singapore

AY2022–23 Semester 2

Module Overview

- Bayesian statistics and machine learning: Two closely related subjects, nearly 50-50 split
- Familiarity with these is important for modern researchers in natural and social sciences
- PC5252/MA5221 is more mathematical with some hands-on content, not vice versa
  - PC5251 in AY2022–23 Special Term is the converse counterpart (highly recommended!)
- Why? Theoretical foundations ARE important for practical applications
- You will hopefully gain:
  - Fundamentals of the theory and practice of Bayesian statistics
  - Conceptual understanding of important machine-learning techniques
  - Essential practical experience with Bayesian statistics and machine learning
- You will not gain:
  - Broad exposure to cutting-edge techniques in deep learning and artificial intelligence
  - Detailed practical experience with popular machine-learning software platforms and libraries
  - Knowledge of best practices in software development
- Feedback is welcome at any time: I will make changes if required (where practical)
- This is a brand-new course, so do bear with me as I iron out the kinks!

Module Logistics

- Lectures
  - S16-0430 every Mon and Thu, 1200–1400
  - Actually start at 1205 and end at 1330+, 5–10m break in between
  - Chinese New Year: No lecture on 23 Jan (Mon of Week 3)
  - Final lecture will be recap of course and preview of final exam
- Tutorials
  - Start in Week 3, will replace second half of Thu lectures
  - Discuss problem sets (both problems and solutions), mid-term solutions
- Office hours
  - Every Mon and Thu, 1500–1700
  - I am still in a temporary office for now: S13-M-05
  - Happy to chat before and after lectures/tutorials as well
  - E-mails are fine, but there might be a time lag in replies anyway
- Assessment
  - Attendance and participation (15%): Virtually free, unless yours is so bad as to be noticeable
  - 5 problem sets (30%): Release at end of Week n, due at start of Week $n + 2$
  - Mid-term exam (25%): 1.5h on 27 Feb (Mon of Week 7), format TBD
  - Final exam (30%): 2h on 2 May

Module Logistics

- Materials and supplementary reading
  - Course notes = these slides
  - This course is based largely on various treatments in *Bayesian Data Analysis* (Gelman et al.), *Information Theory, Inference and Learning Algorithms* (MacKay), *Gaussian Processes for Machine Learning* (Rasmussen & Williams), and *Deep Learning* (Goodfellow et al.)
  - Other useful references: *Pattern Recognition and Machine Learning* (Bishop), *Machine Learning: A Probabilistic Perspective* (Murphy), *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (Hastie et al.)
- Tools
  - Course administration: Canvas
  - Computer algebra system: Mathematica
  - Simple scientific computing: Python, Jupyter notebooks

Module Contents

Module Contents

Module Contents

Module Contents

1. Preliminaries

Preliminaries

Here we list basic definitions, assumptions, conventions and notational choices that may be assumed as default throughout these notes (unless explicitly stated otherwise). This list is mainly for convenient reference, and is not intended to be exhaustive.

- Population: A possibly infinite set of objects with observable properties that can be described by statistical models.
- Sample: A finite subset of the population. However, the term is used more often to describe a set of observations (the *data*) corresponding to such a subset. We typically denote such observations by Roman letters such as x, y.
- Sample (redux): In the context of Monte Carlo simulations, the term can also refer to *individual* realisations of a random variable. This is slightly incongruous with the above definition, since the data is often treated as a set of such realisations.
- Estimand: Unobserved quantities that are the subject of inference. These can either be future/alternative observations (*observables*, which we denote by overtildes, e.g., $\tilde{x}$), or underlying quantities that describe the population (*parameters*, which we typically denote by Greek letters such as θ, ϕ).
- Estimator: A function of the data (a *statistic*) that is used to infer the value of a parameter. We denote this by overhats, e.g., $\hat{\theta}(x)$.

- Probability distribution and mass/density: We use the terms distribution and mass/density interchangeably, and so we overload the lower-case letter p to denote both. In other words, we might say the probability distribution of θ is p (shorthand $\theta \sim p$), or that θ is distributed according to the probability mass/density function $p(\theta)$.
- Probability values: We use the upper-case letter P to denote probability values, i.e., numbers between zero and one. Thus $P(\theta \in \Theta) = \int_\Theta d\theta\, p(\theta)$, for continuous variables. We use this notation for discrete variables as well, where it is taken to mean $\sum_{\theta \in \Theta} p(\theta)$.
- Common distributions: The normal and multinormal distributions are denoted by $\mathcal{N}(\mu, \sigma^2)$ and $\mathcal{N}(\mu, \Sigma)$ respectively. The uniform distribution is denoted by $\mathcal{U}(a, b)$. The χ^2 and t distributions with ν d.o.f. are denoted by χ^2_ν and t_ν respectively. Other standard distributions are denoted more explicitly, e.g., the binomial distribution is $\mathrm{Bin}(n, p)$.
- Exchangeable and iid observations: We assume that any indexed set of observations $\{x_i\}$ is exchangeable, which means the joint distribution $p(\{x_i\})$ is invariant to index permutation. This can always be made valid in principle by including enough explanatory variables (*covariates*) y such that the set $\{(x, y)_i\}$ is exchangeable. In fact, we often use the stronger assumption that the $\{x_i\}$ are independently and identically distributed (*iid*).

2. Recap of Probability Theory

Fundamentals

The *joint distribution* of two random variables x, y is denoted by $p(x, y)$.

Probabilistic conditioning is denoted by vertical bars, e.g., $x|y$ means "x conditioned on y", or simply "x given y". The same notation can be used to describe variables, their distributions, or any computed quantities/probabilities. Conditioning can be on other variables, some higher-level choice of model, or indeed any other explicit assumptions.

Conditional distributions are related to the joint distribution by

$$p(x, y) = p(x|y)p(y) = p(y|x)p(x). \tag{1}$$

The distribution $p(x)$ is referred to in this context as the *marginal distribution* of x, since it can be obtained by marginalising (summing/integrating the joint distribution) over y:

$$p(x) = \int dy\, p(x, y) = \int dy\, p(x|y)p(y). \tag{2}$$

Independence: x and y are independent if and only if

$$p(x, y) = p(x)p(y), \tag{3}$$

or equivalently, if conditionals $=$ marginals.

The *expectation* of a test function $\tau(x)$ under a given distribution $p(x)$ is defined as

$$\mathrm{E}[\tau] \equiv \langle \tau \rangle := \int dx\, \tau(x) p(x). \tag{4}$$

The expectation of x itself is called the mean of the distribution: $\mu := \mathrm{E}[x]$.

The n-th (central) *moment* of the distribution is defined as

$$\mu_n := \mathrm{E}[(x - \mu)^n]. \tag{5}$$

The second central moment is called the variance of the distribution: $\mathrm{Var}[x] \equiv \sigma^2 := \mu_2$.

Exercise

Show that $\mathrm{Var}[x] = \mathrm{E}[x^2] - \mathrm{E}[x]^2$.

For vector-valued x (and thus μ), the generalisation of $\mathrm{Var}[x]$ is the covariance matrix

$$\Sigma := \int dx\, [x - \mu][x - \mu]^T p(x). \tag{6}$$

The mean and variance of conditional distributions such as $p(x|y)$ are referred to as the conditional mean $\mathrm{E}[x|y]$ and conditional variance $\mathrm{Var}[x|y]$.

Here are some useful relations between the moments and their conditional counterparts:

Exercise

Show that for two random variables x, y, $\mathrm{E}[x] = \mathrm{E}[\mathrm{E}[x|y]]$ (where the inner expectation is a function of y, and so the outer expectation is over y).

Exercise

Show that for two random variables x, y, $\mathrm{Var}[x] = \mathrm{E}[\mathrm{Var}[x|y]] + \mathrm{Var}[\mathrm{E}[x|y]]$.

Consider a transformation of variables $x' = f(x)$, for some surjective f.

Let f also be injective. For discrete x, the transformed probability mass function (for x') is

$$p'(x') = p(f^{-1}(x')). \tag{7}$$

For continuous x, the transformed probability density function is

$$p'(x') = |\det J|\, p(f^{-1}(x')), \tag{8}$$

where J is the Jacobian of f^{-1}, i.e., $[J]_{ij} = \partial_i x / \partial_j x'$.

Similar expressions hold even if f is not injective: the RHS is replaced by a sum of terms corresponding to each branch of f^{-1}.

Example

For $x \in \mathbb{R}$ distributed according to $p(x)$, what is the density function of $x' = x^2$?

Here are some useful scalar-variable transformations:

$$(0,\infty) \to (-\infty,\infty) : x \mapsto \ln x, \tag{9}$$

$$(-\infty,\infty) \to (0,\infty) : x \mapsto e^x, \tag{10}$$

$$(0,1) \to (-\infty,\infty) : x \mapsto \mathrm{logit}(x) := \ln\left(\frac{x}{1-x}\right), \tag{11}$$

$$(-\infty,\infty) \to (0,1) : x \mapsto \mathrm{logit}^{-1}(x) := \frac{e^x}{1+e^x}, \tag{12}$$

$$(0,1) \to (-\infty,\infty) : x \mapsto \mathrm{probit}(x) := \Phi^{-1}(x), \tag{13}$$

$$(-\infty,\infty) \to (0,1) : x \mapsto \mathrm{probit}^{-1}(x) := \Phi(x). \tag{14}$$

The letter Φ above denotes the *cumulative distribution function* of the standard normal distribution $\mathcal{N}(0,1)$ (and Φ^{-1} is its *quantile function*). More explicitly, for $x' \sim \mathcal{N}(0,1)$:

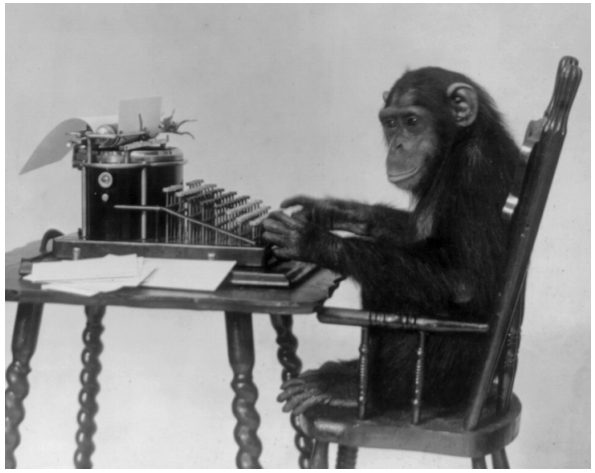
$$\Phi(x) := P(x' \leq x) = \int_{-\infty}^{x} dx'\, \mathcal{N}(x'|0,1). \tag{15}$$

This integral (and its inverse) is very commonplace, but does not have a closed-form solution.

Probability as Uncertainty

Probability can be interpreted as *frequency*, and used to describe it:

- Verifiable frequency: Probability of landing on heads in a fair coin toss
- Past frequency: Probability that a child born in Singapore is female
- Conceptual frequency: Probability that a monkey with a typewriter will reproduce Shakespeare's works within a Hubble time



Credit: New York Zoological Society

The frequency interpretation is often useful but sometimes restrictive. It starts to strain when we talk about more general notions of probability:

- Inverse probability: Probability that a coin landing on heads 140 out of 250 times is fair
- Many hierarchical d.o.f.: Probability that Singapore will win the World Cup in 2026
- Multiple realities needed: Probability that quantum gravity is solved in the next 10 years

Probability can also be interpreted as *uncertainty*, and used to describe it. This is appropriate for more general notions of probability (along with those that can be mapped to frequencies). With the uncertainty interpretation, probability can be used to describe assumptions, as well as inferences given those assumptions.

We can categorise uncertainty in two ways:

Aleatoric	Associated with randomness
Epistemic	Associated with lack of knowledge

Isn't there some overlap? Lack of knowledge due to assumed randomness in a d.o.f., or apparent randomness due to lack of knowledge about all the d.o.f.

Coin example:

- Denote the outcome of a given toss by $x \in \{0, 1\}$ (define heads as $x = 1$)
- Denote the true bias of the coin by $\theta \in [0, 1]$ (define a heads-only coin as $\theta = 1$)
- The aleatoric uncertainty is described by $p(x|\theta)$
- The epistemic uncertainty is described by $p(\theta)$
- So the probability of landing on heads in a fair coin toss is $P(x = 1|\theta = 0.5)$
- What about the probability that a coin landing on heads 140 out of 250 times is fair?

3. Introduction to Bayesian Statistics

Motivation and Background

Constructing *models* of expected data and drawing *inferences* from actual data are central tasks in the natural and social sciences.

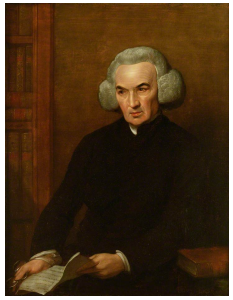
Bayesian statistics is a probability-based framework for quantifying uncertainty.

Bayesian methods thus allow the systematic and explicit quantification of uncertainty in both modelling and inference, which leads to more robust conclusions and predictions.

A brief history:

- Named after Thomas Bayes, who was the first to invert the probability statement
- Paper published posthumously in 1763 (with contributions from Richard Price)
- Example was a binomial model: given data x from a binomial distribution $\mathrm{Bin}(n, p)$ with unknown p, what is the probability that p lies in some interval?
- Independently derived and extended by Pierre-Simon Laplace in 1774
- Laplace is credited with founding the general Bayesian interpretation of probability


Bayes


Price


Laplace

A brief history, continued:

- This early Bayesian interpretation lasted around a century, although it was just called "probability theory" and less rigorously treated
- *Frequentist statistics* became dominant around the turn of the 1900s
- Based on frequency interpretation: Popular as it was rigorous and apparently objective
- Modern Bayesian statistics became more rigorously developed in the mid-1900s

Frequentist statistics has been so dominant in the past century that even now, it is often the "default" framework for statistical data analysis. It has some limitations though:

- Little to no quantification of uncertainty in modelling
- Quantification of uncertainty in inference is given by confidence intervals for estimators, which are frequently misinterpreted and not as descriptive as probability distributions
- Thus inverse probabilities are not (properly) addressed
- Construction of frequentist estimators can often be ad hoc

On the other hand, a common criticism of Bayesian statistics is that it relies heavily on assumptions and is thus "subjective".

Counterpoint: We cannot do inference without assumptions! Frequentist methods also have implicit assumptions, while Bayesian methods force us to make our assumptions explicit.

Fundamentals

Bayes' theorem is a basic property of conditional probability that follows from Eq. (1):

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)}. \tag{16}$$

The various pieces of this expression have natural interpretations in practical applications, as they map neatly to the main procedures in Bayesian modelling and inference.

Likelihood

Assume the data x is a sample from a statistical population that is described by some underlying parameters θ. This is stated by the data distribution $p(x|\theta)$, which is the "forward" probability model for x given θ. We can regard $p(x|\theta)$ as a function of θ given x, which is called the *likelihood* and often denoted $L(\theta|x)$. Crucially, L is not a probability mass/density function for θ since it is generally not normalised for any x: $\int d\theta\, L(\theta|x) \neq 1$.

Likelihood principle: Given some data, all inferences should depend on the data only through the choice of likelihood function (and not, for example, the data collection method).

Prior

We also need to explicitly specify our assumptions about θ through $p(\theta)$, which is called the *prior* distribution and sometimes denoted $\pi(\theta)$. This is central to the principle and practice of Bayesian inference, and π should describe our knowledge about θ before any observations.

Evidence

Note that the LHS of Bayes' theorem is a mass/density function for θ, and so $p(x)$ on the RHS must be the normalising factor for the function $L\pi$. This quantity is called the *evidence* for x, and is often denoted Z. In a more general setting where x is not fixed, $p(x)$ is the marginal distribution of x from Eq. (2), and is sometimes called the *prior predictive* distribution.

Posterior

On the LHS of Bayes' theorem, $p(\theta|x)$ is called the *posterior* distribution of θ. The primary task in Bayesian inference is to compute/estimate the posterior. This can be viewed as an inversion of the probability statement $p(x|\theta)$ (which, recall, is an assumed model). Specific inferences may be made once the posterior is in hand, e.g., the expectation of a test function $\tau(\theta)$ under the posterior: $\mathrm{E}[\tau|x] = \int d\theta\, \tau(\theta)p(\theta|x)$.

To recap, the essence of Bayesian modelling and inference can be summarised as

$$p(\theta|x) = \frac{L(\theta|x)\pi(\theta)}{Z} \propto L(\theta|x)\pi(\theta).$$

Odds

Odds are a useful way to describe relative probabilities in Bayesian statistics, as they often do not require the explicit evaluation of probability values (which can be nontrivial integrals). For a given Bayesian model, the ratio of the posterior mass/density $p(\theta|x)$ evaluated at parameter values θ_1 and θ_2 is called the *posterior odds* for θ_1 against θ_2. From Bayes' theorem, we have

$$\frac{p(\theta_1|x)}{p(\theta_2|x)} = \frac{L(\theta_1|x)}{L(\theta_2|x)} \frac{\pi(\theta_1)}{\pi(\theta_2)}, \tag{17}$$

i.e., the posterior odds are equal to the *likelihood ratio* times the prior odds.

Example

I have three dice: 6-sided, 8-sided and 12-sided. I pick one at random, and roll a 1. What are the posterior odds that the die is 6-sided?

Prediction

Inference can be extended to make predictions about observables, not just statements about parameters. For example, let $\tilde{x}$ be a future observation from the data distribution $p(x|\theta)$ (assuming that $x|\theta$ and $\tilde{x}|\theta$ are iid). The *posterior predictive* distribution of $\tilde{x}$ is

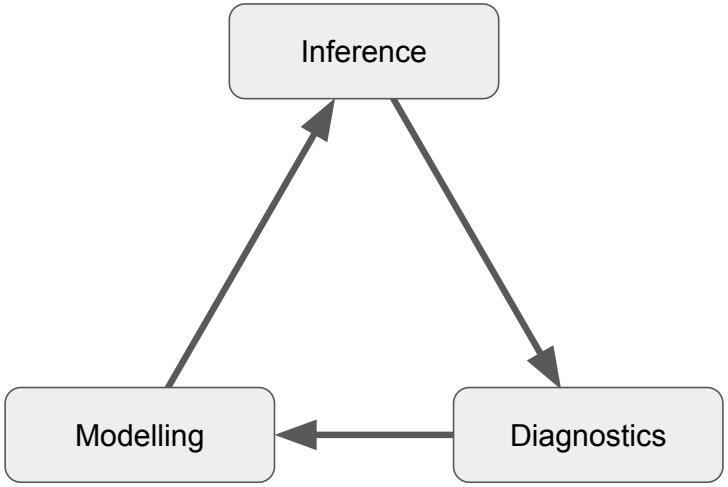
$$p(\tilde{x}|x) := \int d\theta\, p(\tilde{x}|x,\theta)p(\theta|x) = \int d\theta\, p(\tilde{x}|\theta)p(\theta|x), \tag{18}$$

which is defined by the first equation as the expectation of the conditional distribution $p(\tilde{x}|x,\theta)$ under the posterior.

Exercise

Show explicitly how the second equality in Eq. (18) relies on the iid assumption.

Applying Bayesian Statistics



Modelling

The first step is to construct probabilistic statements (likelihood and prior) that describe what we know, what we don't know, and what we assume about the problem.

- These should be informed by the underlying physics, expert knowledge, etc.
- These should account for how the data is observed and not just how it is generated, e.g., estimating Covid trends via testing vs wastewater tracking has different uncertainties
- The probabilistic parts of our models should generally be simple, because we want to be conservative about what we don't know and what we assume

All models are wrong, but some are useful. — George Box

Inference

The next step is to apply Bayes' theorem: conceptually simple, but computationally difficult!

However, we just said the likelihood and prior should generally be simple models, and the posterior is just their product if we are not concerned with the evidence. What's the problem?

Example

Consider the normal data distribution $\mathcal{N}(f(\theta), 1)$ of x for given θ, with $f : \mathbb{R} \to \mathbb{R}$. What is the corresponding likelihood of θ for given x?

The problem is inversion: a simple data distribution does not generally $=$ a simple likelihood.

Thus computing the evidence and other expectations requires numerical sums/integrals of mass/density functions that can be high-dimensional or multimodal. Stochastic *sampling* algorithms are typically needed to achieve this with a feasible number of evaluations.

Diagnostics

The next step is to determine whether our models are useful for describing/predicting the data, and whether our inferences are sensible. This involves the validation of models, the comparison of alternative models, and the interpretation of inferences.

It's not necessarily the final step: the uncertainty in our models (both likelihood and prior) can then be *updated*, which will improve future inferences and lead to better science.

Relationship with Machine Learning

First, what is machine learning?

- Using statistics, algorithms and computers to learn about a problem from data
- Reductionist view: Isn't this just fitting a model to data?

Similarities with Bayesian statistics:

- Main aim is to make estimates of parameters and predictions of observables
- Most methods explicitly quantify aleatoric uncertainty in the model
- Training (learning from repetition) is like updating uncertainty

Differences from Bayesian statistics:

- Some methods do not specify a forward probability model (likelihood), but learn it automatically from the data — although this is Bayesian in principle as well
- Big data reduces the reliance on assumptions about the problem (prior)
- Many applications do not explicitly seek the posterior distribution

In practice, there are many ways to do machine learning without Bayesian methods. Likewise, Bayesian statistics has many applications beyond machine learning.

Still, the two subjects are closely related: knowing one is halfway to knowing the other.

4. One-parameter Models

Binomial Distribution (with Known n)

The binomial distribution is used to model a sequence of n iid trials, each with an outcome in $\{0, 1\}$ (failure or success) and a probability of success p. Because iid $\implies$ exchangeability, the sequence of outcomes can be summarised by the number of successes x.

We focus here on known n and unknown $p \equiv \theta$, which is more common in applications.

- Simplest model to introduce Bayesian concepts and methods
- Also the example used in the first published Bayesian analysis

Bayes' example: Ball A is thrown onto a table, and lands in a position with horizontal coordinate $\theta \in [0, 1]$. Another ball B is thrown onto the table n times, and x is the number of times B lands to the right of A. What is $P(\theta \in \Theta | x)$ for some subinterval $\Theta \subset [0, 1]$?

Other examples:

- Coin toss
- Sex of a newborn
- Pretty much any condition and its negation (as long as iid assumption is sound)

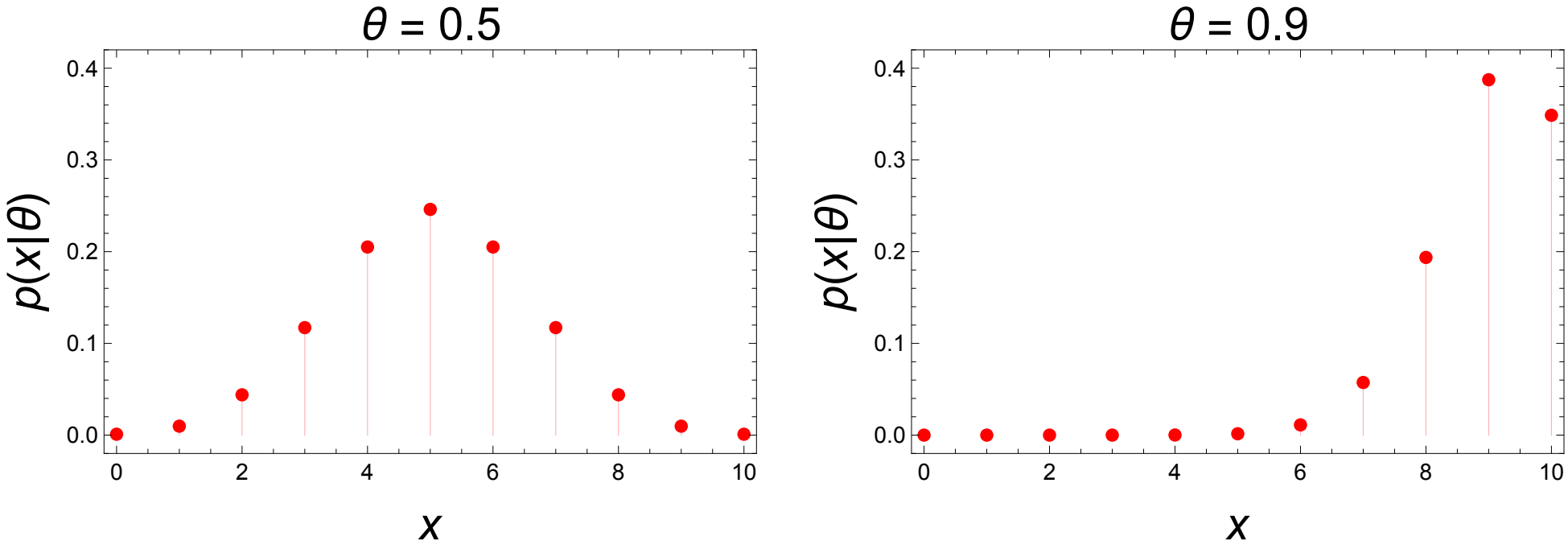
Related models:

- When $n = 1$: Bernoulli distribution
- Not iid but exchangeable: Logistic regression
- Not exchangeable: Ising model

The probability of observing exactly x successes is simply the product of probabilities for x successful trials and $n - x$ unsuccessful trials, times the number of combinations (trials are exchangeable). Thus the probability mass function is

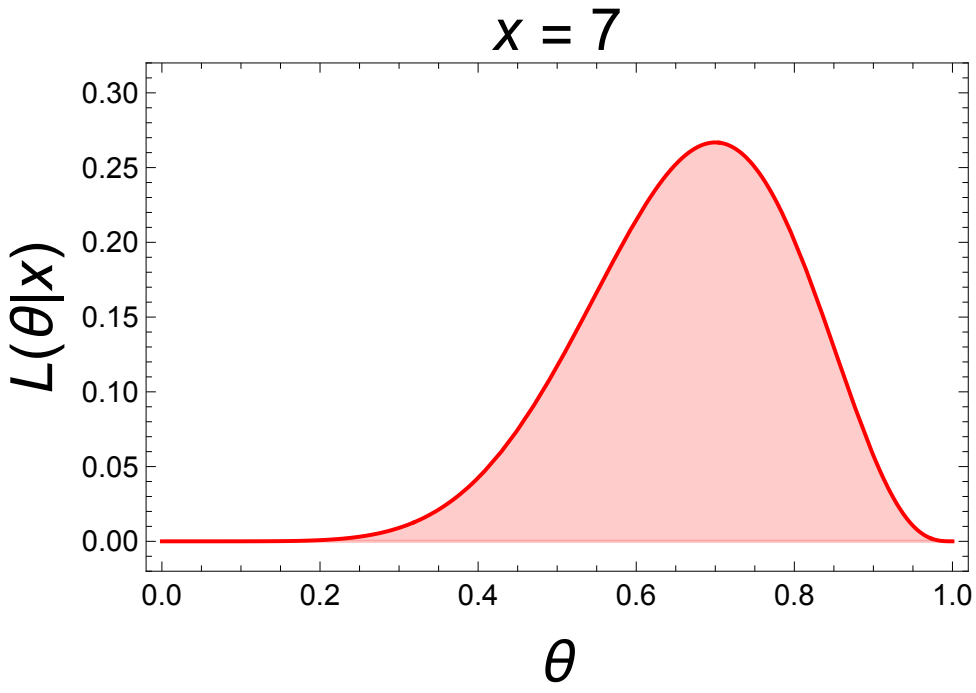
$$\mathrm{Bin}(x|n,\theta) = \binom{n}{x}\theta^x(1-\theta)^{n-x}. \tag{19}$$

Now consider the data distribution $p(x|\theta) = \mathrm{Bin}(x|n,\theta)$ for $n = 10$, and various fixed θ:



The function $p(x|\theta)$ is discrete (in its argument $x \in \{0, \ldots, 10\}$), and is already normalised. It is the forward probability model, and describes the aleatoric uncertainty in the problem.

Then consider the corresponding likelihood $L(\theta|x)$ for some given x, say $x = 7$:



The function $L(\theta|x)$ is now continuous (in $\theta \in [0, 1]$), and is not normalised (on $[0, 1]$). It is an inversion of the forward model, and describes epistemic uncertainty.

What is the Bayesian interpretation of the normalisation constant $\int_{[0,1]} d\theta\, L(\theta|x)$?

It is the evidence Z for given x (or the marginal distribution $Z(x)$ for arbitrary x), if we assume a uniform prior for θ: $\pi(\theta) = \mathcal{U}(\theta|0,1) = \mathbf{1}_{[0,1]}(\theta)$. The explicit solution for Z contains an integral known as the beta function:

$$B(a,b) := \int_0^1 dt\, t^{a-1}(1-t)^{b-1}, \tag{20}$$

where $a, b \in (0, \infty)$. This is commonly written in terms of the gamma function:

$$\Gamma(a) := \int_0^\infty dt\, t^{a-1}e^{-t}, \quad B(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}. \tag{21}$$

For $a \in \mathbb{Z}^+$, the gamma function reduces to $\Gamma(a) = (a-1)!$.

Exercise

Show that the marginal distribution $Z(x) = \binom{n}{x} B(x+1, n-x+1)$ is uniform on $\{0, \ldots, n\}$.

Exercise

Write down the posterior density function $p(\theta|x)$ in terms of Γ.

Thus the posterior distribution is a (particular) *beta distribution*:

$$\theta|x \sim \text{Beta}(x+1, n-x+1), \quad \text{Beta}(t|a,b) = \frac{1}{B(a,b)} t^{a-1}(1-t)^{b-1}. \tag{22}$$

Although its PDF is in closed form, its CDF is not (requires incomplete beta functions).

Laplace's example: Between 1745 and 1770, x out of n children born in Paris were female. If θ is the probability that any such child is female, what is $P(\theta > 0.5|x)$?

A beta distribution is well approximated by a normal distribution for large a, b. With $x = 241945$ and $n = 493472$, Laplace used this approximation (and tables of known values for the normal CDF) to show that $P(\theta > 0.5|x) \sim 10^{-42}$.

The binomial model is also instructive in examining predictive distributions for observables.

We make a small reparametrisation here: $x \in \{0, \ldots, n\}$ is the observed number of successes out of n trials, but $\tilde{x} \in \{0, 1\}$ is the outcome of another trial from the same population.

- Known/assumed θ: Data distribution (describes aleatoric uncertainty through likelihood)

$$p(\tilde{x}|\theta) = \begin{cases} 1-\theta & \tilde{x} = 0 \\ \theta & \tilde{x} = 1 \end{cases} \tag{23}$$

- Unknown θ with uniform prior but no data: Prior predictive distribution (adds epistemic uncertainty through the prior)

$$p(\tilde{x}) = \int_0^1 d\theta\, p(\tilde{x}|\theta)p(\theta) = \begin{cases} 1/2 & \tilde{x} = 0 \\ 1/2 & \tilde{x} = 1 \end{cases} \tag{24}$$

- Unknown θ with uniform prior and observed data: Posterior predictive distribution (updates epistemic uncertainty through the posterior)

$$p(\tilde{x}|x) = \int_0^1 d\theta\, p(\tilde{x}|\theta)p(\theta|x) = \begin{cases} (n-x+1)/(n+2) & \tilde{x} = 0 \\ (x+1)/(n+2) & \tilde{x} = 1 \end{cases} \tag{25}$$

We focus now on $P(\tilde{x} = 1|x)$ (the second case of Eq. (25)), which is simply the posterior mean of $P(\tilde{x} = 1|\theta) = \theta$. This result is known as Laplace's rule of succession.

Example

What is the probability that the sun will rise tomorrow?

That's actually pretty low! It's roughly the probability of a pregnant woman not giving birth to septuplets (or more). Is the Bayesian approach nonsensical?

Laplace was ridiculed for a similar calculation (using $\sim 10^6$ days as the age of the Universe), because his critics failed to appreciate the caveat he provided: "But this number is far greater for him who, *seeing in the totality of phenomena the principle regulating the days and seasons*, realises that nothing at present moment can arrest the course of it."

Bayesian inference is only as meaningful as your assumptions (or lack thereof).

On the Choice of Prior

We have been using the uniform prior $\mathcal{U}(0, 1)$ in our binomial model thus far. Why?

- For convenience: It greatly simplifies certain analytical calculations
- For intuition: It matches expectations in certain problems, e.g., Bayes' example
- For moderation: It imposes few or weak assumptions on θ

However, as the sunrise example shows: a uniform prior does not generally equate to sensible assumptions, or the absence of assumptions.

The uniform prior for the binomial model is an example of a proper, uninformative prior.

Proper vs improper priors

A proper probability mass/density function is one that sums/integrates to unity on its domain, while an improper one does not.

- The propriety of a prior depends on its domain as well as its functional form. For example, the constant prior density $p(\theta) = \mathbf{1}_{(a,b)}(\theta)$ can be made proper for any bounded interval by renormalising it to $\mathcal{U}(\theta|a, b)$. However, this is not possible on the real line.
- Improper priors can still be used for convenience, provided that the resultant posterior is proper (and sensible). For example, the likelihood $\mathcal{N}(\theta|x, 1)$ with a constant prior on $\mathbb{R}$ has a finite evidence, and thus a proper posterior.

Uninformative vs informative priors

An uninformative (aka vague, flat, or diffuse) prior is one that imposes weak constraints on θ, such that it has a minimal effect on the posterior and allows the data to "speak for itself".

- Might lead to nonsensical results
- Is not an absence of assumptions — the weakness of constraints is itself an assumption
- Examples: Uniform, log-uniform, Jeffreys

An informative prior is one that imposes strong constraints on θ, based on existing knowledge about the epistemic uncertainty in the problem.

- Might be informed by previous inferences that are constructed from representative data, e.g., a Bayesian posterior or a frequentist estimate (with confidence interval)
- Might be informed by educated guesses, e.g., present rate of global sea-level rise is between zero and that of filling a swimming pool with a paper cup every hour*
- Might not always be available
- Might introduce bias

A weakly informative prior can be used as a practical middle ground.

- Can be defined by adding information to an uninformative prior
- Can be defined by adding uncertainty to an informative prior

*It's actually slightly faster, unfortunately

Fisher information and Jeffreys prior

Harold Jeffreys suggested a general principle for constructing an uninformative prior given some data distribution, such that the prior is invariant under any transformation of variables.

This construction uses the Fisher information (named after Ronald Fisher), which is a measure of how informative x is about θ (equivalently, how sensitive x is to θ):

$$[\mathcal{I}(\theta)]_{ij} := \mathrm{E}\left[\left(\frac{\partial}{\partial\theta_i}\ln p(x|\theta)\right)\left(\frac{\partial}{\partial\theta_j}\ln p(x|\theta)\right)\middle|\theta\right] = -\mathrm{E}\left[\frac{\partial^2}{\partial\theta_i\partial\theta_j}\ln p(x|\theta)\middle|\theta\right]. \tag{26}$$

The Jeffreys prior is then

$$p(\theta) :\propto \sqrt{\det\mathcal{I}(\theta)}. \tag{27}$$

Exercise

Show for $\theta', \theta \in \mathbb{R}$ that the Jeffreys prior is invariant under transformation, i.e., $p'(\theta') = p(\theta')$.

Example

What is the Jeffreys prior for the binomial model?

The Jeffreys prior is not guaranteed to be proper (although this particular one is).

Jeffreys' principle of constructing transformation-invariant priors is sensible in contexts such as scale parameters, but should not be universally applied in practice. For example, such priors can lead to less robust results when used in high-dimensional problems with many parameters.

Conjugate prior

Another principle for prior construction is that of conjugate priors. For certain data distributions, it is possible to choose a functional form for the prior that gives a posterior with the same form. This aids analytical computation, and can be useful for understanding simple models, or for constructing more complex models.

We will examine the conjugate prior for the binomial data distribution, where the likelihood is

$$L(\theta|x) \propto \theta^x (1-\theta)^{n-x}. \tag{28}$$

The functional form of the likelihood resembles the density function for the beta distribution in Eq. (22). So let us choose a beta prior with arbitrary a, b:

$$\pi(\theta) = \mathrm{Beta}(\theta|a,b) \propto \theta^{a-1}(1-\theta)^{b-1}. \tag{29}$$

This is a conjugate prior, as the posterior is

$$p(\theta|x) \propto \theta^{x+a-1}(1-\theta)^{n-x+b-1} \propto \mathrm{Beta}(\theta|x+a, n-x+b). \tag{30}$$

It is straightforward to see that both the uniform prior $\mathrm{Beta}(1,1)$ and the Jeffreys prior $\mathrm{Beta}(1/2,1/2)$ are special cases of the beta conjugate prior.

Strength of data vs prior

The beta conjugate prior $\mathrm{Beta}(a, b)$ for the binomial data distribution has a useful interpretation as the prior observation of $a - 1$ successes and $b - 1$ failures.

From the properties of the beta distribution, the posterior mean is

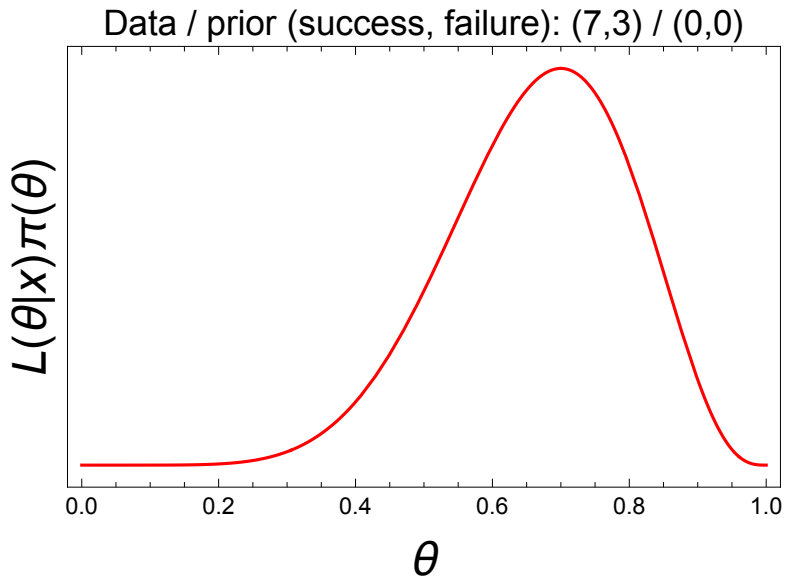
$$\mathrm{E}[\theta|x] = \frac{x + a}{n + a + b}. \tag{31}$$

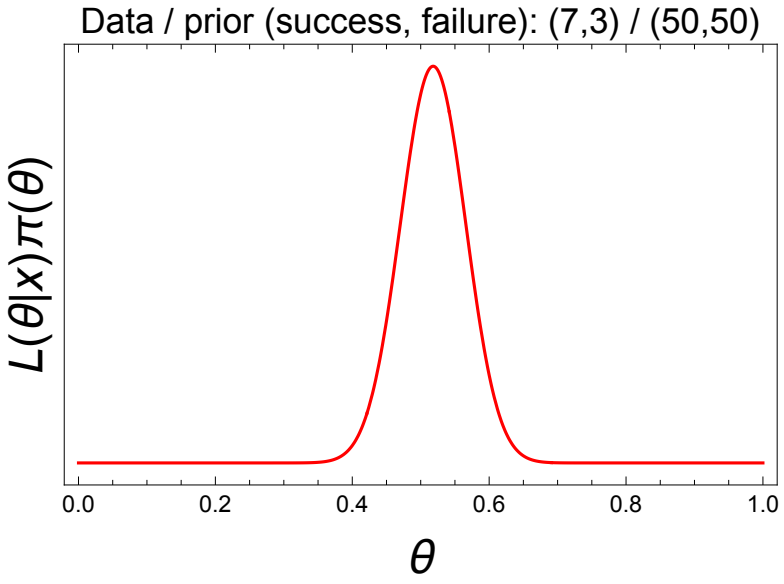
This tends to the data mean x/n when there is more data than prior evidence ($n, x \gg a, b$), and to the prior mean $a/(a + b)$ when the converse is true ($a, b \gg n, x$).

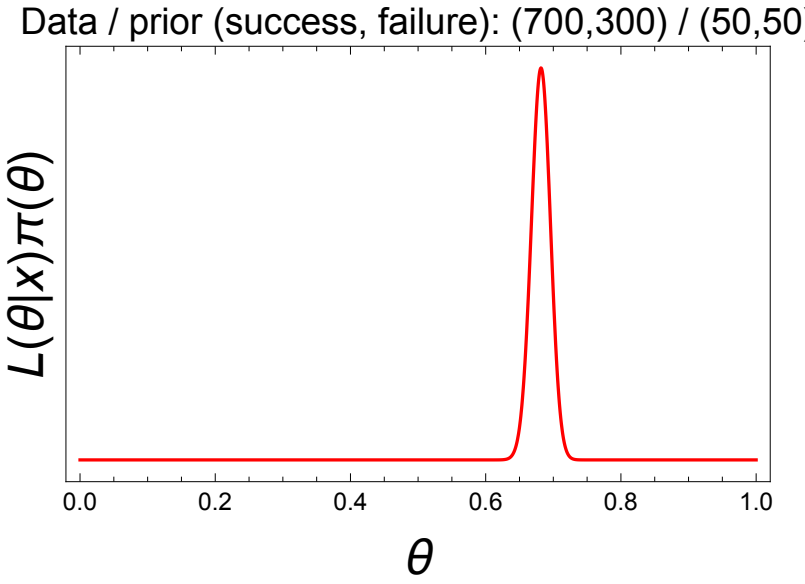
The posterior variance is

$$\mathrm{Var}[\theta|x] = \frac{(x + a)(n - x + b)}{(n + a + b)^2(n + a + b + 1)} = \frac{\mathrm{E}[\theta|x](1 - \mathrm{E}[\theta|x])}{n + a + b + 1}. \tag{32}$$

This tends to zero for a large data set ($n \to \infty$), or for a strong prior ($a + b \to \infty$).


Data / prior (success, failure): (7,3) / (0,0)
$L(\theta|x)\pi(\theta)$
0.0
0.2
0.4
0.6
0.8
1.0
θ


Data / prior (success, failure): (7,3) / (50,50)
$L(\theta|x)\pi(\theta)$
0.0
0.2
0.4
0.6
0.8
1.0
θ


Data / prior (success, failure): (700,300) / (50,50)
$L(\theta|x)\pi(\theta)$
0.0
0.2
0.4
0.6
0.8
1.0
θ

Normal Distribution (with Known σ^2)

The normal (aka Gaussian) distribution is ubiquitous in statistical modelling, both as a simple model and as a part of more complex models. There are two main reasons for this:

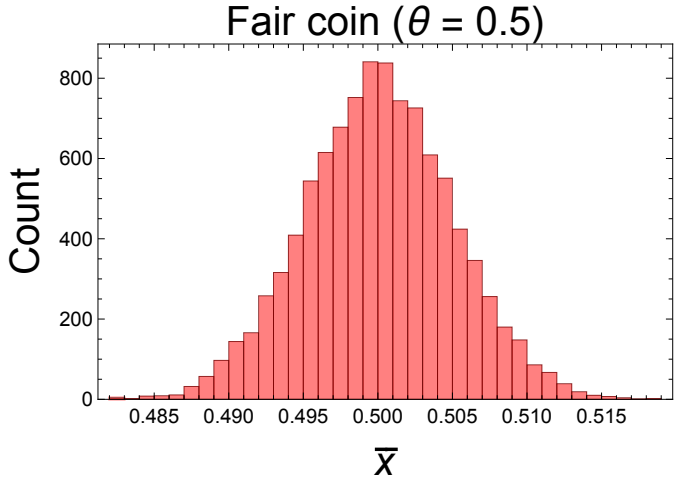
- It is analytically tractable
- Its validity as an approximation in most situations is justified by the *central limit theorem*

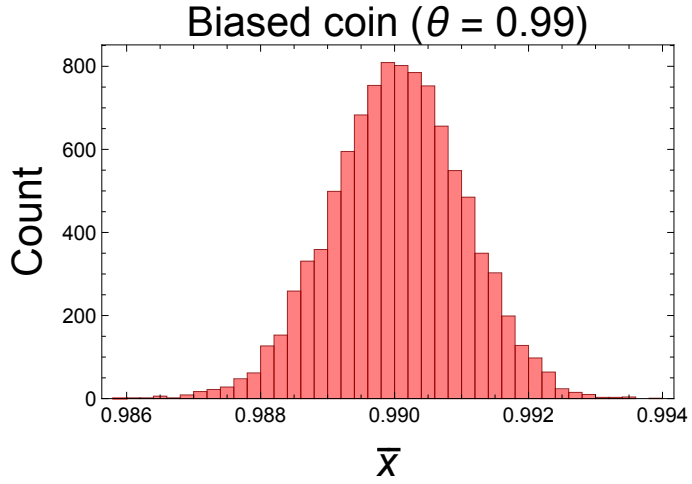
Central limit theorem

A fundamental result in probability theory with many variants, extensions and applications:

Given a sample of n iid random variables $x_i \sim p$ with finite population variance, the sample mean $\bar{x}$ is asymptotically normally distributed as $n \to \infty$ (regardless of p)

In practice, a normal approximation can be poor if the population variance is too large, or if the variables are not iid (e.g., have very different variances), or if n is too small.





The probability density function for the normal distribution is

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right). \tag{33}$$

We focus first on the case of known σ^2 and unknown $\mu \equiv \theta$.

Single-observation data

The likelihood for a single given observation x also has a Gaussian functional form (and is even normalised already), since x and μ are interchangeable in Eq. (33):

$$L(\theta|x) \propto \exp\left(-\frac{1}{2}\frac{(\theta-x)^2}{\sigma^2}\right). \tag{34}$$

Let us choose a Gaussian conjugate prior to simplify calculations:

$$\pi(\theta) = \mathcal{N}(\theta|\mu_0,\sigma_0^2) \propto \exp\left(-\frac{1}{2}\frac{(\theta-\mu_0)^2}{\sigma_0^2}\right), \tag{35}$$

where μ_0, σ_0^2 are the prior mean and variance.

Thus the posterior is also of Gaussian functional form:

$$p(\theta|x) \propto \exp\left(-\frac{1}{2}\left(\frac{(\theta - x)^2}{\sigma^2} + \frac{(\theta - \mu_0)^2}{\sigma_0^2}\right)\right). \tag{36}$$

Exercise

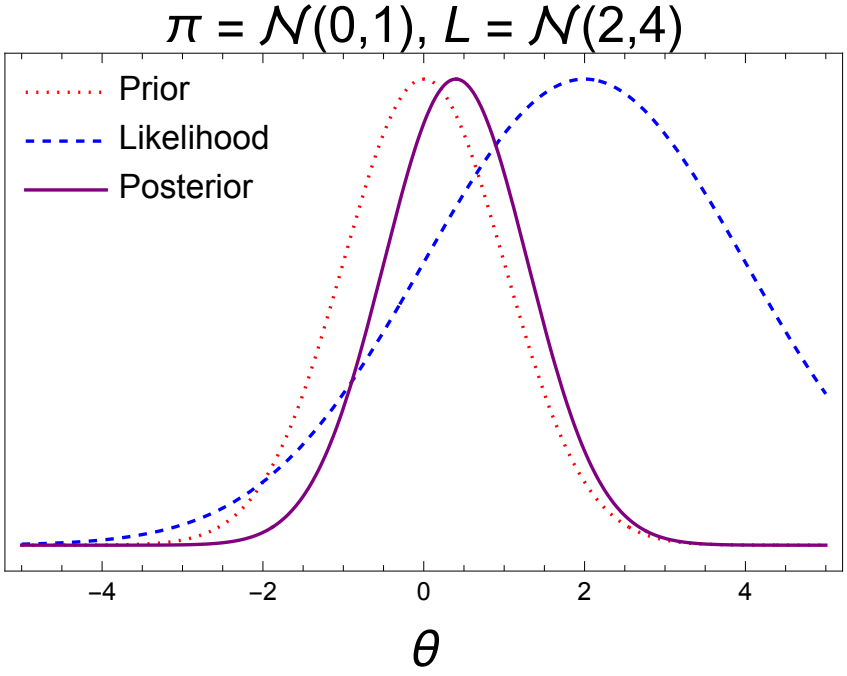
Complete the square in θ and show that

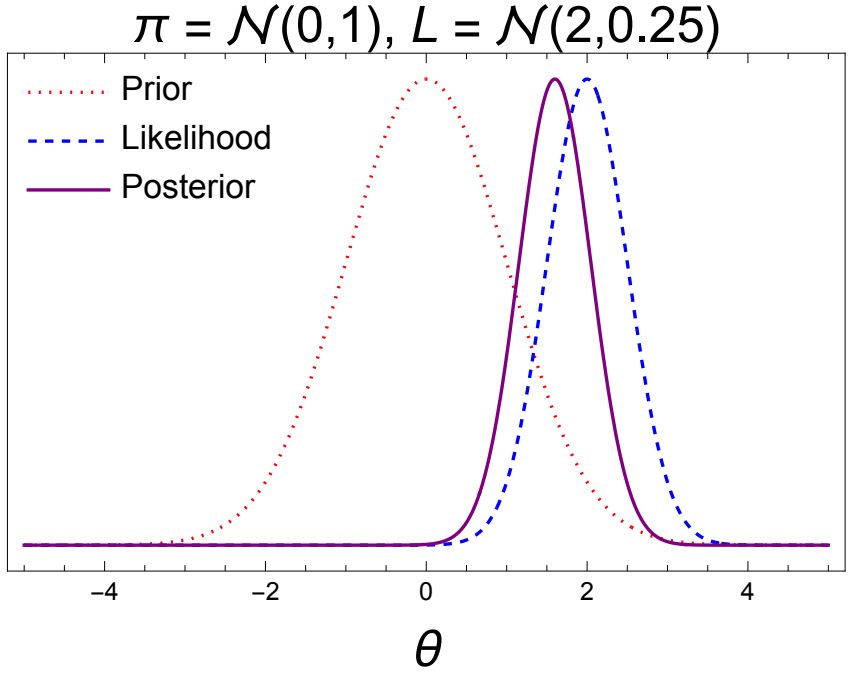
$$p(\theta|x) = \mathcal{N}(\theta|\mu_1, \sigma_1^2), \quad \frac{1}{\sigma_1^2} = \frac{1}{\sigma^2} + \frac{1}{\sigma_0^2}, \quad \frac{\mu_1}{\sigma_1^2} = \frac{x}{\sigma^2} + \frac{\mu_0}{\sigma_0^2}. \tag{37}$$

The inverse variance of a normal distribution is called its *precision*. We see from Eq. (37) that for a normal single-observation likelihood and a normal conjugate prior:

- The posterior precision $1/\sigma_1^2$ is the sum of the likelihood and prior precisions
- The posterior mean μ_1 is the precision-weighted average of the likelihood and prior means

If the likelihood precision is lower than the prior precision (e.g., more uncertainty in the data observations, or stronger prior assumptions), then the prior has a larger effect on the posterior. If the converse is true, then the data speaks more for itself.





The posterior predictive distribution of another observation $\tilde{x}$ is

$$p(\tilde{x}|x) \propto \int_{-\infty}^{\infty} d\theta \, \exp\left(-\frac{1}{2}\frac{(\tilde{x}-\theta)^2}{\sigma^2}\right) \exp\left(-\frac{1}{2}\frac{(\theta-\mu_1)^2}{\sigma_1^2}\right). \tag{38}$$

This is a convolution of two Gaussian functions, which is itself Gaussian.

Exercise

Show without evaluating the integral that $\mathrm{E}[\tilde{x}|x] = \mu_1$ and $\mathrm{Var}[\tilde{x}|x] = \sigma^2 + \sigma_1^2$. Thus

$$p(\tilde{x}|x) = \mathcal{N}(\tilde{x}|\mu_1, \sigma^2 + \sigma_1^2). \tag{39}$$

The variance of the posterior predictive distribution is always greater than the data variance of $\tilde{x}$ (σ^2; aleatoric), since it also contains the posterior variance of θ (σ_1^2; epistemic).

Multiple-observation data

Now consider a set of n given observations $\{x_i\}$ (assumed to be iid). The posterior is

$$p(\theta|\{x_i\}) \propto \prod_i^n p(x_i|\theta)p(\theta). \tag{40}$$

Exercise

From Eqs (36) and (37), show by induction that

$$p(\theta|\{x_i\}) = \mathcal{N}(\theta|\mu_n, \sigma_n^2), \quad \frac{1}{\sigma_n^2} = \frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}, \quad \frac{\mu_n}{\sigma_n^2} = \sum_i^n \frac{x_i}{\sigma^2} + \frac{\mu_0}{\sigma_0^2}. \tag{41}$$

What happens as $\sigma_0^2 \to \infty$? What happens as $n \to \infty$?

Sufficient statistic

Recall that a statistic is a general term for any function of the data, e.g., the sample mean.

The quantity $\sum_i x_i$ in Eq. (41) is called a sufficient statistic for μ in the normal model, since any inference about μ depends on the data only through the value of the statistic (i.e., the full data $x \equiv \{x_i\}$ provides no additional information in any inference about μ).

Sufficient statistics have broader significance in probability theory, but are mainly useful to us here because they simplify notation when generalising to data that comprises multiple iid observations. We denote them by $s(x)$.

For example, in the limit as $\sigma_0^2 \to \infty$ (i.e., an improper constant prior for θ), the multiple-observation posterior for the normal model with known σ^2 is

$$p(\theta|x) = \mathcal{N}\left(\theta \middle| \frac{s(x)}{n}, \frac{\sigma^2}{n}\right), \quad s(x) = \sum_i^n x_i. \tag{42}$$

Other Common Models

Normal distribution (with known μ)

- Likelihood and sufficient statistic

$$L(\theta|x) \propto \theta^{-n/2} \exp\left(-\frac{1}{2}\frac{s(x)}{\theta}\right), \quad s(x) = \sum_i^n (x_i - \mu)^2 \tag{43}$$

- Conjugate prior: Inverse-gamma distribution ($\equiv$ scaled inverse-χ^2 distribution)

$$\pi(\theta) = \text{Inv-Gamma}(\theta|a, b) \propto \theta^{-(a+1)} \exp\left(-\frac{b}{\theta}\right) \tag{44}$$

$$\text{Inv-}\chi^2_\nu(\tau^2) \equiv \text{Inv-Gamma}\left(\frac{\nu}{2}, \frac{\nu\tau^2}{2}\right) \tag{45}$$

Poisson distribution

- Models the discrete number x of events (per unit time) with a constant average rate λ
- Likelihood and sufficient statistic

$$L(\theta|x) \propto \theta^{s(x)} \exp(-n\theta), \quad s(x) = \sum_i^n x_i \tag{46}$$

- Conjugate prior: Gamma distribution

$$\pi(\theta) = \text{Gamma}(\theta|a, b) \propto \theta^{a-1} \exp(-b\theta) \tag{47}$$

5. Multiparameter Models

On Marginalisation

One main benefit of Bayesian modelling and inference is its unified treatment of models with many unknown parameters, and in particular how it handles the task of performing inference on specific parameters of interest (which is often the aim in practical applications).

This is achieved by computing the marginal posterior of such parameters, which allows us to:

- Marginalise over "nuisance" parameters that are needed for a realistic model, but are not of intrinsic interest (e.g., measurement noise)
- Average over different models and assumptions that are indexed by discrete parameters
- Visualise and interpret results for a smaller subset of parameters at a time

For any partition of the parameter set as $\theta = \{\theta_1, \theta_2\}$, the marginal posterior of θ_1 is

$$p(\theta_1|x) = \int d\theta_2\, p(\theta_1, \theta_2|x) = \int d\theta_2\, p(\theta_1|\theta_2, x)p(\theta_2|x). \tag{48}$$

First equality: In non-textbook problems, sampling from the joint posterior $p(\theta_1, \theta_2|x)$ provides the most direct (and often only) way of computing $p(\theta_1|x)$.

Second equality: The factoring of the integrand into the conditional posterior $p(\theta_1|\theta_2, x)$ and the other marginal posterior $p(\theta_2|x)$ is not practical for the numerical computation of $p(\theta_1|x)$. Instead, it suggests an alternative way of sampling from complex joint posteriors if a particular conditional–marginal factorisation involves more tractable distributions.

Normal Distribution

The normal distribution with unknown $(\mu, \sigma^2) \equiv (\theta_1, \theta_2)$ serves as a simple introduction to working with multiparameter models in Bayesian statistics.

Given n iid observations x_i, the joint likelihood for (θ_1, θ_2) is

$$L(\theta_1, \theta_2 | x) \propto \theta_2^{-n/2} \exp\left(-\frac{1}{2} \frac{\sum_i^n (x_i - \theta_1)^2}{\theta_2}\right). \tag{49}$$

Exercise

Write Eq. (49) in terms of the sufficient statistics

$$s_1(x) = \frac{1}{n} \sum_i^n x_i, \quad s_2(x) = \frac{1}{n-1} \sum_i^n (x_i - s_1(x))^2. \tag{50}$$

Uninformative prior

Let us first consider a simple uninformative joint prior for illustrative purposes:

- Assume that μ has an improper constant prior on $\mathbb{R}$ (which is the limit of the conjugate prior $\mathcal{N}(\mu_0, \sigma_0^2)$ as $\sigma_0^2 \to \infty$)
- Assume that σ^2 has an improper reciprocal prior on $\mathbb{R}^+$ (which is the limit of the conjugate prior $\text{Inv-}\chi^2_\nu(\tau^2)$ as $\nu \to 0$)
- Assume that μ and σ^2 are independent (which is sensible)

Putting it all together, the (improper) joint prior is

$$\pi(\theta_1, \theta_2) \propto \theta_2^{-1}. \tag{51}$$

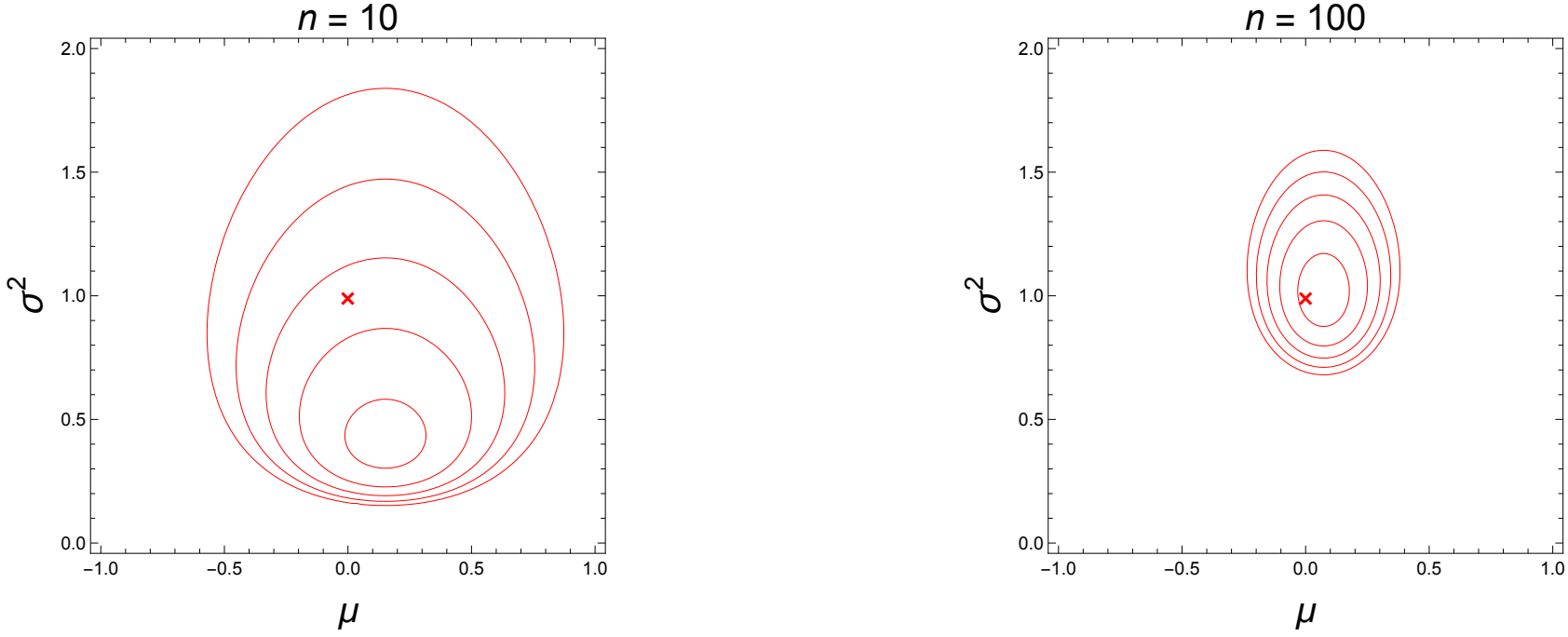
We can now immediately write down the joint posterior, up to normalisation:

$$p(\theta_1, \theta_2 | x) \propto \theta_2^{-(n+2)/2} \exp\left(-\frac{1}{2}\frac{(n-1)s_2(x) + n(s_1(x) - \theta_1)^2}{\theta_2}\right). \tag{52}$$

This turns out to be proper for $n > 1$. How should we visualise it?

In two dimensions, level sets (contours) of the posterior density are the easiest to work with. These are equivalent to level sets of the posterior log-density, which can be used instead to avoid numerical underflow in low probability regions.

As an example, let the data distribution be $\mathcal{N}(0, 1)$, and consider two data realisations with $n = 10$ and $n = 100$ observations. The sufficient statistics evaluate to $(s_1, s_2) = (0.15, 0.55)$ for the first realisation, and to $(s_1, s_2) = (0.07, 1.04)$ for the second. This gives the following log-density contour plots for the joint posterior:



The posterior for the smaller data set has a larger bias and uncertainty, as might be expected.

For higher-dimensional problems, samples from the posterior are typically required — i.e., a set of parameter values that is distributed according to the posterior density. These can be used to construct multidimensional histogram distributions, in which parameters are straightforward to marginalise over (simply combine bins with the same parameter ranges).

Although the joint posterior here is not a standard two-parameter distribution, drawing samples from it with general methods is still straightforward due to the low dimensionality. However, the conditional–marginal factorisation in Eq. (48) admits an even quicker sampling method:

Sampling from joint distribution $\equiv$ pre-sampling from marginal distribution, then sampling from conditional distribution given those pre-samples

First, the conditional posterior $p(\theta_1|\theta_2, x)$ has already been derived in Eq. (42); it is simply the posterior for the normal model with known θ_2 and a constant prior for θ_1:

$$p(\theta_1|\theta_2, x) = \mathcal{N}\left(\theta_1 \middle| s_1(x), \frac{\theta_2}{n}\right) \tag{53}$$

(Note that the sufficient statistic for μ in Eq. (42) was the sample total, not the sample mean.)

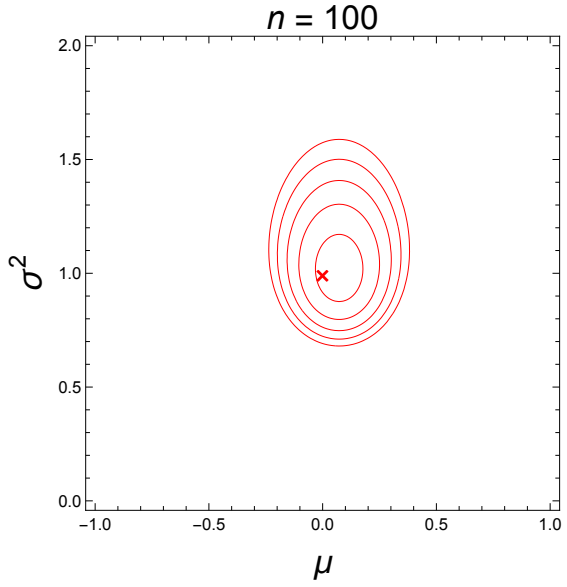
Next, the marginal posterior $p(\theta_2|x)$ is obtained by integrating Eq. (52) over θ_1.

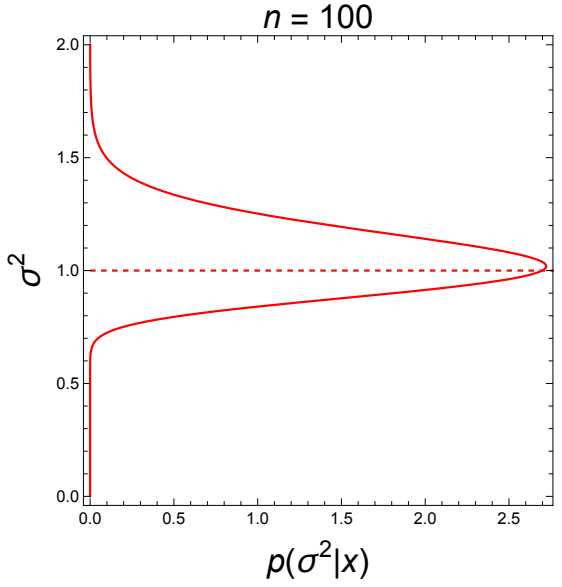
Exercise

Show that

$$p(\theta_2|x) \propto \theta_2^{-(n+1)/2} \exp\left(-\frac{1}{2}\frac{(n-1)s_2(x)}{\theta_2}\right). \tag{54}$$

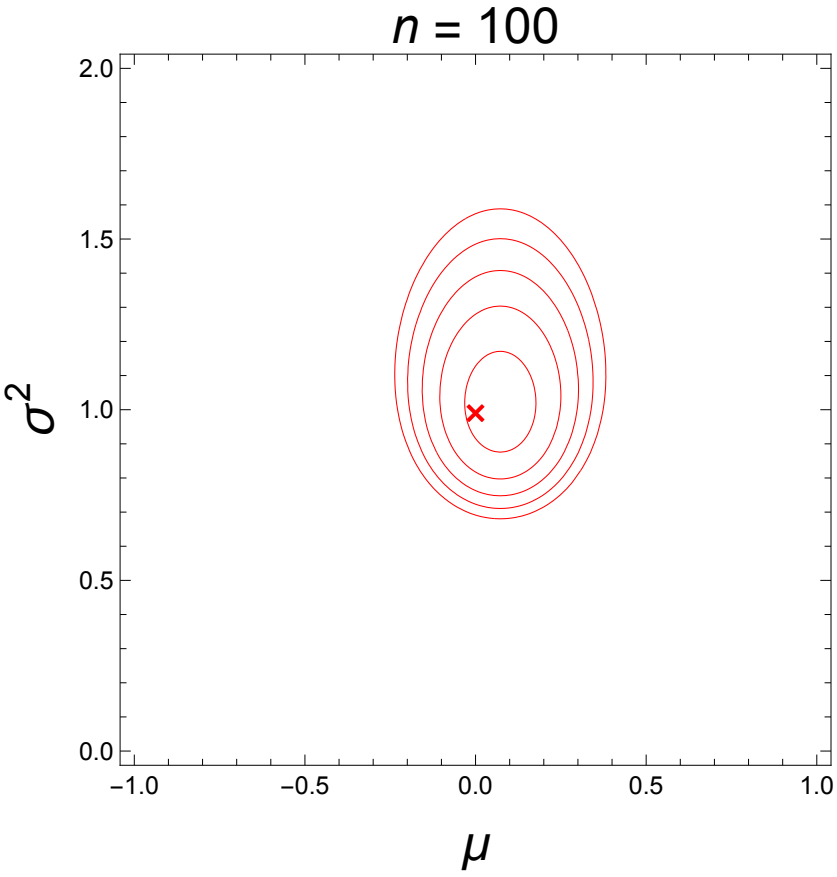
This corresponds exactly to a scaled inverse-χ^2 distribution: $\theta_2|x \sim \text{Inv-}\chi^2_{n-1}(s_2(x))$.

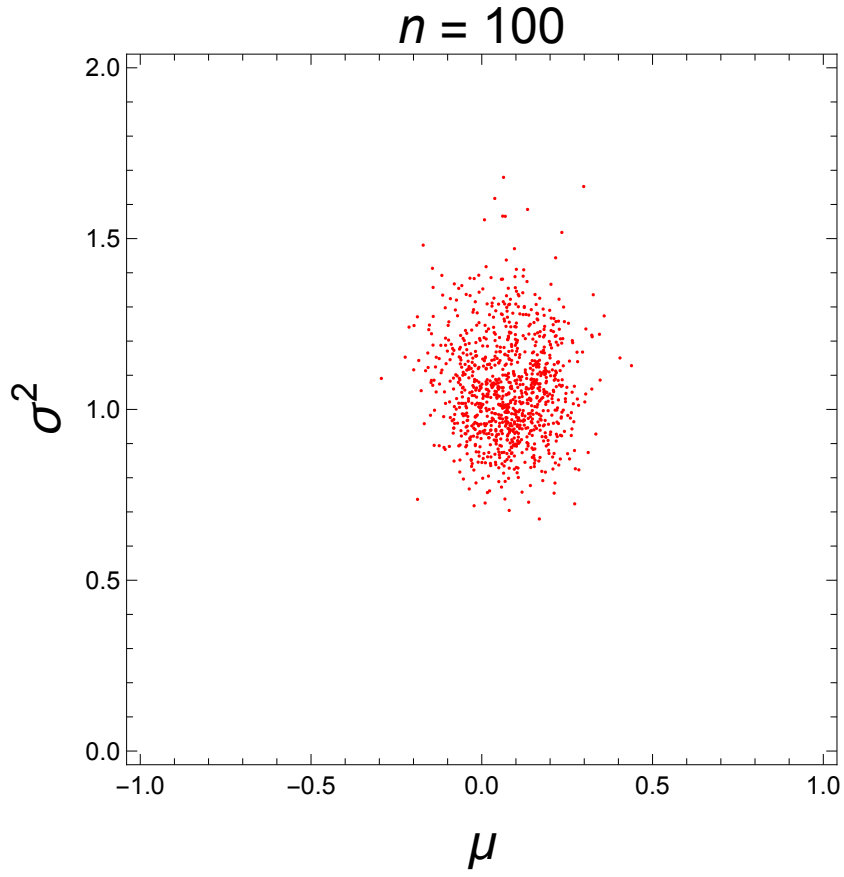




As the scaled inverse-χ^2 distribution is a common one-parameter distribution, it is straightforward to sample from using standard scientific computing libraries.

Thus the sampling of the joint posterior is also trivial:





Considering the parameters in reverse order, a similar analysis yields

$$\theta_2|\theta_1, x \sim \text{Inv-}\chi^2_n \left(\frac{(n-1)s_2(x)}{n} + (s_1(x) - \theta_1)^2 \right), \quad \theta_1|x \sim t_{n-1}\left(s_1(x), \frac{s_2(x)}{n} \right), \quad (55)$$

where $t_\nu(\xi, \tau^2)$ is the generalised t distribution with location ξ and (squared) scale τ^2.

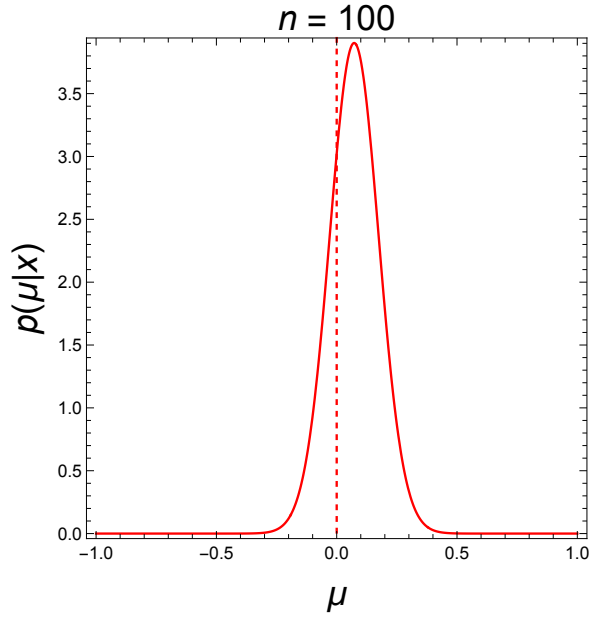
Note that the conditional posterior of the variance (known mean) is an $\text{Inv-}\chi^2_n$ distribution, while the marginal posterior of the variance (unknown mean) is an $\text{Inv-}\chi^2_{n-1}$ distribution.

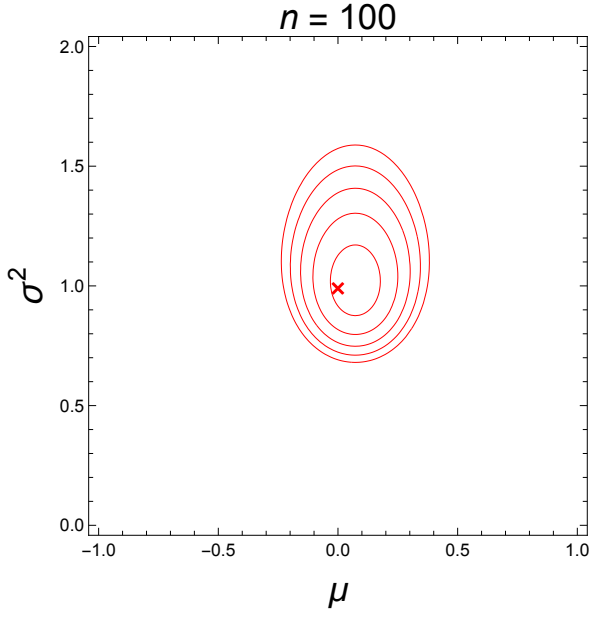
- This is akin to an inversion of a result in frequentist statistics: the sampling distribution of a variance estimator with the population mean is χ^2_n-distributed, while the sampling distribution of a variance estimator with the sample mean is χ^2_{n-1}-distributed.

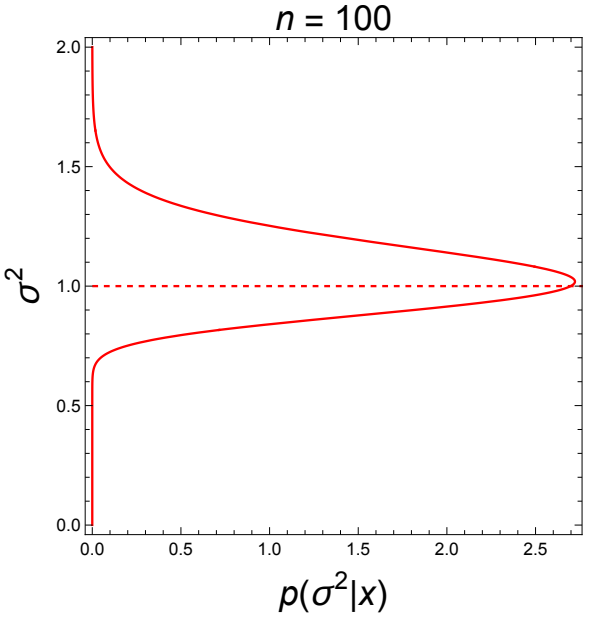
The marginal posterior of θ_1 can be derived via a change of variable from θ_2 to the (negative) exponent in the joint posterior, which transforms the integral of Eq. (52) over θ_2 into a standard gamma integral. The explicit form of the marginal posterior density is

$$p(\theta_1|x) \propto \left(1 + \frac{n(\theta_1 - s_1(x))^2}{(n-1)s_2(x)} \right)^{-n/2}. \quad (56)$$

(Alternatively, the quantity $(\theta_1 - s_1)/\sqrt{s_2/n}$ is t-distributed with $n-1$ d.o.f.)


n = 100
p(μ|x)
μ


n = 100
σ²
μ


n = 100
σ²
p(σ²|x)

Finally, the posterior predictive distribution of another (single) observation $\tilde{x}$ is

$$p(\tilde{x}|x) = \int_0^\infty d\theta_2 \int_{-\infty}^\infty d\theta_1 \, p(\tilde{x}|\theta_1, \theta_2) p(\theta_1, \theta_2|x), \tag{57}$$

where the integrand is a Gaussian density times the (non-standard) joint posterior density.

But the conditional–marginal factorisation of the joint posterior gives

$$p(\tilde{x}|x) = \int_0^\infty d\theta_2 \int_{-\infty}^\infty d\theta_1 \, p(\tilde{x}|\theta_1, \theta_2) p(\theta_1|\theta_2, x) p(\theta_2|x) = \int_0^\infty d\theta_2 \, p(\tilde{x}|\theta_2, x) p(\theta_2|x), \tag{58}$$

where the inner integral over θ_1 is just a convolution of Gaussians:

$$p(\tilde{x}|\theta_2, x) = \int_{-\infty}^\infty d\theta_1 \, \mathcal{N}(\tilde{x}|\theta_1, \theta_2) \mathcal{N}\left(\theta_1 \middle| s_1(x), \frac{\theta_2}{n}\right) = \mathcal{N}\left(\tilde{x} \middle| s_1(x), \left(1 + \frac{1}{n}\right) \theta_2\right). \tag{59}$$

Thus the posterior predictive distribution works out to

$$p(\tilde{x}|x) = t_{n-1}\left(\tilde{x} \middle| s_1(x), \left(1 + \frac{1}{n}\right) s_2(x)\right), \tag{60}$$

where the two "variance" terms correspond to aleatoric and epistemic uncertainty respectively.

Conjugate prior

The analysis so far has used a simple uninformative prior, but is straightforward to generalise to a conjugate prior. Let us factor the joint likelihood in Eq. (49) more explicitly as

$$L(\theta_1, \theta_2|x) \propto \exp\left(-\frac{1}{2}\frac{n(s_1(x)-\theta_1)^2}{\theta_2}\right) \times \theta_2^{-n/2} \exp\left(-\frac{1}{2}\frac{(n-1)s_2(x)}{\theta_2}\right) \tag{61}$$

Thus the conjugate joint prior should also admit the factorisation $p(\theta_1, \theta_2) = p(\theta_1|\theta_2)p(\theta_2)$, where $p(\theta_1|\theta_2)$ is a Gaussian density and $p(\theta_2)$ is an $\text{Inv-}\chi^2$ density:

$$\theta_1|\theta_2 \sim \mathcal{N}\left(\mu_0, \frac{\theta_2}{\kappa_0}\right), \quad \theta_2 \sim \text{Inv-}\chi^2_{\nu_0}(\sigma_0^2), \tag{62}$$

$$\pi(\theta_1, \theta_2) \propto f(\theta_1, \theta_2|\mu_0, \kappa_0, \nu_0, \sigma_0) := \theta_2^{-(\nu_0+3)/2} \exp\left(-\frac{1}{2}\frac{\nu_0\sigma_0^2 + \kappa_0(\mu_0-\theta_1)^2}{\theta_2}\right). \tag{63}$$

(Note that μ and σ^2 are no longer assumed to be independent here, so the previously considered uninformative prior $\pi(\theta_1, \theta_2) \propto \theta_2^{-1}$ is not a limiting case of f.)

With this parametrisation, the joint posterior and the two marginal posteriors may be computed similarly to before, and written succinctly as

$$p(\theta_1, \theta_2|x) \propto f(\theta_1, \theta_2|\mu_n, \kappa_n, \nu_n, \sigma_n), \tag{64}$$

$$p(\theta_1|x) = t_{\nu_n}\left(\theta_1 \middle| \mu_n, \frac{\sigma_n^2}{\kappa_n}\right), \tag{65}$$

$$p(\theta_2|x) = \text{Inv-}\chi^2_{\nu_n}(\theta_2|\sigma_n^2), \tag{66}$$

where

$$\kappa_n := \kappa_0 + n, \tag{67}$$

$$\nu_n := \nu_0 + n, \tag{68}$$

$$\mu_n(x) := \frac{\kappa_0}{\kappa_n}\mu_0 + \frac{n}{\kappa_n}s_1(x), \tag{69}$$

$$\nu_n\sigma_n^2(x) := \nu_0\sigma_0^2 + (n-1)s_2(x) + \frac{\kappa_0 n}{\kappa_n}(s_1(x) - \mu_0)^2. \tag{70}$$

The separate contributions of the prior and data to the posterior are made explicit in these expressions. For example, the posterior location μ_n is the (relative-)precision-weighted average of the prior location μ_0 and sample mean s_1, while the posterior scale σ_n^2 combines the prior scale σ_0^2, the sample variance s_2, and the additional uncertainty due to $s_1 - \mu_0$.

Multinomial Distribution (with Known n)

The multinomial distribution is a generalisation of the binomial distribution: it models a sequence of n iid trials, each with an outcome in a set of k possible outcomes (with associated probabilities $\theta \equiv \{\theta_i\}$). When $n = 1$, it is called the categorical distribution.

Let x_i be the number of times outcome i is observed. The data distribution is

$$p(x|\theta) = \text{Multin}(x|n, \theta) = \frac{n!}{\prod_i^k x_i!} \prod_i^k \theta_i^{x_i}, \tag{71}$$

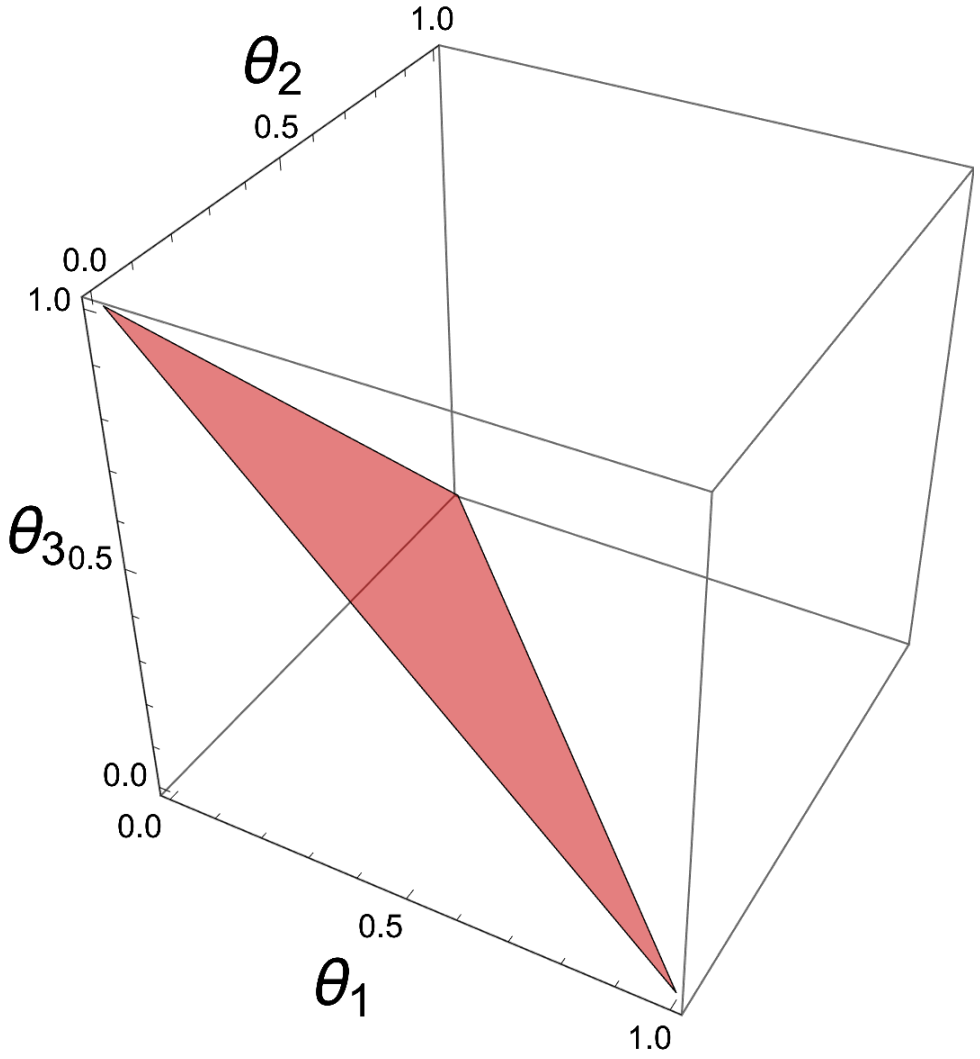
where $\sum_i^k \theta_i = 1$ and $\sum_i^k x_i = n$. Thus the likelihood is

$$L(\theta|x) \propto \prod_i^k \theta_i^{x_i}. \tag{72}$$

A suitable conjugate prior is then the multivariate generalisation of the beta distribution, known as the Dirichlet distribution:

$$\pi(\theta) = \text{Dirichlet}(\theta|\alpha) = \frac{\Gamma(\sum_i^k \alpha_i)}{\prod_i^k \Gamma(\alpha_i)} \prod_i^k \theta_i^{\alpha_i - 1}. \tag{73}$$

Note that the support of the Dirichlet distribution is not $[0,1]^k$, but the *probability simplex* $\{\theta \in [0,1]^k : \sum_i^k \theta_i = 1\}$. The uniform distribution on this space is $\mathrm{Dirichlet}(1,\ldots,1)$.



Exercise

Show that for $\theta \sim \mathrm{Dirichlet}(\theta|\alpha)$, the marginal distribution of each variable is

$$\theta_i \sim \mathrm{Beta}\left(\alpha_i, \sum_{j \neq i}^{k} \alpha_j\right). \tag{74}$$

By analogy to the beta conjugate prior in the binomial model, $\alpha_i - 1$ in the Dirichlet prior can be interpreted as the prior number of times outcome i is observed.

With the Dirichlet prior, the (joint) posterior of θ is simply

$$p(\theta|x) = \text{Dirichlet}(\theta|\alpha + x). \tag{75}$$

Example

I roll a die 100 times. It lands on 1 20 times, and 2–6 16 times each. What is the posterior probability that the probability of rolling a 1 with this die is more than 1/6?

Multinormal Distribution

The multinormal distribution is the multivariate generalisation of the normal distribution, and is used to describe a d-vector of Gaussian random variables that are generally neither independent nor identically distributed. It is fully specified by a mean vector μ of length d, and a $d \times d$ covariance matrix Σ (which is symmetric and positive-definite).

- A Gaussian process is a further generalisation in the limit as $d \to \infty$. Multinormal distributions naturally play a central role in practical applications of Gaussian processes.

For multinormally distributed data x with unknown $\{\mu, \Sigma\} \equiv \theta$, the data distribution is

$$p(x|\theta) = \mathcal{N}(x|\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^d \det \Sigma}} \exp\left(-\frac{1}{2}[x-\mu]^T \Sigma^{-1} [x-\mu]\right), \tag{76}$$

and the likelihood is

$$L(\theta|x) \propto (\det \Sigma)^{-1/2} \exp\left(-\frac{1}{2}[x-\mu]^T \Sigma^{-1} [x-\mu]\right). \tag{77}$$

The special case $\Sigma = \sigma^2 I_d$ corresponds superficially to the normal model with multiple-observation data, but note the conceptual difference — we consider here a single observation of a d-vector observable, rather than d iid observations of a scalar observable.

But now we may also consider n iid observations $x \equiv \{x_i\}$ of the vector observable, which yields the multiple-observation multivariate likelihood

$$L(\theta|x) \propto (\det \Sigma)^{-n/2} \exp\left(-\frac{1}{2}\sum_i^n [x_i - \mu]^T \Sigma^{-1} [x_i - \mu]\right). \tag{78}$$

Exercise

Show that

$$\sum_i^n [x_i - \mu]^T \Sigma^{-1} [x_i - \mu] = \mathrm{tr}\left(\Sigma^{-1} \sum_i^n [x_i - \mu][x_i - \mu]^T\right). \tag{79}$$

The remaining analysis follows by analogy to the univariate case. As in Eq. (61), the likelihood factorises into a conditional normal density for $\mu|\Sigma$ and a marginal inverse-Wishart density for Σ (a multivariate generalisation of the scaled inverse-χ^2 density), with the sufficient statistics

$$s_1(x) = \frac{1}{n}\sum_i^n x_i, \quad s_2(x) = \sum_i^n [x_i - s_1(x)][x_i - s_1(x)]^T. \tag{80}$$

This factorisation invokes a conjugate prior with a similar form to Eq. (63):

$$\mu|\Sigma \sim \mathcal{N}\left(\mu_0, \frac{\Sigma}{\kappa_0}\right), \quad \Sigma \sim \text{Inv-Wishart}_{\nu_0}(\Lambda_0^{-1}), \tag{81}$$

$$\begin{aligned}\pi(\mu, \Sigma) &\propto F(\mu, \Sigma|\mu_0, \kappa_0, \nu_0, \Lambda_0) \\ &:= (\det \Sigma)^{-(\nu_0+d+2)/2} \exp\left(-\frac{1}{2}\text{tr}((\Lambda_0 + \kappa_0[\mu_0 - \mu][\mu_0 - \mu]^T)\Sigma^{-1})\right),\end{aligned} \tag{82}$$

where Λ denotes (the inverse of) the scale matrix of the inverse-Wishart distribution.

The joint posterior and the two marginal posteriors are then

$$p(\mu, \Sigma|x) \propto F(\mu, \Sigma|\mu_n, \kappa_n, \nu_n, \Lambda_n), \tag{83}$$

$$p(\mu|x) = t_{\nu_n - d + 1}\left(\mu \middle| \mu_n, \frac{\Lambda_n}{\kappa_n(\nu_n - d + 1)}\right), \tag{84}$$

$$p(\Sigma|x) = \text{Inv-Wishart}_{\nu_n}(\Sigma|\Lambda_n^{-1}), \tag{85}$$

where the t distribution has been replaced with its multivariate variant, and

$$\kappa_n := \kappa_0 + n, \tag{86}$$

$$\nu_n := \nu_0 + n, \tag{87}$$

$$\mu_n(x) := \frac{\kappa_0}{\kappa_n}\mu_0 + \frac{n}{\kappa_n}s_1(x), \tag{88}$$

$$\Lambda_n(x) := \Lambda_0 + s_2(x) + \frac{\kappa_0 n}{\kappa_n}[s_1(x) - \mu_0][s_1(x) - \mu_0]^T. \tag{89}$$

6. Hierarchical Models

Motivation

Consider an example where the data x comprises the outcomes of multiple Bernoulli trials, e.g., the survival or death of a lab rat after some drug is administered.

Say we want to infer the probability of survival. As a first cut, we might describe x using a binomial model where θ is the overall probability of survival, and then compute $p(\theta|x)$.

But is θ well defined in realistic scenarios? It is not too meaningful to talk about a fixed overall probability if there are significant differences among rats, experiments, labs, etc.

Let us assume that all rats and experiments are identical, but that the data comes from two labs (with slightly different conditions). Then we might take the following steps:

1. Partition the data as $\{x_1, x_2\}$
2. Partition the model into binomial sub-models with parameters $\{\theta_1, \theta_2\}$
3. Perform inference on $\{\theta_1, \theta_2\}$
4. Report $p(\theta_1|x_1)$ and $p(\theta_2|x_2)$, or some average thereof

What if there are $N \gg 1$ labs, or a total of N slightly different experiments over the two labs?

- We might take the above approach, but this may cease to be practical or useful.
- Is it always valid to assume the θ_i are completely independent?
- Note also that with a fixed amount of data x, the amount of data x_i with which to perform inference on each θ_i goes down (limiting case: rats are not identical).

Solution: Bayesian statistics provides a hierarchical framework for the probabilistic modelling of not just the data, but the model parameters too.

In hierarchical modelling, we describe the parameters θ as random variables (not necessarily iid) whose prior distribution is conditioned on some *hyperparameters* ϕ, i.e., $\pi(\theta) = p(\theta|\phi)$.

How do we "know" the family of prior distributions $p(\theta|\phi)$ to use?

- Recall that choosing the prior is an assumption, like assuming the data distribution; again, critics might say this "adds" subjectivity — but it is at least explicit!
- Recall also that a simple model is often the most appropriate, e.g., a normal distribution.
- We have been working with parametrised priors already, e.g., $\phi \equiv (a, b)$ in the beta conjugate prior for the binomial model.

How do we "know" the hyperparameter values ϕ to use?

- Assume a fixed value of ϕ, e.g., by choosing a uniform prior. This approach is still Bayesian at the level of the parameters, but is more arbitrary and not hierarchical.
- Choose ϕ based on prior data if available, e.g., a beta prior for the binomial model where (a, b) is based on prior successes and failures. This approach is known as "empirical Bayes"; it is less arbitrary, but the distinction between prior and actual data is still ad hoc.
- Infer the "hyperposterior" $p(\phi|x)$ from the data. This is the fully Bayesian approach, and allows us to do things like marginalise over ϕ in the posterior $p(\theta|x)$.

Philosophical question: Since ϕ is supposed to describe *a priori* knowledge about θ (before the data), does it actually make sense to estimate ϕ *a posteriori* (after the data)?

On Exchangeability

Recall that an indexed set of observations $\{x_i\}$ is said to be exchangeable if its joint distribution is invariant to index permutation.

Example

Consider $(x_1, x_2) \sim \mathcal{N}(\mu, \Sigma)$. Show that x_1 and x_2 are exchangeable if $\mu = 0$ and $\operatorname{diag} \Sigma = (1, 1)$. Show that they are additionally iid if $\Sigma = I_2$.

More generally, the exchangeability of random variables:

- Is a weaker condition than that of iid (i.e., iid $\implies$ exchangeability); for example, the components of a random unit vector are exchangeable but not iid
- Is a valid assumption when there is ignorance about the distinguishability of variables
- Simplifies probability models greatly, due to symmetry

Example

I draw a stone twice from a bag of Go stones. Are the outcomes x_1 and x_2 exchangeable/iid in the following scenarios?

1. The bag contains 1 black stone and 1 white stone; draws are done with replacement.
2. The bag contains 1 black stone and 1 white stone; draws are done without replacement.
3. The bag contains 1000 black stones and 1000 white stones; draws are done without replacement.
4. The bag contains b black stones and w white stones; draws are done with replacement.
5. The bag contains b black stones and w white stones; draws are done without replacement.
6. The bag contains $b \gg 1$ black stones and $w \gg 1$ white stones; draws are done without replacement.

The assumption of exchangeability is often valid for data $x \equiv \{x_i\}$ in practice, and can usually be applied to the model parameters $\theta \equiv \{\theta_i\}$ as well.

Let us return to the lab rat example with N different labs/experiments. We denote the survival probability for each lab/experiment by θ_i, and the outcome for each rat by $x_{ij} \equiv x_k$ (where the combined index k is agnostic to i, j).

- If we assume that the difference in labs/experiments has negligible impact on the data, then the x_k are exchangeable and the standard binomial model would suffice.
- If we still have reason to believe that the x_k are not exchangeable, we may add random/fixed covariates until they are (e.g., the rats' ages y_k might make a significant difference, but (x_k, y_k) or $x_k|y_k$ is still exchangeable).
- If we think that the difference in labs/experiments will affect the data but have no information to distinguish them, then the x_k are not exchangeable but both the θ_i and $x_{ij}|\theta_i$ are, and we can construct a hierarchical model that reflects this.
- If we have information to distinguish the labs/experiments (e.g., a ranking of lab conditions or experimental quality), then both the x_k and θ_i are not exchangeable but the $x_{ij}|\theta_i$ are. We could still turn this into an exchangeable hierarchical model by treating such information as covariates for x_k or θ_i.

The upshot: Exchangeability allows us to construct priors $p(\theta|\phi)$ that are simple, rather than having to describe complex relationships between the θ_i with many hyperparameters.

Binomial–Beta Model

The general aim of hierarchical Bayesian inference is to infer the joint posterior $p(\theta, \phi|x)$, from which the marginal posterior $p(\theta|x)$ can be computed.

- Depending on context, the marginal hyperposterior $p(\phi|x)$ might also be of interest.
- Another application is to obtain the posterior predictive distribution of the data: $p(\tilde{x}|x) = \int d(\theta, \phi)\, p(\tilde{x}|\theta, \phi)p(\theta, \phi|x)$.
- It is potentially useful to look at $p(\tilde{x}|\tilde{\theta}, x)$ as well, where $\tilde{\theta}$ is drawn from the posterior predictive distribution of the parameters: $p(\tilde{\theta}|x) = \int d\phi\, p(\tilde{\theta}|\phi)p(\phi|x)$.

Focus then on the joint posterior $p(\theta, \phi|x)$, which factorises as

$$p(\theta, \phi|x) \propto p(x|\theta, \phi)p(\theta, \phi) = L(\theta|x)\pi(\theta|\phi)p(\phi), \tag{90}$$

where L is not a function of ϕ since x depends on ϕ only through θ.

Again, we seem to have written down the answer right at the start, but the joint posterior is generally not a standard distribution even if L, π and the "hyperprior" $p(\phi)$ are simple. For realistic problems, we will need to draw samples from $p(\theta, \phi|x)$ in order to do anything with it.

For illustrative purposes, we examine now a relatively simple binomial–beta example, where parts of the model can be solved for analytically with a conjugate prior.

Consider the lab rat example with $N = 10$ different experiments, and let x_i denote the observed number of surviving rats in experiment i. Assume that all of the x_i are independent (but not identically distributed) variables, and that each has a binomial distribution:

$$x_i \sim \mathrm{Bin}(n_i, \theta_i), \tag{91}$$

where n_i is known. We set $n_i = n = 10$ here for convenience.

Let θ_i denote the underlying probability of survival in experiment i. Assume that all of the θ_i are iid variables, and that each is distributed according to the same beta conjugate prior:

$$\theta_i \sim \mathrm{Beta}(a, b). \tag{92}$$

Finally, assume an uninformative hyperprior:

$$p(a, b) \propto (a + b)^{-5/2}, \tag{93}$$

which is improper but turns out to give a proper hyperposterior.

We will find it convenient to work in terms of transformed hyperparameters:

$$\phi \equiv (\phi_1, \phi_2) := \left(\ln\left(\frac{a}{b}\right), \ln(a+b) \right) \iff (a,b) = \left(\frac{e^{\phi_1+\phi_2}}{1+e^{\phi_1}}, \frac{e^{\phi_2}}{1+e^{\phi_1}} \right), \tag{94}$$

$$\left| \det \frac{\partial(a,b)}{\partial(\phi_1,\phi_2)} \right| = ab. \tag{95}$$

Exercise

Write down the joint posterior $p(\theta, \phi|x)$, but leave the RHS in terms of (a, b) for brevity.

We see that the marginal posterior $p(\theta|x)$ is not trivial to compute analytically, but the marginal hyperposterior $p(\phi|x)$ is available due to the conjugacy of the prior.

From the conditional–marginal factorisation, we may write

$$p(\phi|x) = \frac{p(\theta, \phi|x)}{p(\theta|\phi, x)}. \tag{96}$$

As all of the θ_i are independent, the conditional posterior $p(\theta|\phi, x)$ is the product of the individual conditional posteriors $p(\theta_i|\phi, x_i)$ (i.e., Eq. (30)):

$$p(\theta|\phi, x) = \prod_i^N p(\theta_i|\phi, x_i) = \prod_i^N \frac{\Gamma(n+a+b)}{\Gamma(x_i+a)\Gamma(n-x_i+b)} \theta_i^{x_i+a-1}(1-\theta_i)^{n-x_i+b-1}. \tag{97}$$

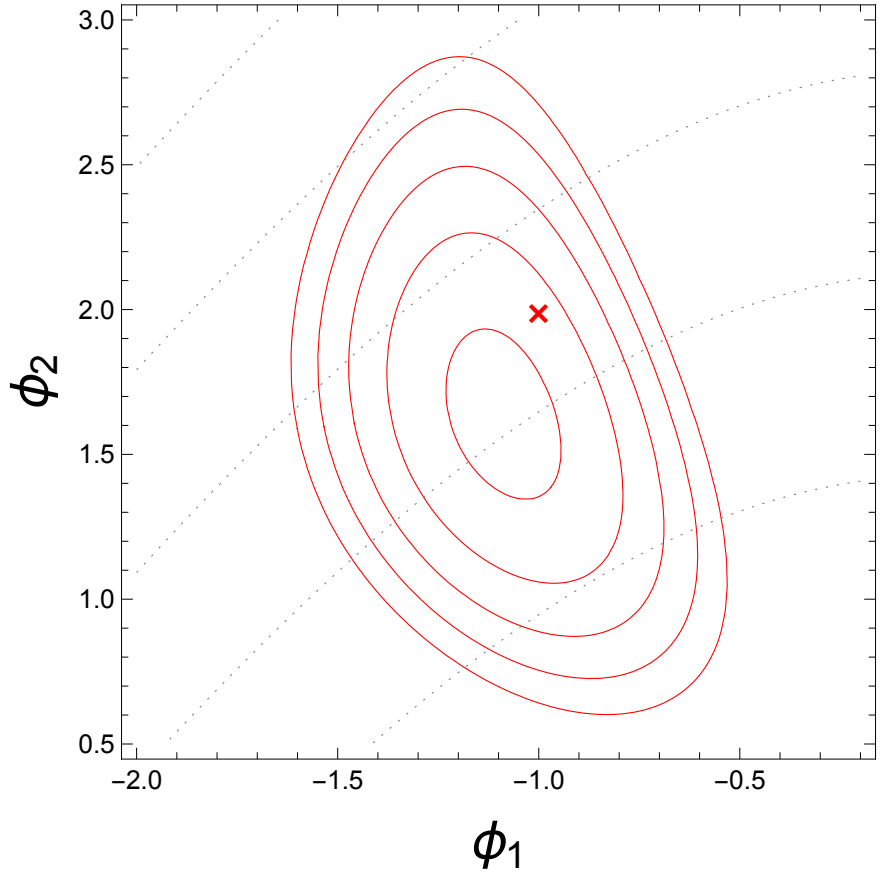
Thus the θ-dependence cancels neatly out of Eq. (96), and we have

$$p(\phi|x) \propto ab(a+b)^{-5/2} \prod_i^N \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \frac{\Gamma(x_i+a)\Gamma(n-x_i+b)}{\Gamma(n+a+b)}. \tag{98}$$

Consider now a parameter realisation where the "true" hyperparameters are $\phi = (-1, 2)$ (i.e., $\theta_i \sim \mathrm{Beta}(2.0, 5.4)$; $\mathrm{E}[\theta_i] = 0.27$), and a data realisation conditioned on those parameters:

$$x = (1, 0, 5, 1, 7, 1, 2, 1, 2, 3). \tag{99}$$

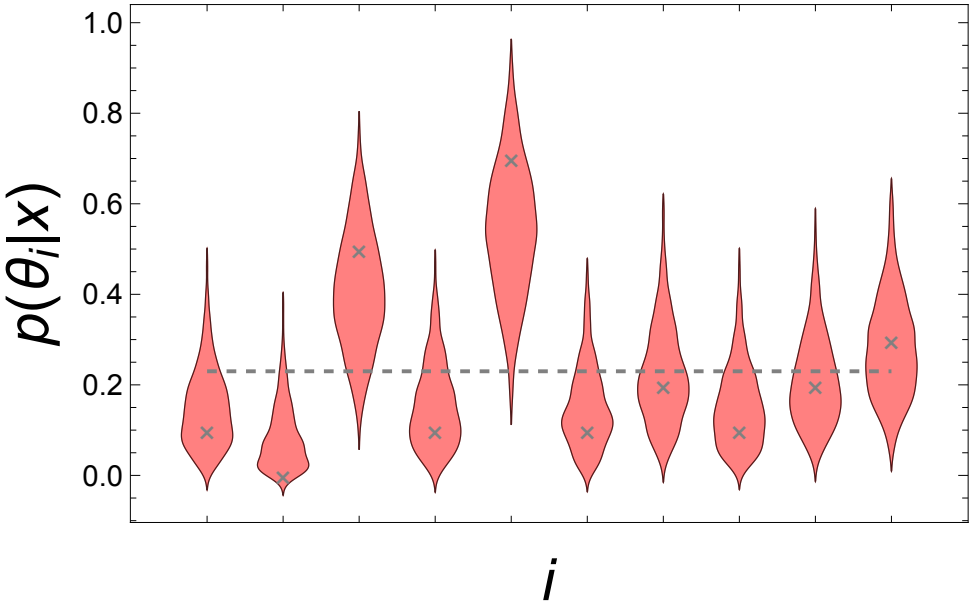
We may start our analysis by examining a log-density contour plot of the marginal hyperposterior $p(\phi|x)$ (the dotted grey lines indicate contours of the uninformative hyperprior).



In the example of the (two-parameter) normal model, it was trivial to sample from the posterior due to its conditional–marginal factorisation into two standard distributions. This is not available for $p(\phi|x)$, so more general methods are required to sample from it.

Once those samples are obtained, sampling from $p(\theta, \phi|x)$ and thus $p(\theta|x)$ is straightforward:

1. Transform each ϕ sample back into (a, b)
2. Draw the θ_i from their conditional posteriors $p(\theta_i|a, b, x_i) = \mathrm{Beta}(\theta|x_i + a, n - x_i + b)$
3. Marginalise over ϕ by grouping all of the samples for each θ_i



The grey crosses here are the values of the frequentist point estimators $\hat{\theta}_i := x_i/n$, while the dashed grey line indicates the value of the combined point estimator $\hat{\theta} := \sum_i^N \hat{\theta}_i/N$.

Normal Model

We may also use drugged lab rats as an example for a hierarchical normal model, where both the data and parameters are normally distributed.

Say the drug that is being administered is no longer potentially lethal, but does have a nonzero effect on the size of the rats. Again consider N different experiments, but let x_{ij} denote the change in the size of rat j in experiment i (e.g., its relative change in weight after 3 days).

- We allow the (known) number of rats n_i in each experiment to be different here.
- We assume the x_{ij} are normally distributed with unknown means θ_i and a common known variance σ^2 (e.g., the measurement process is standardised).

Then the data $x \equiv \{x_{ij}\}$ can be summarised by the sufficient statistics

$$s_i = \frac{1}{n_i} \sum_j^{n_i} x_{ij}, \tag{100}$$

which are the sample means of each experiment. So we work solely with $x \equiv \{s_i\}$, and write:

$$s_i \sim \mathcal{N}(\theta_i, \sigma_i^2), \tag{101}$$

where $\sigma_i^2 = \sigma^2/n_i$ is known. (To be crystal clear, σ_i^2 is the sampling variance of the statistic s_i, and not the sample variance of experiment i.)

Now assume that all of the θ_i are iid variables, and that each is distributed according to the same normal conjugate prior:

$$\theta_i \sim \mathcal{N}(\phi_1, \phi_2). \tag{102}$$

Also assume an uninformative hyperprior:

$$p(\phi_1, \phi_2) \propto 1, \tag{103}$$

which is improper but turns out to give a proper hyperposterior.

Exercise

Write down the joint posterior $p(\theta, \phi|x)$.

We proceed as in the binomial–beta case, and perform the conditional–marginal factorisation

$$p(\theta, \phi|x) = p(\theta|\phi, x)p(\phi|x). \tag{104}$$

As all of the θ_i are independent, the conditional posterior $p(\theta|\phi, x)$ is the product of the individual conditional posteriors $p(\theta_i|\phi, x_i)$ (i.e., Eq. (37)):

$$p(\theta|\phi, x) = \prod_i^N p(\theta_i|\phi, s_i) = \prod_i^N \mathcal{N}(\theta_i|\phi_{1;i}, \phi_{2;i}), \tag{105}$$

$$\frac{1}{\phi_{2;i}} = \frac{1}{\sigma_i^2} + \frac{1}{\phi_2}, \quad \frac{\phi_{1;i}}{\phi_{2;i}} = \frac{s_i}{\sigma_i^2} + \frac{\phi_1}{\phi_2}, \tag{106}$$

where the derivation of Eq. (106) is similar to that of Eq. (37).

For each experiment i, the conditional posterior precision $1/\phi_{2;i}$ is the sum of the experiment precision $1/\sigma_j^2$ and the prior precision $1/\phi_2$, while the conditional posterior mean $\phi_{1;i}$ is the precision-weighted average of the experiment mean s_i and the prior mean ϕ_1.

The marginal hyperposterior $p(\phi|x)$ may be examined directly via sampling as in the binomial–beta model, but the tractability of Gaussians motivates a further factorisation:

$$p(\phi|x) = p(\phi_1|\phi_2, x)p(\phi_2|x). \tag{107}$$

At the same time, we can show that

$$p(\phi|x) \propto p(x|\phi) = \int d\theta\, p(x|\theta, \phi)p(\theta|\phi) = \prod_i^N \mathcal{N}(s_i|\phi_1, \sigma_i^2 + \phi_2), \tag{108}$$

where the integral over each θ_i is just a convolution of Gaussians.

Comparing the ϕ_1-dependence in the above equations shows that $\phi_1|\phi_2, x$ is also normal:

$$p(\phi_1|\phi_2, x) = \mathcal{N}(\phi_1|\nu, \tau), \tag{109}$$

$$\frac{1}{\tau} = \sum_i^N \frac{1}{\sigma_i^2 + \phi_2}, \quad \frac{\nu}{\tau} = \sum_i^N \frac{s_i}{\sigma_i^2 + \phi_2}. \tag{110}$$

Finally, we have

$$p(\phi_2|x) = \frac{p(\phi|x)}{p(\phi_1|\phi_2, x)} \propto \tau^{1/2} \prod_i^N (\sigma_i^2 + \phi_2)^{-1/2} \exp\left(-\frac{1}{2}\frac{(s_i - \nu)^2}{\sigma_i^2 + \phi_2}\right). \tag{111}$$

To summarise, the full factorisation of the joint posterior is

$$p(\theta, \phi|x) = p(\theta|\phi, x)p(\phi_1|\phi_2, x)p(\phi_2|x), \tag{112}$$

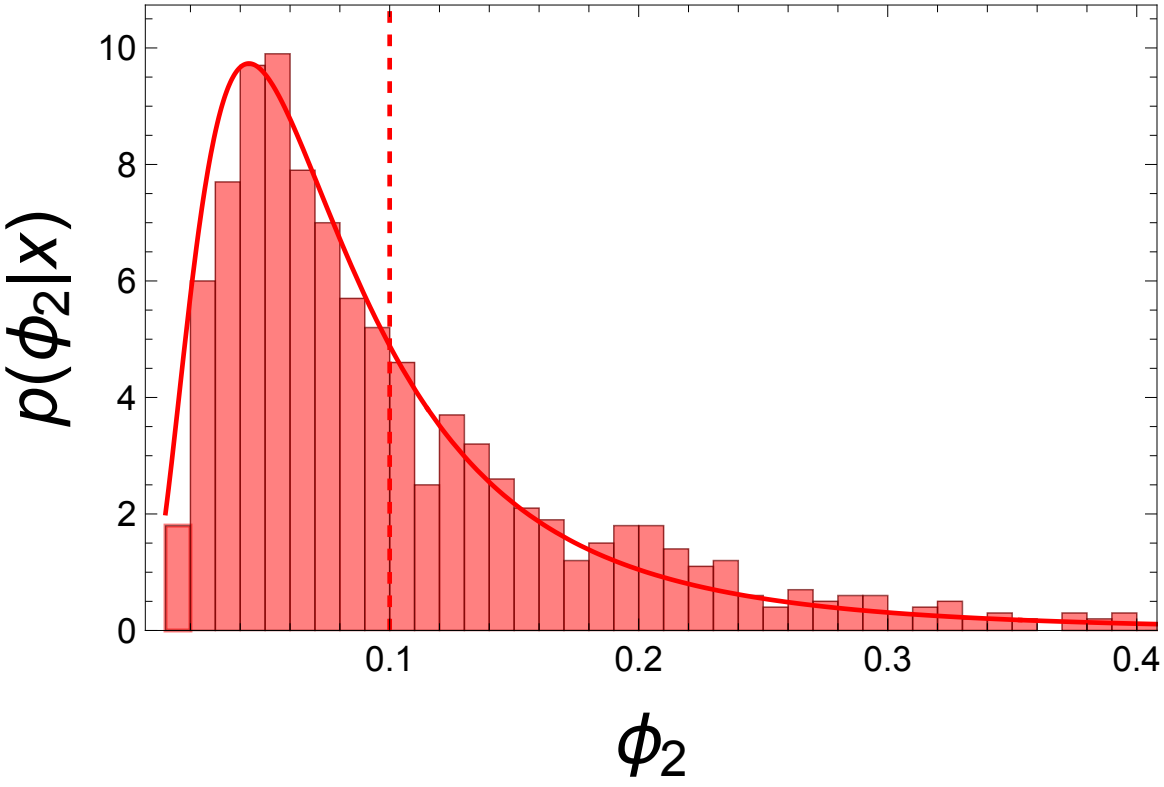
where $\theta|\phi, x$ and $\phi_1|\phi_2, x$ are normally distributed. Only $\phi_2|x$ has an analytically nontrivial distribution, although this too is relatively easy to sample from since it is one-dimensional.

Credit: Leah Saulnier

Consider now $N = 10$ experiments with $n = (10, 20, \ldots, 100)$, a parameter realisation with $\phi = (1, 0.1)$ (i.e., $\theta_i \sim \mathcal{N}(1, 0.1)$), and a data realisation conditioned on those parameters:

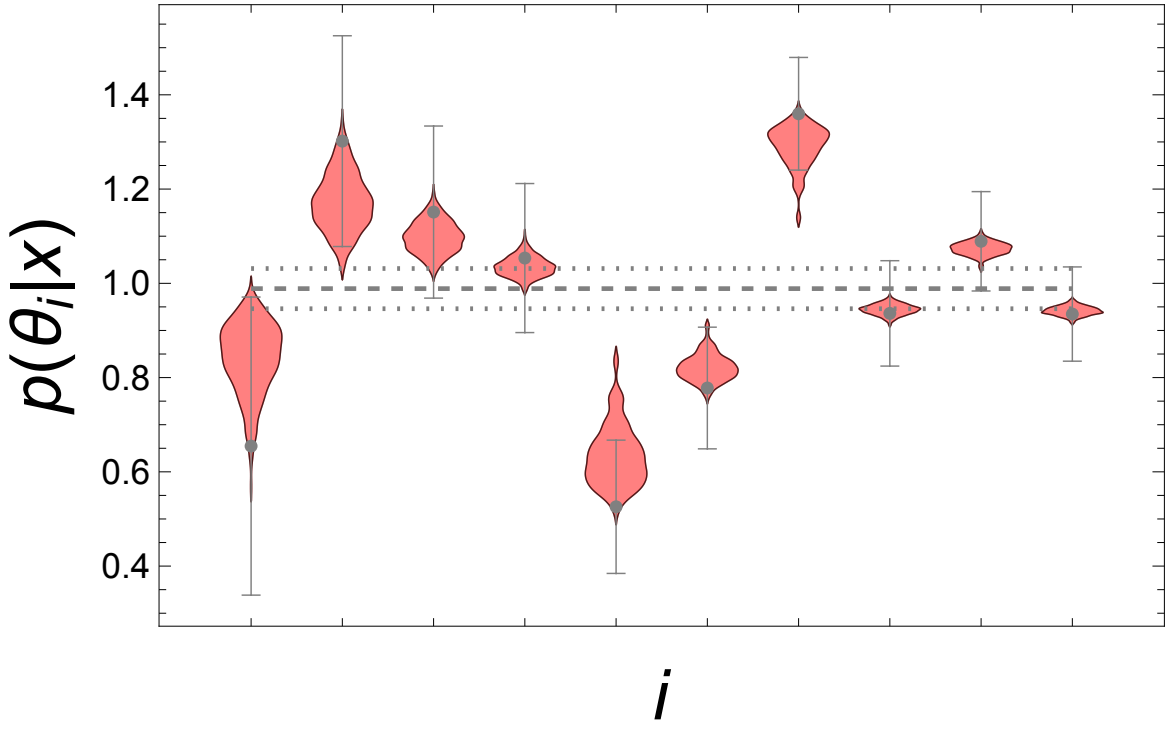
$$x = (0.65, 1.30, 1.15, 1.05, 0.53, 0.78, 1.36, 0.94, 1.09, 0.93). \tag{113}$$

The plot below depicts the marginal hyperposterior $p(\phi_2|x)$ (with constant hyperprior) in two ways: the exact probability density function (red curve), and a histogram of samples that will be used to compute the joint posterior. The dashed red line indicates the true value $\phi_2 = 0.1$.



With samples from $p(\phi_2|x)$ in hand, sampling from $p(\theta|x)$ is straightforward:

1. Draw ϕ_1 from its conditional posterior $p(\phi_1|\phi_2, x) = \mathcal{N}(\phi_1|\nu, \tau)$
2. Draw the θ_i from their conditional posteriors $p(\theta_i|\phi, s_i) = \mathcal{N}(\theta_i|\phi_{1;i}, \phi_{2;i})$
3. Marginalise over ϕ by grouping all of the samples for each θ_i



The grey error bars here are the frequentist 1-sigma confidence intervals $s_i \pm \sigma_i$ for each experiment, while the horizontal grey lines indicate the combined 1-sigma confidence interval.

7. Monte Carlo Sampling

Overview of Sampling

The example models we have studied so far are built from simple data distributions, but already lead to non-standard posterior distributions that must be analysed via sampling. Thus we turn now to the general problem of generating samples from an arbitrarily specified *target distribution*. This problem extends beyond the context of sampling from Bayesian posteriors, so for compactness we will denote target densities by $p(\theta)$ instead of $p(\theta|x)$.

Since the target distribution is specified, we can evaluate the value of $p(\theta)$ for any given θ, or at least the value of an unnormalised density $p'(\theta) = Zp(\theta)$.

In principle, we could do everything we need to do in Bayesian inference with just density evaluations on a Cartesian grid over parameter space.

Visualisation

- This is straightforward up to three dimensions, e.g., density plots over θ_1, density contours/surface over (θ_1, θ_2), density heat maps over $(\theta_1, \theta_2, \theta_3)$
- For higher dimensions, we can only look at marginal densities, e.g., a five-parameter distribution can be summarised by a "triangle" of plots for its two-parameter marginals:

$$p(\theta_1, \theta_2) \propto \sum_{\theta_3} \sum_{\theta_4} \sum_{\theta_5} p(\theta_1, \ldots, \theta_5) \tag{114}$$

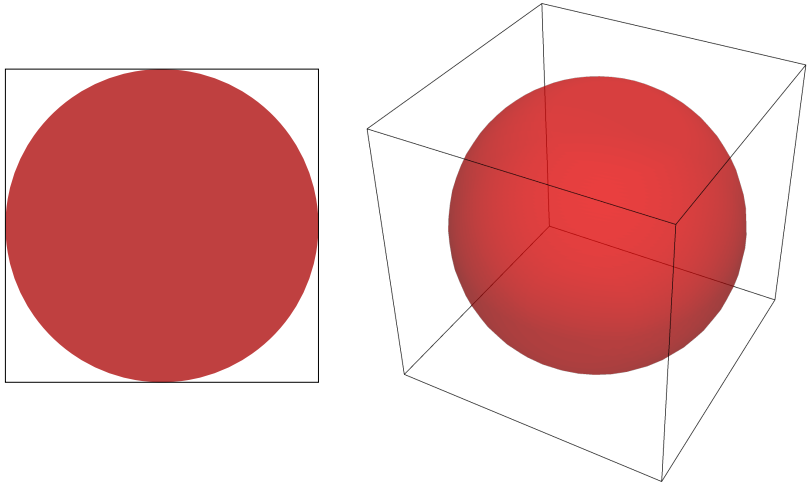
Computation of expectations

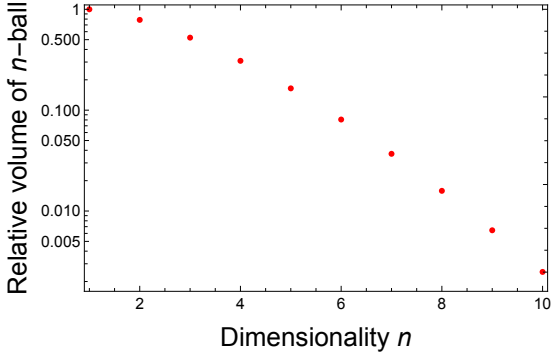
- Proper normalisation is required here, unlike in visualisation
- For a test function τ, we can sum its values over the grid G, weighted by the probability of each point $\theta \in G$, i.e., its value of p (times the volume of space each point represents, but this is equal in a Cartesian grid). Then the normalisation is the sum of weights:

$$\mathrm{E}[\tau] \approx \frac{\sum_{\theta \in G} \tau(\theta) p(\theta)}{\sum_{\theta \in G} p(\theta)} \tag{115}$$

In practice, grid evaluations are clearly awkward for both visualisation and expectation calculations, because the number of required points scales exponentially with dimensionality.

Efficiency is also a problem: As dimensionality increases, high-probability regions are more likely to have a vanishing volume relative to the span of a enclosing Cartesian grid.





The aim of sampling is to obtain a set S of N points that are distributed in proportion to the target density p. Since the samples $\theta \in S$ are by definition concentrated in high-probability regions, visualisation and expectation calculations become more computationally feasible.

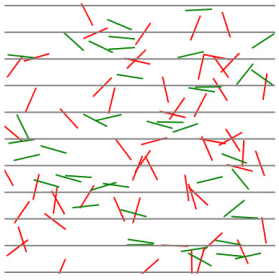
Samples are already weighted by their probability, i.e., their effective normalised weights are all equal to $1/N$. Thus marginalisation and visualisation are straightforward to perform on a histogram approximation of the target distribution, while expectations can be computed as

$$\mathrm{E}[\tau] \approx \frac{1}{N} \sum_{\theta \in S} \tau(\theta). \tag{116}$$

What is a Monte Carlo method?

Broadly, this just refers to a stochastic sampling method: random simulations are repeated many times to obtain a numerical approximation for some quantity of interest.

- Named after the Monte Carlo casino in Monaco; term was coined in the 1940s to refer to methods developed by John von Neumann and Stanislaw Ulam
- First adopted in physics/chemistry, then statistics with the advent of personal computers
- Spirit is much older, e.g., Buffon's needle: $P = 2l/(\pi w)$ for $l < w$



Monte Carlo standard errors

In comparison to deterministic methods, Monte Carlo sampling allows a much larger class of problems to be solved, but also introduces additional error in calculations due to randomness.

- Grid evaluations can be viewed as deterministic sampling, since the probability-weighted grid points are effectively "samples". When feasible, it generally leads to lower error.
- Similarly, if it is possible to add determinism to Monte Carlo methods by obtaining samples in a less random way, this can also reduce error and accelerate convergence.

In frequentist statistics, the standard error refers to the standard deviation of the sampling distribution of a statistic. By analogy, the Monte Carlo standard error (MCSE) refers to this quantity when the "statistic" is a function of Monte Carlo samples from a target distribution.

As Monte Carlo samples are not necessarily independent draws from the target distribution, the MCSE is generally best estimated from multiple sets of simulations. However, there are some useful connections to standard results in the case of (possibly approximate) independence:

- For the mean of the target distribution, the MCSE is the standard error of the sample mean — i.e., $\sqrt{\sigma^2/N}$, where σ^2 is the variance of the target distribution
- For probabilities computed under the target distribution, the MCSE is the standard error of the sample proportion in a corresponding binomial experiment — i.e., $\sqrt{P(1-P)/N}$, where P is the computed probability itself

Example

How many samples from the standard normal distribution $\theta \sim \mathcal{N}(0, 1)$ are required to numerically estimate the following quantities?

1. The mean $\mathrm{E}[\theta]$, to within an absolute error of 0.1
2. The probability $P(\theta > 0)$, to within a relative error of 0.1
3. The probability $P(\theta > 2)$, to within a relative error of 0.1

Generally, more samples are required to estimate:

- Probabilities of rare events
- Extreme quantiles
- Summaries of high-dimensional target distributions

Practicalities

- Numerical underflow is a common problem: values are bounded, but can be arbitrarily close to zero and smaller than what can be stored in floating-point representation
  - Especially the case when working with products of densities
  - On 64-bit machines (double-precision floats), smallest number is $\approx 10^{-308} \approx e^{-700}$
- Logarithms can mitigate (but not remove) the problem of underflow. Some tricks:

$$\ln\left(\frac{\exp x_1 \exp x_2}{\exp x_3}\right) = x_1 + x_2 - x_3 \tag{117}$$

$$\ln\left(\exp x_1 + \exp x_2\right) = x_1 + \ln\left(1 + \exp\left(x_2 - x_1\right)\right) \tag{118}$$

Example

Evaluate $\ln\left(\exp\left(-1000\right) + \exp\left(-1001\right)\right)$ by i) approximation; ii) numerical computation.

- Generally no need to report MCSEs, but they should factor into the precision of results, e.g., it makes no sense to report probabilities $< 1\%$ beyond 1 s.f. with $\lesssim 10^4$ samples

Inverse Transform Sampling

This is a direct method that uses density evaluations on a Cartesian grid to generate random samples from the target distribution. It relies on the computation of the cumulative distribution function F for the target distribution, and its inverse (the quantile function Q):

$$F(\theta) := P(\theta' \leq \theta), \quad Q(u) := F^{-1}(u). \tag{119}$$

Inverse transform sampling is very accurate, and straightforward to implement:

Algorithm

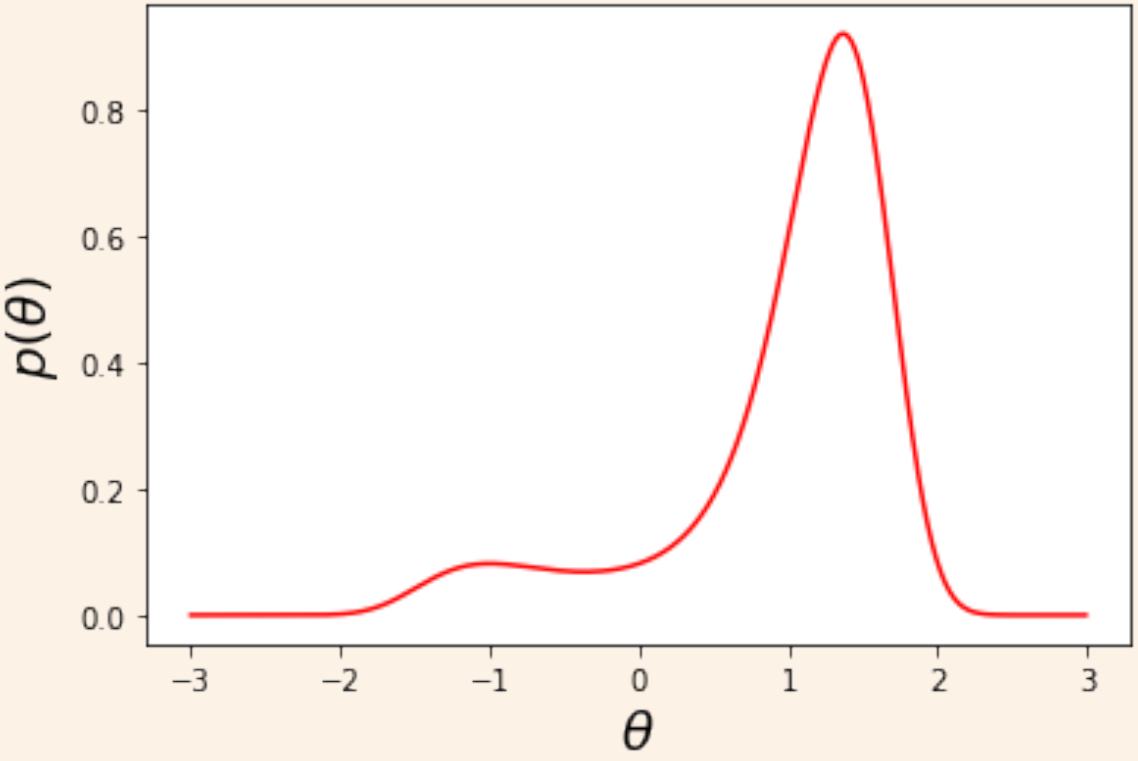
1. Choose a grid with sufficient volume and density
2. Evaluate the target PDF at the grid points and renormalise
3. Compute the discrete CDF and interpolate
4. Draw samples $u_i \sim \mathcal{U}(0, 1)$
5. Transform samples as $\theta_i = Q(u_i)$

The quantile function is not defined for multivariate distributions — it is possible to construct generalisations that "map" $u \in [0, 1]$ to a subspace of parameter space, but there is no unique definition. Thus inverse transform sampling is only applicable to one-parameter distributions.

Example

Jupyter notebook: Use inverse transform sampling to sample from the target density

$$p(\theta) \propto \exp\left(-\frac{1}{2}(\theta^4 - 3\theta^2 - 2\theta + 5)\right). \tag{120}$$



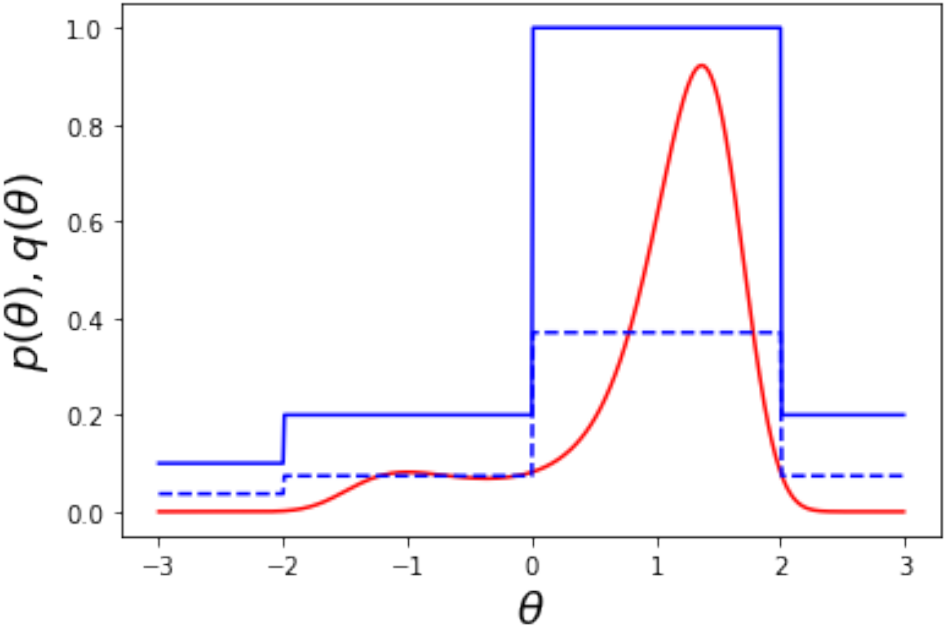
Rejection Sampling

In rejection sampling, we first draw samples from a *proposal distribution* q that is more tractable than the target distribution p, then accept or reject sample θ_i according to an *acceptance rule* that depends on the values $p(\theta_i)$ and $q(\theta_i)$.

Crucially, there must exist some known constant $C > 0$ such that for all θ:

$$Cq(\theta) > p(\theta). \tag{121}$$

Choosing q and determining C appropriately is generally not too difficult, especially in low-dimensional problems. For example, q might be chosen as a diffuse normal distribution for near-Gaussian p, or a piecewise-uniform distribution for more complex p. A suitable value of C may then be determined from inspection (or folded into the definition of q itself).



After q and C are in hand, implementation is trivial:

Algorithm

1. Draw samples $\theta_i \sim q$
2. Draw random variates $u_i \sim \mathcal{U}(0, 1)$
3. Reject all samples where $u_i > p(\theta_i)/(Cq(\theta_i))$

Steps 2 and 3 here are equivalent to "accept θ_i with probability $p(\theta_i)/(Cq(\theta_i))$".

Why does this algorithm work?

- Observe that the points $(\theta_i, Cq(\theta_i)u_i)$ are uniformly distributed in the volume under Cq, while the acceptance rule rejects all such points in the volume above p.
- Thus the remaining points are uniformly distributed in the volume under p, which means their θ_i values are distributed according to p.

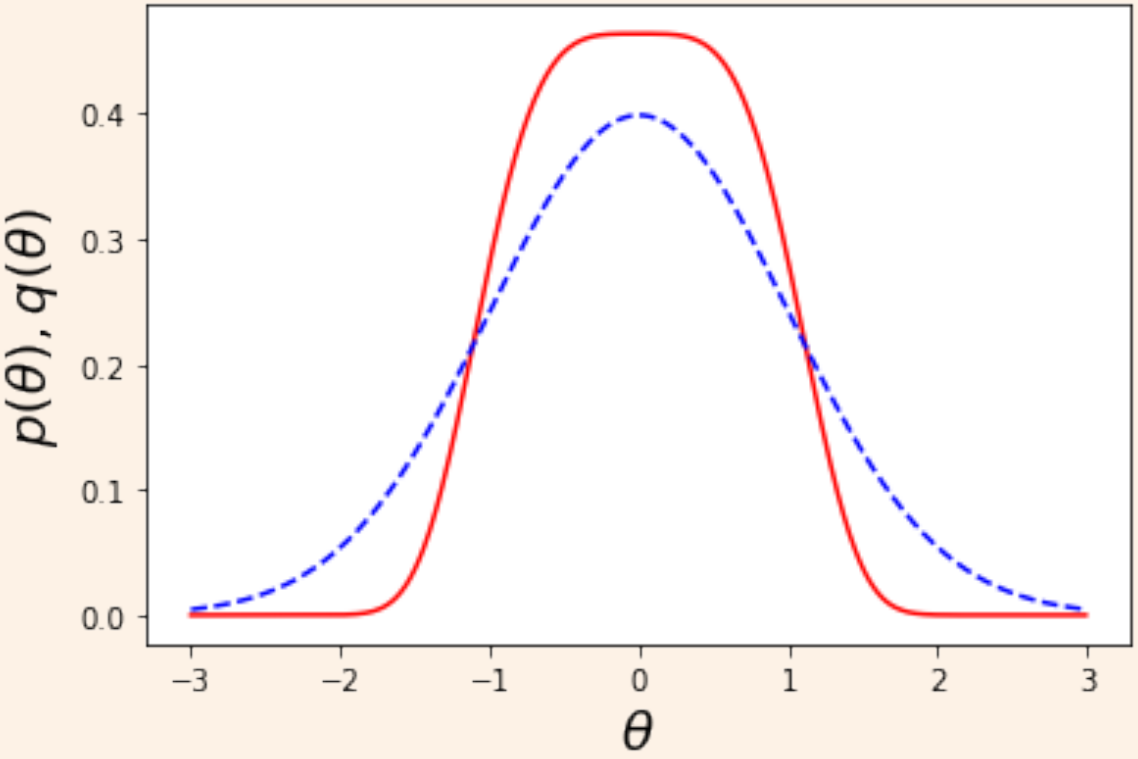
Rejection sampling is generally optimal when $Cq \approx p$ (in the limiting case $Cq = p$, every sample is accepted with a probability of 1 — but then we are already sampling directly from p to begin with). The method is however impractical in more than a few dimensions, because:

- It is difficult to know p everywhere, so choosing $Cq \approx p$ is not feasible.
- For generically chosen Cq, the overall acceptance rate is essentially the ratio of the volumes under p and Cq, and this falls off rapidly with dimensionality (recall the n-ball).

Example

Jupyter notebook: Use rejection sampling with a standard-normal proposal distribution to sample from the target density

$$p(\theta) \propto \exp\left(-\frac{1}{2}\theta^4\right). \tag{122}$$



Importance Sampling

In importance sampling, samples are again drawn from some proposal distribution q as in rejection sampling, but are then weighted appropriately rather than accepted/rejected.

The main utility of importance sampling is in computing expectations. Explicitly denoting the expectation under the target distribution p by E_p, we have

$$\mathrm{E}_p[\tau] = \int d\theta\, \tau(\theta) p(\theta) = \int d\theta\, \tau(\theta) \frac{p(\theta)}{q(\theta)} q(\theta) = \mathrm{E}_q[\tau w], \tag{123}$$

where

$$w(\theta) := \frac{p(\theta)}{q(\theta)}. \tag{124}$$

Note that the function w is normalised under q, i.e., $\mathrm{E}_q[w] = \mathrm{E}_p[1] = 1$.

Since we are computing expectations with a finite set of samples $\theta_i \sim q$, w must be properly renormalised in the discrete approximation to Eq. (123):

$$\mathrm{E}_p[\tau] = \mathrm{E}_q[\tau w] \approx \frac{\sum_i^N \tau(\theta_i) w_i}{\sum_i^N w_i}, \tag{125}$$

where the $w_i := w(\theta_i)$ are known as *importance weights*.

Implementation of the method is again very simple:

Algorithm

1. Draw samples $\theta_i \sim q$
2. Weight samples by $w_i = p(\theta)/q(\theta)$
3. Normalise weights if expectations are required

Example

Jupyter notebook: Use importance sampling with a standard-normal proposal distribution to sample from the target density in Eq. (122). Compute the expectation of θ^2.

Note that when the end goal is not to draw random samples but to compute expectations, the "samples" do not strictly have to come from a proposal distribution and can instead be points on a Cartesian grid. This connects back to deterministic sampling and Eq. (115).

Pros and cons of importance sampling:

- More generalisable and integrable than rejection sampling
- Useful for quickly sampling from similar distributions in model validation/comparison
- Still inefficient in high dimensions unless $q \approx p$ (weights can have very high variance)

Effective sample size

The efficiency of a sampling procedure depends partly on the effective sample size (ESS) for a raw set of N samples. We denote the ESS by N_{eff}.

- In inverse transform sampling, all samples are direct samples from the target distribution, so $N_{\text{eff}} = N$ (but the sampling process itself is not very efficient).
- In rejection sampling, the raw set of samples is from the proposal distribution, but N_{eff} is explicit — it is simply the number of accepted samples.

In importance sampling, N_{eff} should naturally depend on the spread of weights. If a subset of weights are much larger than all of the other weights, then only their corresponding samples are actually providing any significant information about the target distribution.

A quick way of estimating the ESS for importance sampling is

$$N_{\text{eff}} = \frac{1}{\sum_i^N \hat{w}_i^2}, \tag{126}$$

where the $\hat{w}_i$ are the normalised weights $w_i / \sum_i^N w_i$.

Example

Importance sampling with a standard-uniform proposal distribution is used to draw 10^3 samples from the target density

$$p(\theta) \propto \begin{cases} 10^3 & \theta \in [0, 10^{-3}) \\ 10^{-3} & \theta \in [10^{-3}, 1] \end{cases}. \tag{127}$$

What is N_{eff} in the following scenarios?

1. The interval $[0, 10^{-3})$ contains 10 samples.
2. The interval $[0, 10^{-3})$ contains no samples.

Common misconception: A high ESS estimate does not indicate sampling convergence (to the target distribution), while a low ESS does not indicate the lack of convergence.

Importance resampling

It is sometimes more convenient to work with equally weighted samples, i.e., without weights altogether. The simplest way to turn a set of importance samples θ_i into equally weighted samples is *resampling* — to randomly choose (with replacement) samples from $\{\theta_i\}$, but where the probability of choosing each θ_i is proportional to its weight w_i:

Algorithm

1. Draw samples $\theta_i \sim \mathrm{Multin}(1, \{\hat{w}_i\})$

Recall that $\mathrm{Multin}(1, \{\hat{w}_i\})$ is known as the categorical distribution: it describes a single trial with an outcome in $\{\theta_i\}$, with associated probabilities $\{\hat{w}_i\}$. The probability mass function is

$$p(\theta_i) = \prod_j^N \hat{w}_j^{\delta_{ij}} = \hat{w}_i. \tag{128}$$

Practicalities:

- The ESS provides a natural number of equally weighted samples to draw, since resampling any more does not actually provide additional information about the target distribution.
- Resampling with replacement can lead to repeated samples, especially if the weights have high variance. Possible alternatives are to resample without replacement, or to bin the importance samples and draw fresh samples from the resultant histogram distribution.

8. Markov Chain Monte Carlo

Markov Chains

What is a Markov chain?

- A sequence of random variables $\{\theta_i\}$ where the probability distribution of each θ_i, given all previous $\theta_{j<i}$, depends only on θ_{i-1}:

$$p(\theta_i|\theta_{j<i}) = p(\theta_i|\theta_{i-1}) \tag{129}$$

- If the (time) index i is continuous, it is called a Markov process
- Named after Andrey Markov, who studied them in the early 20th century

In the first half of the 20th century, Markov chains/processes were mainly used to describe stochastic processes that are defined on a countable "state space", e.g., a finite set, $\mathbb{Z}$, $\mathbb{Q}$, etc. Now they are also used more generally for measurable state spaces, e.g., $\mathbb{R}$, $\mathbb{C}$, etc.

Examples of Markov chains/processes:

- A random walk
- Brownian motion
- A sequence of independent events (technically)

In the context of Monte Carlo sampling, we are primarily concerned with discrete-time chains (the samples) defined on a general state space (the sample space).

We also focus on chains for which there exists a unique *stationary distribution* p_S, i.e., the distribution of the sequence $\{\theta_i\}$ converges to some distribution in equilibrium.

Existence of a stationary distribution

This is guaranteed by a condition known as *detailed balance*. The process that is described by a Markov chain is fully specified by its *transition probabilities* $p_T(\theta'|\theta)$ for all possible pairs of states θ, θ'. Detailed balance is then the reversibility of transitions — for $\theta, \theta' \sim p_S$, the probability of transitioning to θ' from θ must equal that of transitioning to θ from θ':

$$p_T(\theta'|\theta)p_S(\theta) = p_T(\theta|\theta')p_S(\theta'). \tag{130}$$

In practice, detailed balance can be enforced during the construction of the chain.

Uniqueness of a stationary distribution

This is guaranteed by *ergodicity* — for all pairs of states θ, θ', the probability of eventually (directly or otherwise) transitioning to θ' from θ must be nonzero. In other words, any state can be reached from any other state in finite time. Ergodicity is harder to prove rigorously, but in practice: i) it is often obvious if a chain is non-ergodic; ii) it is less important.

The central idea of Markov chain Monte Carlo (MCMC) sampling methods is to construct a Markov chain whose stationary distribution is the target distribution: $p_S = p$.

The Markov property defined by Eq. (129) is not actually crucial, but only used to prove (via detailed balance and ergodicity) that MCMC methods do converge to the target distribution.

What is the main motivation for MCMC? A quick recap of the methods we've seen so far:

- Fully analytical computation
- Conditional–marginal factorisation
- Grid density evaluations
- Inverse transform sampling
- Rejection sampling
- Importance sampling

There is clearly a need for algorithms that can sample efficiently from general high-dimensional distributions. MCMC is not a perfect solution to the problem of high-dimensional sampling, but it is the most powerful and versatile class of methods by far.

Metropolis–Hastings

While simple, the Metropolis–Hastings algorithm is arguably the most important MCMC method, as it provides the basis and framework for many more advanced sampling algorithms.

- Named after Nicholas Metropolis, who was first author on the 1953 paper "Equation of State Calculations by Fast Computing Machines" in the Journal of Chemical Physics
- Also named after Wilfred Keith Hastings, who generalised the algorithm in 1970

Like rejection and importance sampling, the Metropolis–Hastings algorithm also uses proposal distributions q, but these differ depending on the current state of the chain. We denote the proposal distribution for state θ by $q(\cdot|\theta)$, and refer to $q(\cdot|\cdot)$ as the *proposal kernel*.

Algorithm

1. Choose a starting sample θ_0
2. At iteration i, draw a proposed sample $\theta' \sim q(\cdot|\theta_{i-1})$
3. Calculate the acceptance ratio

$$r(\theta'|\theta_{i-1}) = \frac{p(\theta')q(\theta_{i-1}|\theta')}{p(\theta_{i-1})q(\theta'|\theta_{i-1})} \tag{131}$$

4. Set $\theta_i = \theta'$ with probability $\min\{1, r\}$; otherwise set $\theta_i = \theta_{i-1}$

The original method in the Metropolis et al. paper is a special case of the above algorithm, where the proposal kernel is symmetric: $q(\theta'|\theta) = q(\theta|\theta')$ for all θ, θ'.

By construction, the sequence $\{\theta_i\}$ obtained from the Metropolis–Hastings algorithm is clearly a Markov chain: the procedure of obtaining θ_i depends only on θ_{i-1}.

Exercise

Show that the Metropolis–Hastings chain satisfies detailed balance.

Thus the chain has a stationary distribution, which is precisely $p_S = p$ as desired.

Technically, a proposal kernel that does not depend on any past state of the chain also falls within the Metropolis–Hastings framework: the proposal distribution is then static, and the method reduces essentially to rejection sampling. We will not consider such kernels.

Practicalities

Choosing a proposal kernel:

- Should be tractable to sample from, e.g., a Gaussian kernel
- Should be tunable such that proposed samples are accepted often but also informative
- Should try to satisfy ergodicity (usually obvious)

"Burning in" the chain:

- Early samples might be highly dependent on the starting sample θ_0, which was arbitrary
- The chain needs time to reach equilibrium, so these are not likely to be samples from the stationary distribution, and it makes sense to discard some number of them

Using multiple chains:

- Different starting samples to cover the region of interest in parameter space
- Can split chains and consider each part as a chain (useful for diagnostics)
- Can be very effective when combined with parallel computing

Monitoring quantities of interest separately:

- For diagnostics, we will talk about the "marginal" chain for a single parameter
- The full chain might reach equilibrium at different rates in each parameter
- Expectations of (functions of) parameters that are of interest can also be monitored

Assessing stationarity and mixing

In MCMC sampling, it is important to check whether the samples have reached equilibrium in a single chain (stationarity) or across multiple chains (mixing). A simple diagnostic is the *potential scale reduction factor* (PSRF) for a given chain or set of chains, which estimates how much the variance of the samples might be reduced if the chain(s) were run to infinity.

Given a set of equal-length chains (after burn-in), let us split each chain into two equal-length parts. We then have a set of m partial chains (indexed by i), each with n samples (indexed by j). For brevity, we denote the mean and variance of $\{\theta_{ij}\}$ over some index k by sm_k and sv_k respectively. The variances within (W) and between (B) the partial chains are estimated as

$$W := \mathrm{sm}_i \left[\frac{n}{n-1} \mathrm{sv}_j[\theta_{ij}] \right], \quad B := \frac{nm}{m-1} \mathrm{sv}_i[\mathrm{sm}_j[\theta_{ij}]]. \tag{132}$$

The overall variance of $\{\theta_{ij}\}$ is defined as a weighted average of W and B:

$$V := \frac{n-1}{n} W + \frac{1}{n} B, \tag{133}$$

which is an overestimate of the target variance $\mathrm{Var}[\theta]$ if the chains are not stationary/mixed. In that scenario, W is an underestimate of $\mathrm{Var}[\theta]$, with $V \to W \to \mathrm{Var}[\theta]$ as $n \to \infty$.

Finally, the PSRF is defined as $\sqrt{V/W}$, which approaches a value of 1 as $n \to \infty$.

Assessing efficiency

MCMC samples that are close to one another in a chain will generally not be independent, due to the Markov property. Thus it also makes sense to define an ESS for the chain. This is done by treating the chain as stationary and computing its autocorrelation (the correlation of the chain with a lagged version of itself, as a function of lag time). The autocorrelation is then integrated over all lag times to estimate the average separation between independent samples.

First, the (normalised) autocorrelation function of a stationary process $\theta(t)$ is

$$\rho(\tau) = \frac{\mathrm{Cov}[\theta(t), \theta(t+\tau)]}{\mathrm{Var}[\theta(t)]}. \tag{134}$$

There are several different ways to estimate $\rho_j \equiv \rho(\tau_j)$ from a finite chain $\{\theta_i \equiv \theta(t_i)\}$ of length N, but it is fairly standard functionality in most numerical packages. The ESS is then $N_{\mathrm{eff}} = N/\tau_\theta$, where τ_θ is the integrated autocorrelation time:

$$\tau_\theta = \int_{-\infty}^{\infty} d\tau\, \rho(\tau) \approx \sum_{j=-N}^{N} \rho_j = 1 + 2\sum_{j=1}^{N} \rho_j. \tag{135}$$

- This discussion is for a single chain; for multiple chains, we might just sum their ESSs
- In most applications, we seek to reach at least 10 effective samples per chain
- We might take every τ_θ-th sample ("thin" the chain) to simulate independent draws

Note that the sampling efficiency is minimal ($N_{\text{eff}}/N = 1/\tau_\theta \ll 1$) for the two extreme scenarios in the Metropolis–Hastings algorithm:

- If the proposed samples are too close to the current state, the acceptance ratio is $r \approx 1$. But the chain only moves within a small region, and is effectively constant.
- If the proposed samples are too far from the current state, the acceptance ratio is $r \approx 0$ (especially if the current state already has a high probability). Thus the chain seldom moves, and is again effectively constant.

In most applications, the maximal value for the efficiency is actually < 1, i.e., $\tau_\theta = 1$ generally cannot be achieved for arbitrary target and proposal distributions.

The maximal efficiency can be translated in terms of the *acceptance rate* — the proportion of times θ' is accepted in a chain. Consider a d-dimensional standard multinormal target distribution $\mathcal{N}(0, I_d)$, and Metropolis–Hastings with a class of proposal kernels $\mathcal{N}(\cdot|\cdot, CI_d)$:

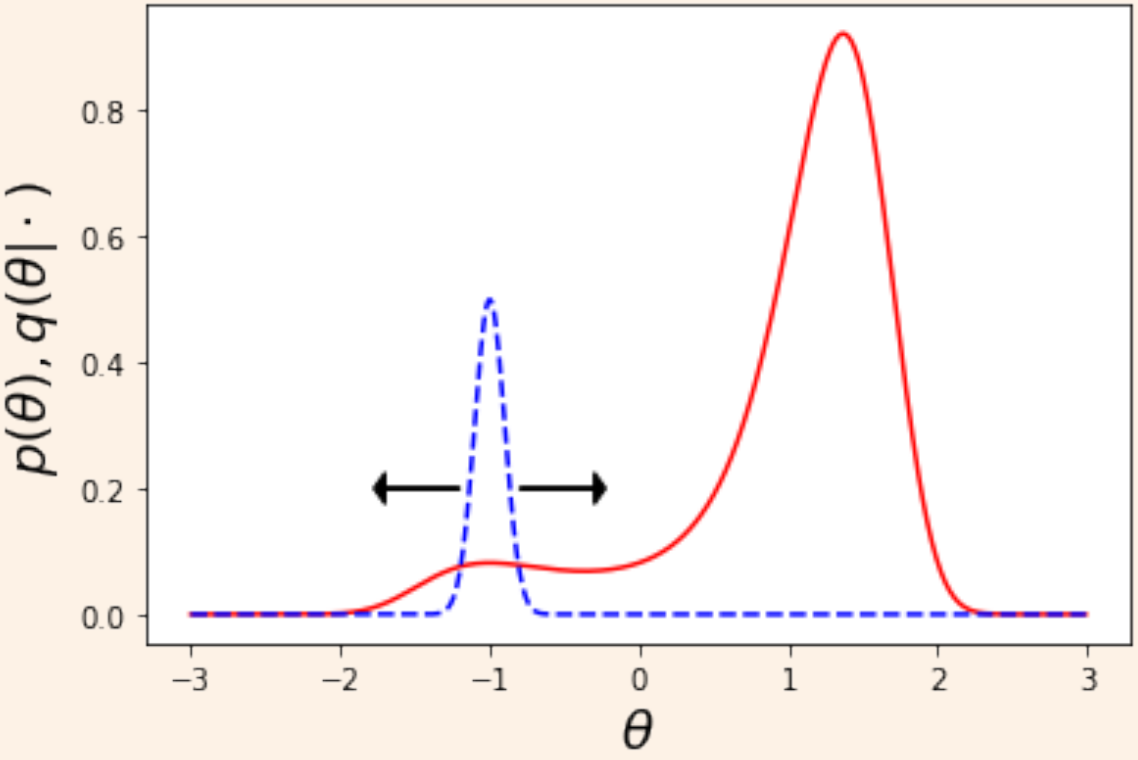
- The most efficient value of C turns out to be $\approx 5.76/d$, and it corresponds to an acceptance rate between ≈ 0.44 ($d = 1$) and ≈ 0.23 ($d > 5$).
- Since most target distributions are near-Gaussian, and Gaussian proposal kernels are commonly used, this is a useful heuristic for tuning Metropolis–Hastings samplers.

To conclude our discussion of practicalities for general MCMC sampling, a word of caution:

After all this work to assess stationarity, mixing and efficiency — we still cannot guarantee convergence to the target distribution! All we have done is to check for non-convergence.

Example

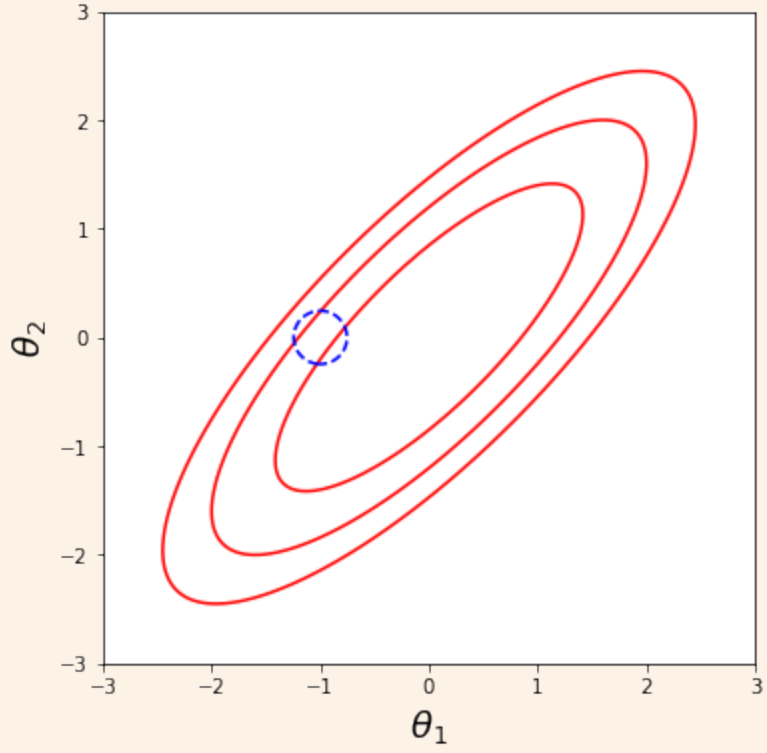
Jupyter notebook: Use the Metropolis–Hastings algorithm with a tunable Gaussian proposal kernel $\mathcal{N}(\cdot|\cdot, C)$ to sample from the target density in Eq. (120).



Example

Jupyter notebook: Use the Metropolis–Hastings algorithm with a tunable Gaussian proposal kernel $\mathcal{N}(\cdot|\cdot, \Sigma')$ to sample from the target density

$$p(\theta) = \mathcal{N}(\theta|0, \Sigma), \quad \Sigma = \begin{pmatrix} 1 & 0.8 \\ 0.8 & 1 \end{pmatrix}. \tag{136}$$



Gibbs Sampling

Gibbs sampling is a general algorithm for multivariate target distributions ($d > 1$). It involves sampling sequentially from the one-dimensional conditional distributions for each parameter.

- Named after Josiah Willard Gibbs for connections to methods in statistical mechanics
- Actually formalised and popularised by Stuart Geman and Donald Geman in a 1984 paper

The base algorithm is very simple:

Algorithm

1. Choose a starting sample θ_0
2. At iteration i, update each component j of θ_i sequentially, by sampling from

$$\theta_{i;j} \sim p(\cdot|\theta_{i;1}, \ldots, \theta_{i;j-1}, \theta_{i-1;j+1}, \ldots, \theta_{i-1;d}) \tag{137}$$

Note that the d conditional distributions in Eq. (137) are generally not standard distributions, but they are one-dimensional and thus still relatively straightforward to sample from.

It is clear that the above algorithm gives rise to a Markov chain. Indeed, Gibbs sampling can be viewed as a special case of Metropolis–Hastings: with a proposal kernel that is built from the conditionals of the target distribution, resulting in an acceptance probability of 1.

Exercise

Write down the conditional densities for the general binormal target density

$$p(\theta) = \mathcal{N}(\theta|\mu, \Sigma), \quad \Sigma = C \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}, \quad \rho \in (-1, 1). \tag{138}$$

Example

Jupyter notebook: Use Gibbs sampling to sample from the target density in Eq. (136).

Blocked Gibbs sampling

Since the Metropolis–Hastings algorithm can itself be used to sample from the conditional distributions in Gibbs sampling, the latter might also be viewed as an extension of the former.

Such a view leads naturally to a generalisation known as blocked Gibbs sampling, where each component j of θ_i is not a single parameter but a set of parameters (called a block).

- The conditional distributions in Eq. (137) are then potentially complex multivariate distributions, which can be sampled with general algorithms such as Metropolis–Hastings.
- Rather than moving completely randomly in the full parameter space, the chain moves sequentially through a sequence of subspaces, and randomly within each subspace.

Advantages and applications:

- If parameters that are more strongly correlated are assigned to the same blocks, this can greatly improve the chain's convergence to the target distribution
- In Bayesian inference, if the data x is described by a combination of multiple statistical models $\mathcal{M}_m \equiv p_m(x|\theta_m)$, the joint posterior $p(\theta|x)$ might be sampled with the blocks θ_m
- In hierarchical Bayesian inference, the joint (parameter–hyperparameter) posterior $p(\theta, \phi|x)$ might be sampled with the blocks θ and ϕ

9. Advanced Sampling

Slice Sampling

All of the methods we have seen so far (even the tunable ones) have a fixed algorithm that does not evolve between iterations. We now examine slice sampling as an example of an algorithm that incorporates adaptive (self-tuning) aspects.

- Recall from our discussion of rejection sampling that if the target density $p(\theta)$ is considered as a function over parameter space, then sampling uniformly from the volume under $p(\theta)$ is equivalent to sampling from p.
- More formally:

$$p(\theta, u) \propto \mathbf{1}_{[0,p(\theta)]}(u). \tag{139}$$

- In its most general sense, slice sampling is a framework for adaptive methods that sample from the $(d+1)$-dimensional distribution $p(\theta, u)$.
- We focus on the $d = 1$ case, which serves as a building block for more complex methods.

Algorithm

1. Choose a starting sample θ_0
2. At iteration i, draw a sample $u' \sim \mathcal{U}(0, p(\theta_{i-1}))$
3. Draw a sample $\theta_i \sim \mathcal{U}(\Theta)$, where $\Theta = \{\theta' : p(\theta') > u'\}$

If executed perfectly, the above algorithm gives exact samples from p. Note that it still describes a Markov chain, so the samples are correlated.

Step 3 is clearly the nontrivial step, and for target distributions with multiple modes, the set Θ is actually a union of disjoint intervals. A practical approximation to Step 3 is needed. There are several different methods, but we only describe one here:

Algorithm

I Choose an interval $[\theta_l, \theta_r]$ containing θ_{i-1}, by stepping out from θ_{i-1} in steps of size s until $p(\theta_l), p(\theta_r) < u'$

II Draw a proposed sample $\theta' \sim \mathcal{U}(\theta_l, \theta_r)$

III If $p(\theta') < u'$, shrink the interval by setting $\theta_l = \theta'$ or $\theta_r = \theta'$, then repeat Step II

The step size s used in this procedure is not that crucial to the accuracy or efficiency of the algorithm, as the interval $[\theta_l, \theta_r]$ is defined adaptively at an additional cost that is:

- $\mathcal{O}(c/s)$ if s is smaller than the characteristic scale c of the target distribution (as opposed to $\mathcal{O}((c/s)^2)$ in Metropolis–Hastings with a fixed step)
- $\mathcal{O}(\ln(s/c))$ if s is larger than c (as opposed to $\mathcal{O}(e^{s/c})$ (!) in Metropolis–Hastings)

Example

Jupyter notebook: Use slice sampling to sample from the target density in Eq. (120).

Applications:

- Can be used to sample from one-dimensional conditional distributions in Gibbs sampling.
- One-dimensional slice sampling does not have to be performed on conditional distributions (i.e., θ is a single parameter), but can also be applied to arbitrary combinations of parameters. Thus it can be used in more complex methods as a way of sampling along desired slices of the full parameter space.
- Can be generalised to multivariate target distributions by using hyperrectangles.

Hamiltonian Monte Carlo

Recall that deterministic sampling is generally more accurate than Monte Carlo sampling since it reduces MCSEs. Thus adding deterministic aspects to MCMC methods can accelerate convergence. This is the main idea behind the Hamiltonian Monte Carlo (HMC) algorithm, which is inspired by Hamiltonian mechanical systems in physics.

Hamiltonian mechanics

In classical mechanics, a Hamiltonian system that is described by configuration variables q and conjugate momenta p obeys Hamilton's equations (a system of coupled first-order ODEs):

$$\dot{q} = \partial_p H(q,p), \quad \dot{p} = -\partial_q H(q,p). \tag{140}$$

A natural Hamiltonian system has $H(q,p) = T(q,p) + V(q)$, where T and V are the kinetic and potential energies of the system respectively.

For one particle of mass m with one d.o.f., $T = p^2/(2m)$ and we have

$$v = \dot{q} = \partial_p T(p) = \frac{p}{m}, \quad ma = \dot{p} = -\partial_q V(q) = F. \tag{141}$$

More generally, we can write $T = p^T M(q)^{-1} p/2$, where M is called the mass matrix. Then

$$\dot{q} = M(q)^{-1} p, \quad \dot{p} = -\partial_q V(q). \tag{142}$$

The HMC algorithm

In HMC, we view a chain as a particle moving in state space, and allow its movement to be partially guided by Hamiltonian mechanics rather than a completely random walk.

- Proposed by Duane et al. in 1987 for calculations in lattice QCD
- Sometimes called hybrid Monte Carlo for historical reasons (it is a hybrid of Monte Carlo and deterministic sampling)

To do this, we need to relate the potential under which the chain is moving to the target distribution. But first, a change to the physics notation is prudent: $(q, p) \equiv (\theta, \phi)$. We also require M to be constant, i.e., not dependent on q.

Treat (θ, ϕ) as random variables, and consider the joint density

$$p(\theta, \phi) \propto \exp\left(-H(\theta, \phi)\right) = \exp\left(-V(\theta)\right) \exp\left(-\frac{1}{2}\phi^T M^{-1} \phi\right), \tag{143}$$

such that the marginal density of θ is simply $\propto \exp^{(-V(\theta))}$. The marginal is then defined to be the target density by setting

$$V(\theta) := -\ln\left(p(\theta)\right), \tag{144}$$

which allows p to be sampled from by marginalising over the joint distribution of (θ, ϕ).

We cannot just evolve (θ, ϕ) with Hamilton's equations, because the Hamiltonian is a conserved quantity!

Example

If the target distribution is the standard normal distribution, how does any chain move under Hamilton's equations alone?

The resultant "samples" are neither random nor distributed according to p. But consider the motion of the chain at different points in the state space — how is this beneficial for sampling?

HMC works because it adds randomness to the deterministic motion in three ways:

- The momenta ϕ are drawn randomly from their marginal distribution $\mathcal{N}(0, M)$ at each iteration. This is the main source of randomness (changes the Hamiltonian).
- Hamilton's equations are integrated numerically with a finite step size, so there is numerical error and the Hamiltonian is not conserved over the integration either.
- The chain is only evolved for some time under Hamilton's equations, after which the end point is accepted or rejected according to some rule (typically Metropolis–Hastings).

In other words, HMC is essentially MCMC with a partially deterministic proposal kernel.

Algorithm

1. Choose a starting sample θ_0
2. At iteration i, draw a sample $\phi_{i-1} \sim \mathcal{N}(0, M)$
3. Compute a proposed sample (θ', ϕ') by evolving $(\theta_{i-1}, \phi_{i-1})$ for L steps of size ϵ with a leapfrog (or some other symplectic) integrator
4. Calculate the acceptance ratio

$$r(\theta'|\theta_{i-1}) = \frac{p(\theta', \phi')}{p(\theta_{i-1}, \phi_{i-1})} \tag{145}$$

5. Set $\theta_i = \theta'$ with probability $\min\{1, r\}$; otherwise set $\theta_i = \theta_{i-1}$

Note that the acceptance ratio here is simply the ratio of the (joint) target density at the proposed and current states, as opposed to Eq. (131). This is because the "proposal distribution" here is deterministic and reversible, thus symmetric.

Step 3 above is the deterministic component of the HMC algorithm, and usually employs a simple leapfrog integrator:

Algorithm

I At each integration step, update $\phi \to \phi + (\epsilon/2)\partial_\theta \ln p(\theta)$ (half-step in ϕ)

II Update $\theta \to \theta + \epsilon M^{-1}\phi$ (full step in θ)

III Update $\phi \to \phi + (\epsilon/2)\partial_\theta \ln p(\theta)$ (half-step in ϕ)

For all integration steps, Step III above can be performed together with Step I for the next integration step.

HMC has several nice properties:

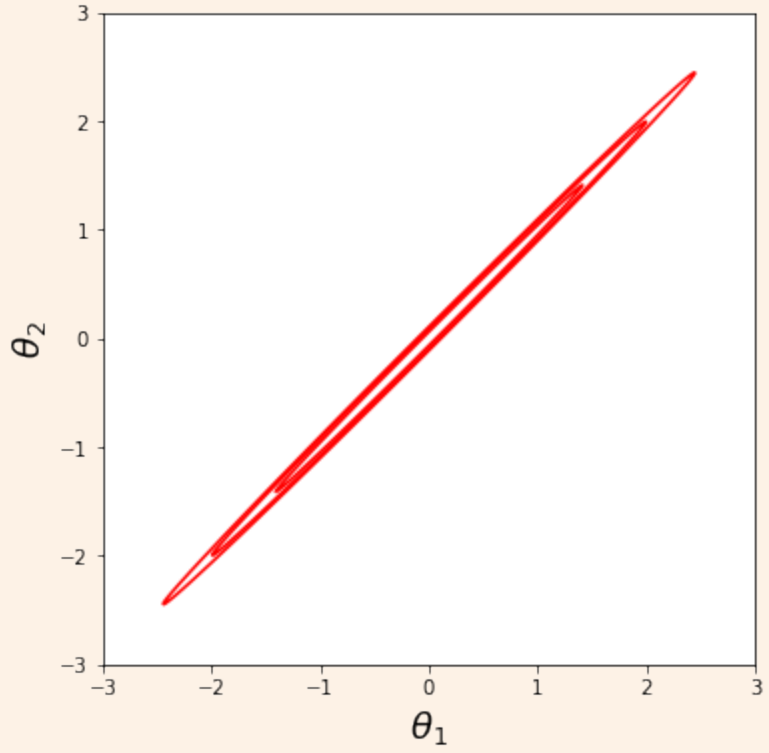
- Leapfrog integration error does not grow in time*, but oscillates around exact trajectory
- The overall algorithm satisfies detailed balance

*Only if ϵ is small enough

Example

Jupyter notebook: Use HMC to sample from the target density

$$p(\theta) = \mathcal{N}(\theta|0, \Sigma), \quad \Sigma = \begin{pmatrix} 1 & 0.999 \\ 0.999 & 1 \end{pmatrix}. \tag{146}$$



Practicalities

Tuning parameters: Mass matrix, step size and number of leapfrog steps

- The mass matrix is essentially the scaling of each parameter, and its tuning can be informed by prior knowledge, or the covariance of burn-in samples. It can be chosen such that the transformed parameters have characteristic scales ~ 1 and correlations ≈ 0.
- If this is fulfilled, it is appropriate to choose an integration length (step size times number of steps) that is also ~ 1.
- An efficiency heuristic (derived similarly to that for Metropolis–Hastings) yields an optimal acceptance rate of around 0.65.

Dynamic HMC

- Various extensions to adjust number of steps adaptively
- Basic idea is no-U-turn sampling (NUTS): Integrate until U-turn is detected, then no point in going much further because points are on a closed trajectory
- To preserve detailed balance in NUTS, need to integrate in two directions and select random point along the trajectory

Possible problems

- May break for distributions where log-density has a lot of structure, e.g., changes on very different scales, or multiple modes
- Step sizes must be fine-tuned, and large step sizes can lead to divergence
- Also affects dynamic HMC; generally the case for gradient-based methods

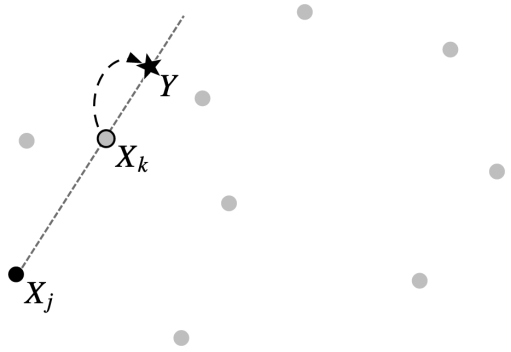
Other Common Algorithms

Adaptive proposals

Another general way to add adaptiveness to the Metropolis–Hastings algorithm is to update the proposal based on the distribution of previous samples, rather than the local geometry of the problem as in slice sampling or dynamic HMC.

- For a single chain or ensemble of chains, the proposal can be periodically updated to match the scale (covariance matrix) of all the samples up to that point. This can be done with a Gaussian kernel, or a more general kernel density estimate. It works because the chain is closer to convergence at each step (whether or not the proposal was altered).
- In the ensemble case, another strategy is to incorporate a partially deterministic proposal for each chain that is informed by the current locations of the other chains (so that their overall movements are correlated). Example proposals: leapfrog; stretch; walk (like scale-based, but using the locations of other chains); replace (like walk, but not centered on the chain that is being updated).

When using adaptive proposals, it is important to prove or at least check for convergence. For example, the leapfrog/stretch/walk/replace proposals can be shown to satisfy detailed balance.



Credit: Goodman & Weare, 2010

Multimodal difficult

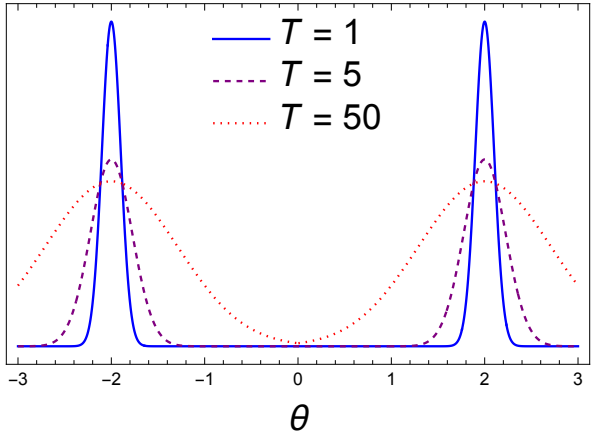
Simulated annealing and parallel tempering

Multimodal distributions are generally problematic for all sampling algorithms. In the case of MCMC, it can be difficult for the chains to move from one mode to another. One way to mitigate this is to sample from variants of the target distribution that have the same profile, but which are easier to explore on different scales:

$$p_i(\theta) \propto p(\theta)^{1/T_i}, \tag{147}$$

where the $T_i > 1$ are temperatures (by analogy to the Boltzmann distribution), and $\{T_i\}$ is called a temperature ladder.

- The temperature ladder can be tuned to the specific scales in the target distribution
- In simulated annealing, sampling is performed at an initial high temperature, then gradually decayed to $T = 1$ based on some schedule
- In parallel tempering, sampling is performed in parallel for all of the p_i, and chains with different temperatures are occasionally swapped



Transdimensional sampling

In transdimensional sampling, the dimensionality of parameter space is allowed to change between iterations. Example applications include: averaging inference results over multiple models, or analysing data containing an unknown number of realisations from a single model.

- Consider a set of models (indexed by m) and their corresponding parameters θ_m. In the general product-space formulation, the target distribution is defined on the direct product of the index set and the combined parameter space: $p(m, \{\theta_m\})$.
- Assume θ_m and θ'_m are independent for all m, m'. In the context of inference, the aim is typically to obtain the marginal model posterior $p(m|x)$ and conditional parameter posterior $p(\theta_m|m, x)$ by sampling from $p(m, \{\theta_m\}|x)$.
- To sample from $p(m, \{\theta_m\}|x)$ using MCMC, one method of ensuring detailed balance is to match the dimensionality of parameter spaces in any proposal $(m, \theta_m) \to (m', \theta_{m'})$, by adding auxiliary variables. This is known as reversible-jump MCMC.
- More generally, the models can have shared or dependent parameters, which requires some extra bookkeeping.
- The set of indices does not have to be finite, as long as the target distribution is proper. For example, model m might describe the existence of m realisations from the same model in some aggregate data.

Evolutionary/genetic algorithms

This class of algorithms originally arose in the context of search and optimisation, but can be adapted for sampling as well. They involve a large ensemble of chains that evolve based on the target density at their current locations.

- Chains can live, die, split, combine, or mutate
- Specific examples: differential evolution, particle filtering

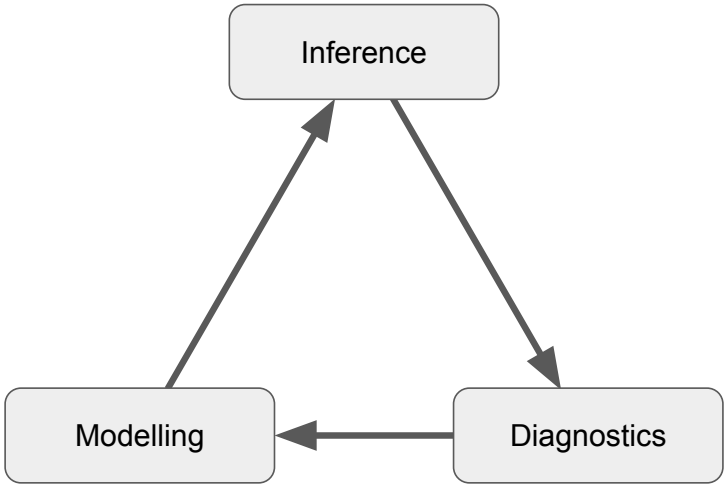
Nested sampling

Nested sampling is a general framework for computing evidences in Bayesian inference (effectively, the normalisation constant for a general target distribution). It generates samples from the distribution as a by-product.

- A large set of "live points" are evolved by replacing at each iteration the lowest-probability point, with one that is uniformly drawn from the region of points with a higher probability.
- The "dead points" that are replaced along the way are effectively weighted samples from the target distribution.
- The freedom (and difficulty) is in the uniform sampling step, which can use any method. If that is done properly, nested sampling is accurate, efficient and requires minimal tuning.

10. Model Validation and Comparison

Motivation



Model validation and comparison comprise the diagnostics step of applied Bayesian statistics. Its overall aim is to construct models that are useful and lead to sensible results.

Validation

There are three broad ways of validating the performance of a Bayesian model:

- Through common sense or intuition, by examining inferred and predicted quantities. This can and should be done during model construction as well, by examining draws from the prior predictive distribution.
- Through external validation, by obtaining and comparing to additional data (essentially the scientific method).
- Through internal validation. This will be our main focus here.

In order of increasing specificity, internal validation can be done by:

- Conducting sensitivity analyses
- Performing posterior predictive checks: test statistics, p-values
- Estimating posterior predictive accuracy: information criteria, cross-validation

Comparison

Model comparison is closely related to validation, and the conceptual distinction between the two can sometimes be blurry:

- If two models are validated in the same way, they can be compared as a by-product (especially with meaningful, quantitative validation).
- At the same time, if a model is adjusted and compared to its old version, that can also be viewed as validation.
- Comparison does not need a measure of absolute performance, only relative performance. The former is harder to define, so comparison is often more clear-cut than validation.

We may compare models by:

- Comparing posterior predictive accuracy
- Computing probability odds: Bayes factors, posterior odds

Sensitivity Analyses

In realistic problems:

- A model might fit (describe) the observed data well, but not be very robust to any potential outliers in the data
- Multiple models might fit the data similarly well, but at the same time lead to significantly different inferences or predictions
- Fitting the data is not everything, e.g., good likelihood + bad prior might give less sensible results than average likelihood + average prior

Thus one should also analyse the sensitivity (lack of robustness) of results to the model being used. This is typically done by testing different likelihoods and priors, and/or expanding the model by switching to a hierarchical approach.

General guidelines:

- Hierarchical models $\theta_i \sim \pi(\theta|\phi)$ are the middle-ground between separate models $\theta_i \sim \pi_i(\theta_i)$ and pooled models $\theta_i = \theta_j \sim \pi(\theta)$. Thus they are generally more robust than the former, and can provide a better fit than the latter.
- Inference of extreme probabilities/quantiles is generally more sensitive (similar to MCSE).
- Prediction of extrapolated ("out-of-sample") data is generally more sensitive.
- Using proper prior distributions can help to improve robustness.

Posterior Predictive Checks

The general idea of such checks is to generate multiple replicated data sets (i.e., realisations of the posterior predictive distribution) and compare them to the original data.

If the data is partitioned as $\{x_i\}$, the marginal posterior predictive distributions can be examined also. This can help in determining whether or not a hierarchical model is appropriately partitioned.

- In the lab rat example, we might assume that the main underlying effect on survival rate might be due to conditions at two separate labs. But if the main effect is from two different populations of rats being used at both labs, then a marginal posterior predictive check might indicate this.

The discrepancy between the observed data x and replicated data $\tilde{x}$ may be quantified through the definition of a suitable test statistic $\tau(\cdot)$ that can be evaluated for both x and $\tilde{x}$. If $\tau(x)$ is inconsistent with the distribution of $\tau(\tilde{x})$, the model is making poor predictions.

- It is also possible to generalise τ to depend on the parameters θ in some way, which can allow different visualisations by examining $\tau(\cdot, \theta)$ under the posterior $p(\theta|x)$.

Good test statistics should be *ancillary*: they should depend mainly on x or $\tilde{x}$, and be almost independent of θ.

- In the normal model with parameters $\theta \equiv (\mu, \sigma^2)$, examples of ancillary test statistics are the minimum/maximum of the data, or the sample skewness/kurtosis.
- The sample mean/variance are estimators of θ, which were in turn fitted to x. Thus their values for $\tilde{x}$ would naturally be consistent with that for x.

Posterior predictive p-values

In frequentist statistics, the standard p-value for a test statistic τ is (in our notation)

$$\text{pv} := P(\tau(\tilde{x}) \geq \tau(x)|\hat{\theta}), \tag{148}$$

where the probability is under the data distribution $p(\tilde{x}|\hat{\theta})$ with a fixed value of an estimator for θ. (In one dimension, this expression is for a right-sided test; the inequality is flipped for a left-sided test, and for a two-sided test, pv is twice the smaller of the one-sided values.)

The posterior-predictive p-value is defined analogously as

$$\text{ppv} := P(\tau(\tilde{x}) \geq \tau(x)|x), \tag{149}$$

where the probability is under the posterior predictive distribution $p(\tilde{x}|x)$:

$$\text{ppv} = \iint d\tilde{x}\, d\theta\, \mathbf{1}_{\tau(\tilde{x})\geq\tau(x)}(\tilde{x}) p(\tilde{x}|\theta) p(\theta|x). \tag{150}$$

Its interpretation is similar as well: the ppv is the probability of obtaining a test-statistic value at least as extreme as $\tau(x)$, but under the posterior predictive distribution instead of the hypothesis being tested (the estimated data distribution).

Posterior predictive p-values should be used with care:

- The choice of a good test statistic is far more important than the ppv itself.
- It can be useful as a quick check, but is really just a summary of τ itself under $p(\tilde{x}|x)$. Collapsing this information into a single number can be deceptive.
- A similar criticism also holds for frequentist p-values, among other problems.

Posterior predictive hypothesis tests

The p-value is a simple form of hypothesis testing. More detailed tests (defined analogously under the posterior predictive distribution) can likewise be useful — but also should not admit a black-or-white interpretation, as this can actually be counterproductive in model validation.

In such tests, x and $\tilde{x}$ can be compared directly. Some examples:

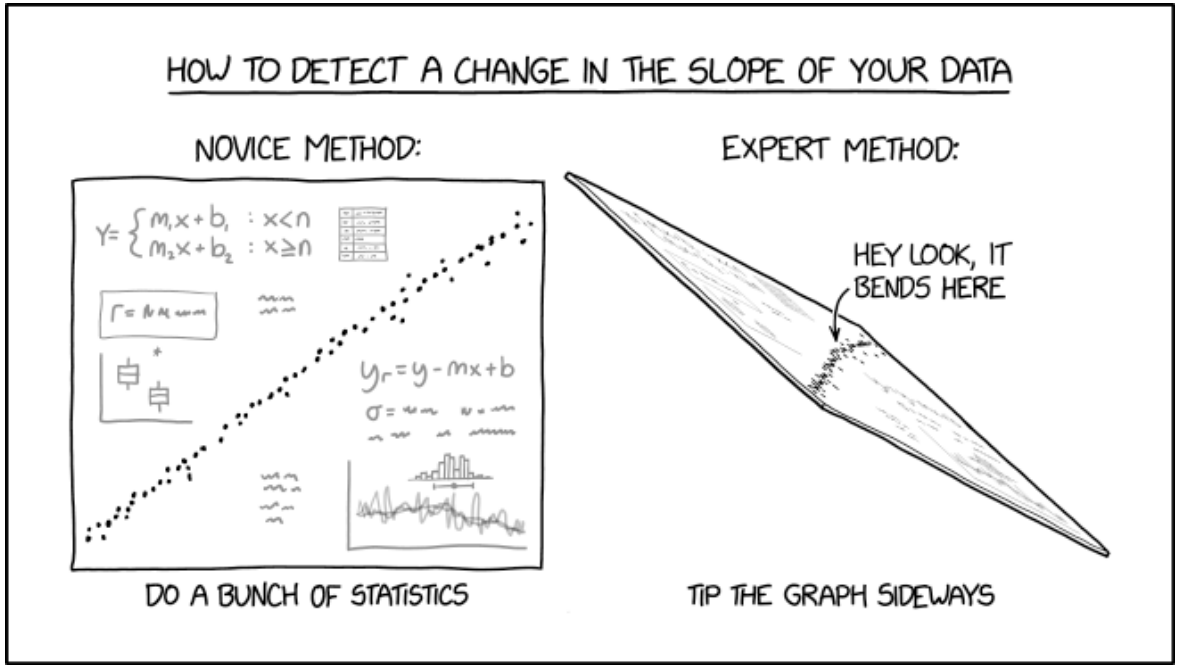
- Cramér–von Mises (compare empirical CDF of x to CDF of $\tilde{x}|x$)
- Kolmogorov–Smirnov (as above, but with a different metric)
- Anderson–Darling (as above, but with a different metric)
- Shapiro–Wilks (only for testing normality)

Graphical checks

The "qualitative" approach of directly visualising the data or test statistic is actually better than the "quantitative" methods above, since less information is discarded. It requires more manual intervention, but then that should be the nature of model validation in the first place.

The main things to do are:

- Examine posterior distribution of θ
- Compare posterior predictions of $\tilde{x}$ to x
- Compare posterior predictive distribution of $\tau(\tilde{x})$ to $\tau(x)$



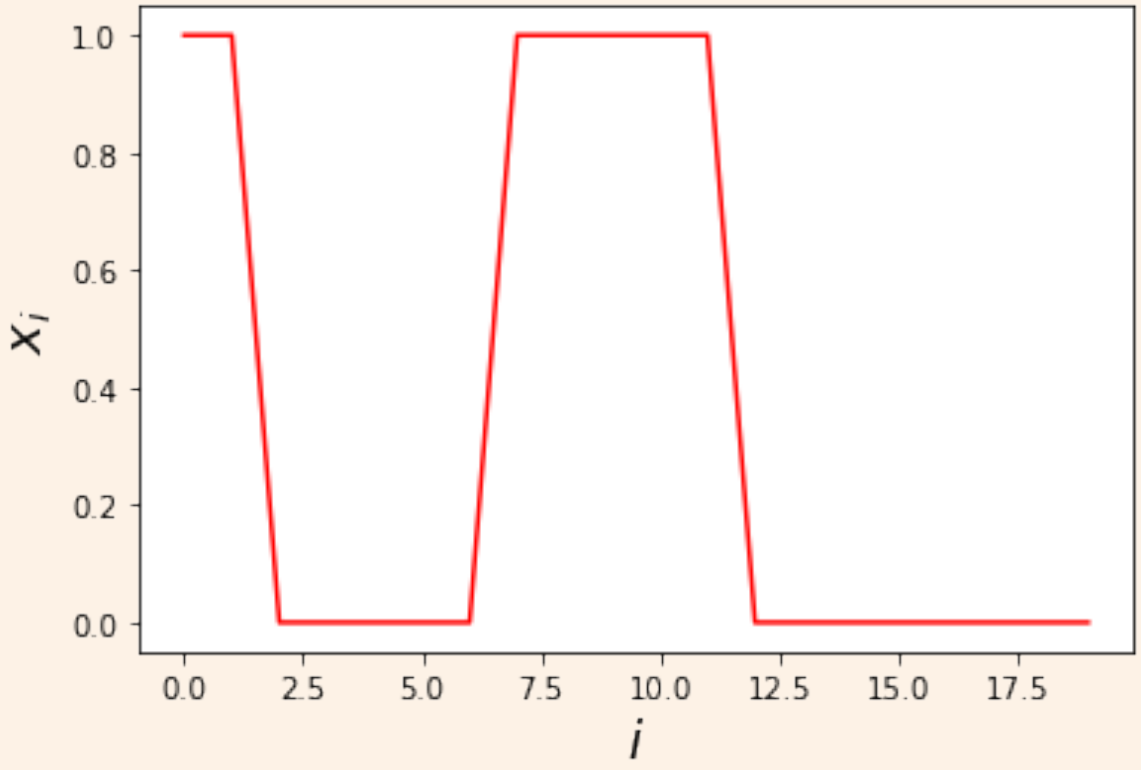
Credit: xkcd

Example

Jupyter notebook: Apply the binomial model to the observed sequence of binary-valued trials

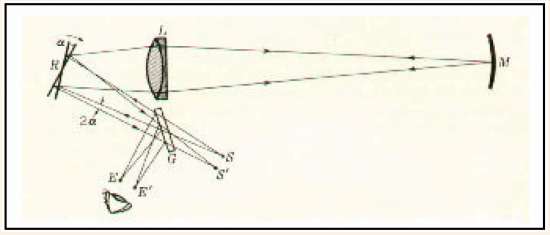
$$x = (1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0). \tag{151}$$

The sequence looks autocorrelated, so check the model using an appropriate test statistic.

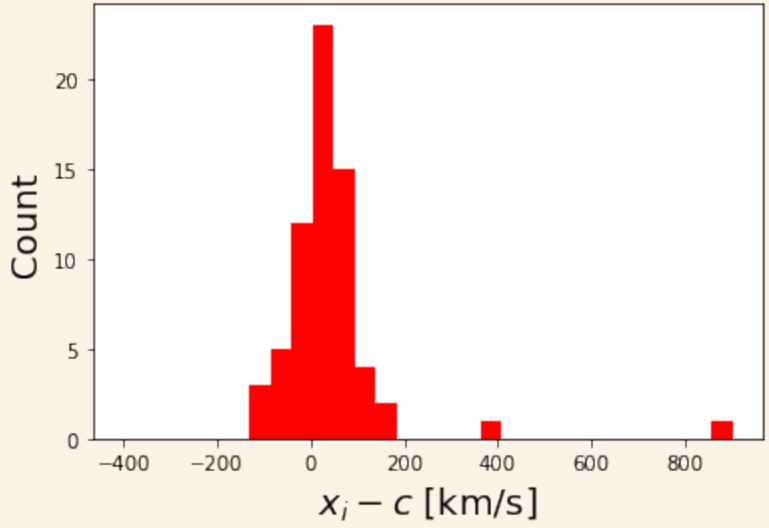


Example

Jupyter notebook: In 1882, Simon Newcomb measured* the speed of light c over a distance of 7.4 km, using a method devised by Léon Foucault. Apply the normal model (with unknown mean and variance) to his data, and validate it.



Credit: Carter & Carter, 2002



*Over the next 100 years, measurements of c became more precise than definitions of the metre, so c was defined in 1983 (with the metre redefined in terms of c).

The normal model does not provide an adequate fit to Newcomb's data — perhaps a skewed or heavy-tailed likelihood might be better.

This lack of fit is presumably caused by the presence of one or two outliers. However, it could well be that the "true" model for the experiment is not normal (more precisely, the nature of noise in the experiment really gives rise to a skewed or heavy-tailed distribution).

- The best approach is always to get more data if possible, which will help to clearly validate or invalidate a particular model.
- The next best approach is to improve the model to get a better fit.
- However tempting it is to remove "outliers", one should never do so unless there is a strong justification (e.g., independent knowledge that some data points are rubbish).

Example

Jupyter notebook: Remove just the strongest outlier from Newcomb's data (as he did), and reassess the validity and accuracy of the normal model.

Posterior Predictive Accuracy

Posterior predictive checks Is $p(\tilde{x}|x)$ consistent with the observed data x?

Posterior predictive accuracy Is $p(\tilde{x}|x)$ consistent with out-of-sample data $\tilde{x}$ (not replications)?

In practical applications, there might be certain quantities $f(\tilde{x})$ whose inference/prediction is prioritised. One may then construct specific utility/cost functions based on those quantities, and use them to measure model performance.

More generally, predictive accuracy can be assessed with generic calculations on $\tilde{x}$ itself, e.g., a (frequentist) point-prediction error $|\tilde{x} - \mathrm{E}[\tilde{x}|\theta]|$. For Bayesian probabilistic prediction, the predictive accuracy can be summarised by the logarithm of the posterior predictive density *value*. To streamline the presentation, we denote the posterior predictive density value by p_p:

$$p_p(\cdot) := \int d\theta \, p(\cdot|\theta)p(\theta|x), \tag{152}$$

and its logarithm by l_p (neither of these is standard notation).

Both p_p and l_p are *relative* metrics of accuracy, with respect to some unspecified baseline.

We will also make use of the point-predictive density value $p(\cdot|\theta)$ as a function of θ (i.e., the probability density of some data under the model with specific parameters θ), as well as its distribution under the posterior.

The full posterior predictive density value is then $p_p(\tilde{x})$. For more generality, we can write this in terms of individual data points $\tilde{x}_i$ (assumed to be iid observations):

$$p_p(\tilde{x}) = \prod_i p_p(\tilde{x}_i), \quad l_p(\tilde{x}) = \sum_i l_p(\tilde{x}_i). \tag{153}$$

Exercise

What is the relationship between $l_p(\tilde{x})$ and $|\tilde{x} - \mathrm{E}[\tilde{x}|\theta]|$ for a normal model?

If $\tilde{x}$ is available, then it is of course possible to compute the true predictive performance (external validation). We would like an estimate of the expected predictive performance (internal validation) without $\tilde{x}$. This is clearly a far more difficult task, but there are proposed ways of doing so by using only x.

The general idea of posterior predictive accuracy estimation is to compute $l_p(x)$ or some approximation, and then apply some correction to estimate $l_p(\tilde{x})$ for out-of-sample data.

Note that $l_p(x)$ is *on average* an overestimate of $l_p(\tilde{x})$, because the model was fitted to x and not $\tilde{x}$. The correction can be done in two ways: information criteria and cross-validation.

Information Criteria

Information criteria estimate the relative amount of information lost when fitting a model to data. They measure goodness of fit, penalised by the fitting power of the model.

Akaike information criterion (AIC)

The AIC is not Bayesian, as it uses the point-predictive density rather than the posterior predictive density. The particular point in question is the *maximum-likelihood estimate* (MLE), i.e., the value of θ that maximises the likelihood function $L(\theta|x)$. We denote this by $\hat{\theta}_{\mathrm{MLE}}$.

Exercise

Show that $\hat{\theta}_{\mathrm{MLE}}$ for the normal model (known variance, multiple data) is the sample mean.

The AIC is then defined as the *deviance* for the log point-predictive density with a correction:

$$\mathrm{aic} := -2 \times (\ln p(x|\hat{\theta}_{\mathrm{MLE}}) - k), \tag{154}$$

where k is the number of parameters in the model. Again, the deviance of a model is relative to some unspecified baseline. A lower deviance indicates higher predictive accuracy.

- In some contexts, the deviance is defined with respect to some best-fitting model.
- The "Bayesian" information criterion (BIC) replaces k with $(k/2) \ln n$, where n is the sample size. This estimates model evidence rather than predictive accuracy.

Deviance information criterion (DIC)

The DIC is slightly more Bayesian in nature, as it still uses the point-predictive density but for the posterior mean $\hat{\theta}_p := \mathrm{E}_p[\theta] := \mathrm{E}[\theta|x]$. It also replaces the simple correction k with

$$k_{\mathrm{dic}} := 2 \times (\ln p(x|\hat{\theta}_p) - \mathrm{E}_p[\ln p(x|\theta)]), \tag{155}$$

such that

$$\mathrm{dic} := -2 \times (\ln p(x|\hat{\theta}_p) - k_{\mathrm{dic}}). \tag{156}$$

By analogy to the AIC, k_{dic} can be interpreted as an effective number of parameters that is now estimated from the data. Its second term is Bayesian: it is the posterior expectation of the log point-predictive density. Note that this is not the same as the log posterior predictive density! The former is "expectation of log", while the latter is "log of expectation".

Watanabe–Akaike information criterion (WAIC)

The WAIC is more fully Bayesian. It replaces the point-predictive density $p(\cdot|\theta)$ in the DIC with the posterior predictive density $p_p(\cdot)$:

$$k_{\mathrm{waic}} := 2 \times (l_p(x) - \mathrm{E}_p[\ln p(x|\theta)]) \approx \mathrm{Var}_p[\ln p(x|\theta)], \tag{157}$$

$$\mathrm{waic} := -2 \times (l_p(x) - k_{\mathrm{waic}}). \tag{158}$$

The variance approximation of k_{waic} is more stable to the data, and can be used as the definition instead.

Cross-validation

It is evident that the practical difficulty in estimating predictive accuracy for truly out-of-sample data is the lack of access to said data (otherwise one can always treat it as in-sample). The idea behind cross-validation is to artificially leave out some observed data for that purpose, then to average over the left-out data by repeating the analysis multiple times.

Beyond serving as an estimate of predictive accuracy, cross-validation is a more general principle in statistics to assess the robustness of modelling/analysis, i.e., how well their results will generalise to an independent data set.

- It is a form of resampling, like classical methods such as jackknife and bootstrap.
- Notably, it is used in machine learning to check whether models are overfitting, or as a regularisation method to mitigate overfitting.

We will focus here on leave-one-out (LOO) cross-validation, where only a single data point is left out at each time.

- In the context of inference and prediction, LOO cross-validation can be used to examine the effect of individual data points on the results.
- More generally, LOO cross-validation can be impractical when the size n of the data set is large. A more feasible alternative is k-fold cross-validation and its variants, where n/k data points are left out at each time.

Recall from Eq. (153) that $l_p(x) = \sum_i^n l_p(x_i)$. To obtain a LOO estimate of $l_p(x)$, we:

- Compute n posteriors with each data point left out
- Compute the log predictive density for each point under the posterior where it was left out
- Sum those values instead of the full log posterior predictive density for each point

Some useful notation: $x_{[-i]}$ denotes x with x_i left out, while

$$p_{p[-i]}(x_i) := \int d\theta \, p(x_i|\theta)p(\theta|x_{[-i]}), \quad l_{p[-i]}(x_i) := \ln p_{p[-i]}(x_i), \tag{159}$$

$$l_{\text{loo}}(x) := \sum_i^n l_{p[-i]}(x_i). \tag{160}$$

Like $l_p(\tilde{x})$, the LOO estimate $l_{\text{loo}}(x)$ is generally lower than $l_p(x)$ because each of the n sub-models are fitted to slightly different x ($x_{[-i]}$), as opposed to a single model fitted to x.

By analogy to the information criteria, we define the LOO cross-validation estimate of posterior predictive accuracy as the deviance associated with $l_{\text{loo}}(x)$:

$$\text{loocv} := -2 \times l_{\text{loo}}(x). \tag{161}$$

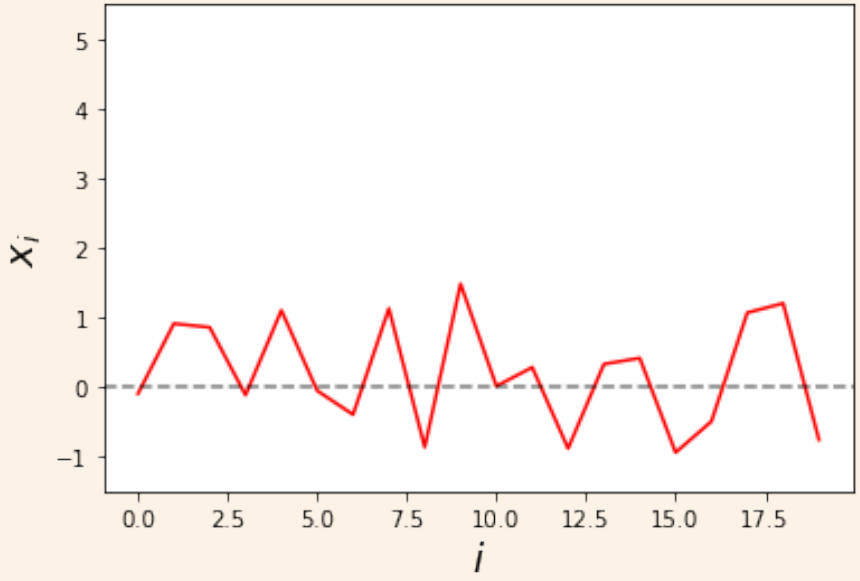
It is possible to derive a bias correction to loocv for small n, but this is typically negligible.

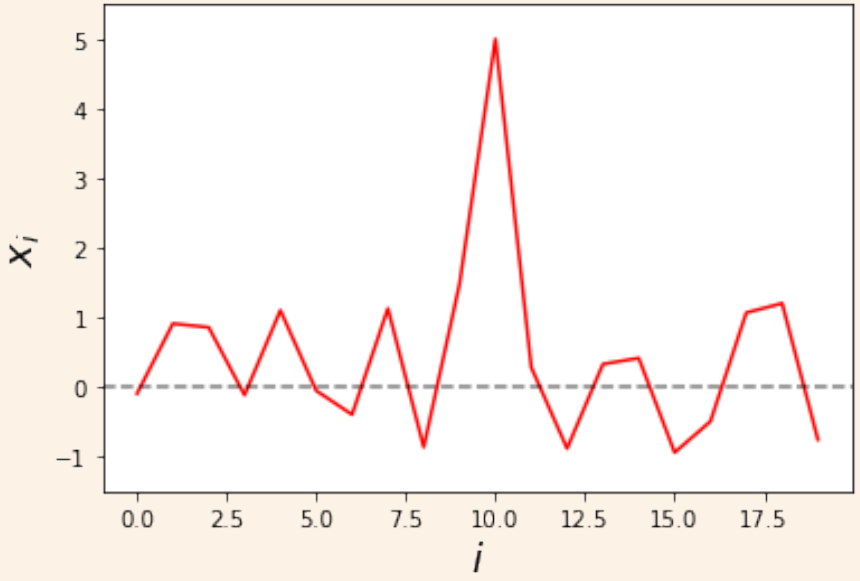
Example

Jupyter notebook: A data set of 20 observations is generated from a standard normal distribution. What is the posterior predictive accuracy for the following models?

1. $\mathcal{N}(\theta, 1)$
2. $\mathcal{N}(\theta, 2)$

Repeat the analysis for the same data set with a single outlier replacement $x_i = 5$.





Bayes Factors and Posterior Odds

Model comparison only requires a metric for relative performance, so it can also be done directly without any explicit validation. It can also be used for hypothesis testing.

The broad idea for comparing models directly in a fully Bayesian sense is to take a hierarchical view: to treat the model itself as a hyperparameter (discrete or otherwise). By analogy to Eq. (17), one may then examine the hyperposterior odds

$$\frac{p(\mathcal{M}_1|x)}{p(\mathcal{M}_2|x)} = \frac{L_{\mathcal{M}}(\mathcal{M}_1|x)}{L_{\mathcal{M}}(\mathcal{M}_2|x)} \frac{\pi_{\mathcal{M}}(\mathcal{M}_1)}{\pi_{\mathcal{M}}(\mathcal{M}_2)} \equiv \frac{Z_1}{Z_2} \frac{\Pi_1}{\Pi_2}, \tag{162}$$

where the marginal hyperlikelihood

$$Z_i := p(x|\mathcal{M}_i) = \int d\theta_i \, p(x|\theta_i, \mathcal{M}_i) p(\theta_i|\mathcal{M}_i) \tag{163}$$

is precisely the evidence for the data under each model. The hyperlikelihood or evidence ratio Z_1/Z_2 is known as the Bayes factor — i.e., the model posterior odds are equal to the Bayes factor times the model prior odds Π_1/Π_2. We denote the Bayes factor by $\mathrm{bf}^i_j := Z_i/Z_j$.

The Bayes factor has a built-in "Occam penalty", in that model complexity is penalised. Simpler models with fewer d.o.f. can be preferred even if they are a worse fit to the data.

- In fact, using a normal approximation, it can be shown that the model evidence is related to $-1/2$ times the BIC at leading order: $Z_i = \mathrm{ML} - (k/2) \ln n + \mathcal{O}(1)$ as $n \to \infty$.

A very common scenario in model comparison is that of nested models. Two models are nested if they share a common form for their likelihood and prior, but those for one model are conditioned on specific values for a subset of parameters: $\theta = \theta_0$.

- For example: consider an experiment to distinguish between general relativity $\mathcal{M}_G$, and an alternative theory of gravity $\mathcal{M}_A$ that reduces to $\mathcal{M}_G$ if some of its parameters $\theta = \theta_0$. Then $p(x|\phi, \mathcal{M}_G) = p(x|\theta = \theta_0, \phi, \mathcal{M}_A)$, where ϕ are effectively nuisance parameters.

For nested models $\mathcal{M}_i \subset \mathcal{M}_j$, bf^i_j can be computed directly via the Savage–Dickey ratio:

$$\mathrm{bf}^i_j = \frac{p(\theta = \theta_0|x, \mathcal{M}_j)}{p(\theta = \theta_0|\mathcal{M}_j)}, \tag{164}$$

where the numerator is the value of the (marginal) posterior density for θ at θ_0 under $\mathcal{M}_j$, and the denominator is the value of the (marginal) prior density for θ at θ_0 under $\mathcal{M}_j$.

Exercise

Show that the Savage–Dickey ratio equals the Bayes factor for nested models.

Example

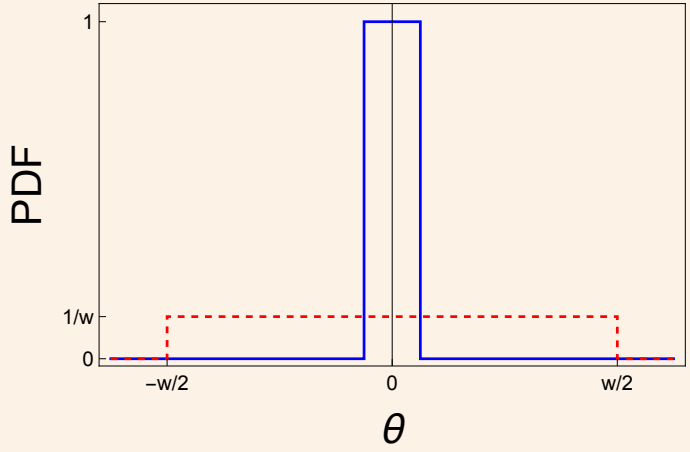
What is the probability that a coin landing on heads 140 out of 250 times is fair?

Disadvantages of Bayes factors:

- Require explicit evidence calculations for general models, which are computationally costly
- Can be very sensitive to prior choice
- The actual values can be misleading and difficult to interpret

Example

Consider a model with likelihood $L(\theta|x) = \mathbf{1}_{|\theta - x| < 1/2}(\theta)$ and prior $\pi(\theta) = \mathcal{U}(\theta| - w/2, w/2)$. Let $x = 0$ and $w > 1$. What is the Bayes factor for the two hypotheses $\theta = 0$ and $\theta \neq 0$?



The misuse of Bayes factors (and bad statistics in general) can lead to embarrassing mistakes in science. In 2020, someone from MIT proposed the "measurement" of the mathematical constant π from gravitational-wave observations of astrophysical binaries.

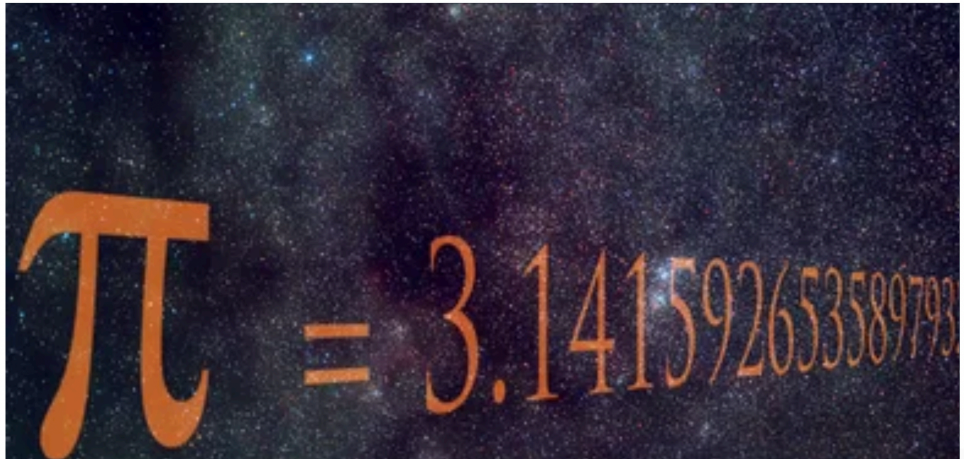
- These sources are modelled assuming general relativity, and semi-analytical models of their signals contain π in their expressions for the signal phase.
- The idea is to treat π as a parameter and perform Bayesian inference on it.
- Measuring π is just the attention-grabbing headline — the actual aim is to test relativity by checking if the posterior for π is consistent with its actual value.
- A Bayes factor of > 300 was reported in favour of relativity.

SPACE & PHYSICS

Pi in the Sky: General Relativity Passes the Ratio's Test

Using gravitational waves to approximate pi, physicists see no problem with Einstein's theory

By Daniel Garisto on May 19, 2020

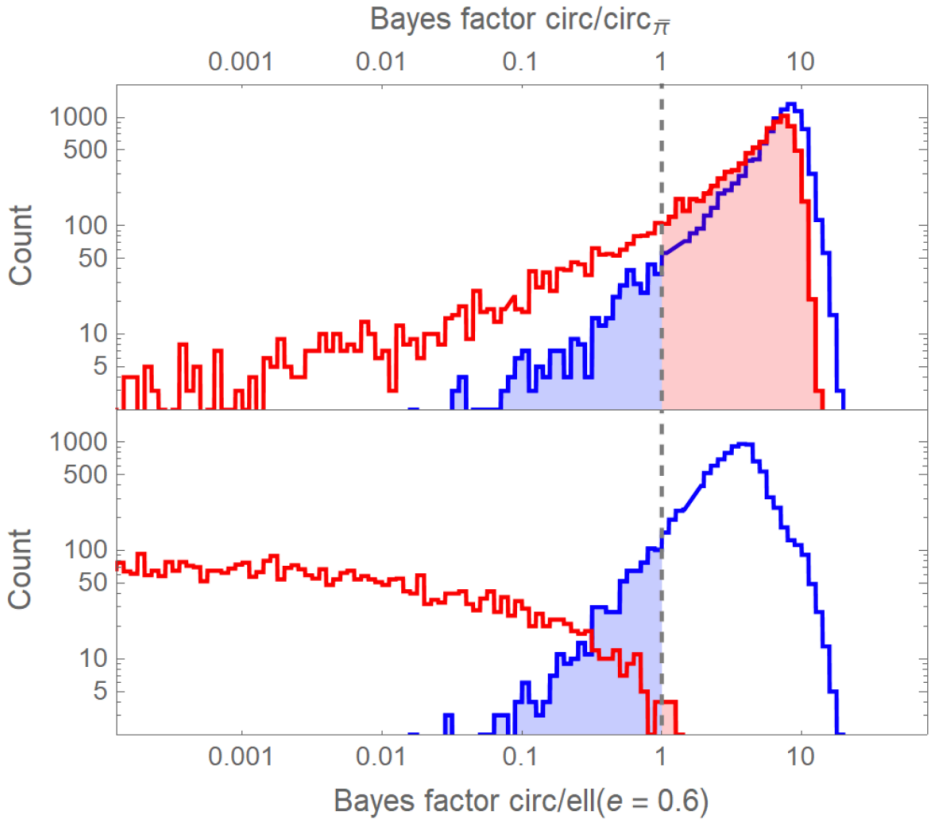


Credit: Scientific American

Statisticians generally recommend that Bayes factor values $\gtrsim 100$ can be interpreted as decisive. So this seems like a pretty strong test of relativity, right?

Not really. It suffers from many logical and practical problems, including:

- What is relativity actually being compared to here?
- What is the sensitivity of the result to the arbitrary choice of prior?
- What is the sensitivity of the result to the correctness of the likelihood?



Credit: Chua & Vallisneri, 2020

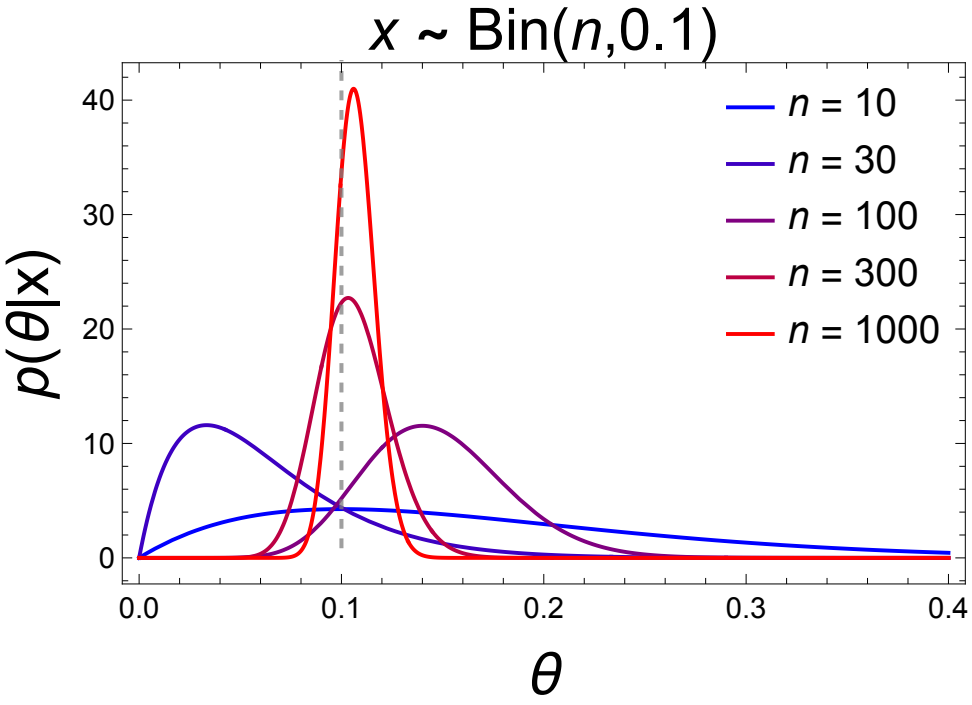
11. Asymptotics and Connections

Asymptotic Normality of the Posterior

Under certain assumptions, the Bayesian posterior distribution can be shown to converge to a normal distribution in the limit of large sample size. This is known as asymptotic normality.

Asymptotic normality might be viewed as the Bayesian analogue of the central limit theorem, but is not quite as universally valid because there is more freedom in the posterior for it to fail.

To illustrate this, recall that the binomial model is essentially a Bernoulli model with n iid observations. The posterior distribution for the binomial model with known n and a uniform prior is $\theta|x \sim \text{Beta}(x+1, n-x+1)$. We may examine the posterior density as $n \to \infty$, with $x \sim \text{Bin}(n, 0.1)$:



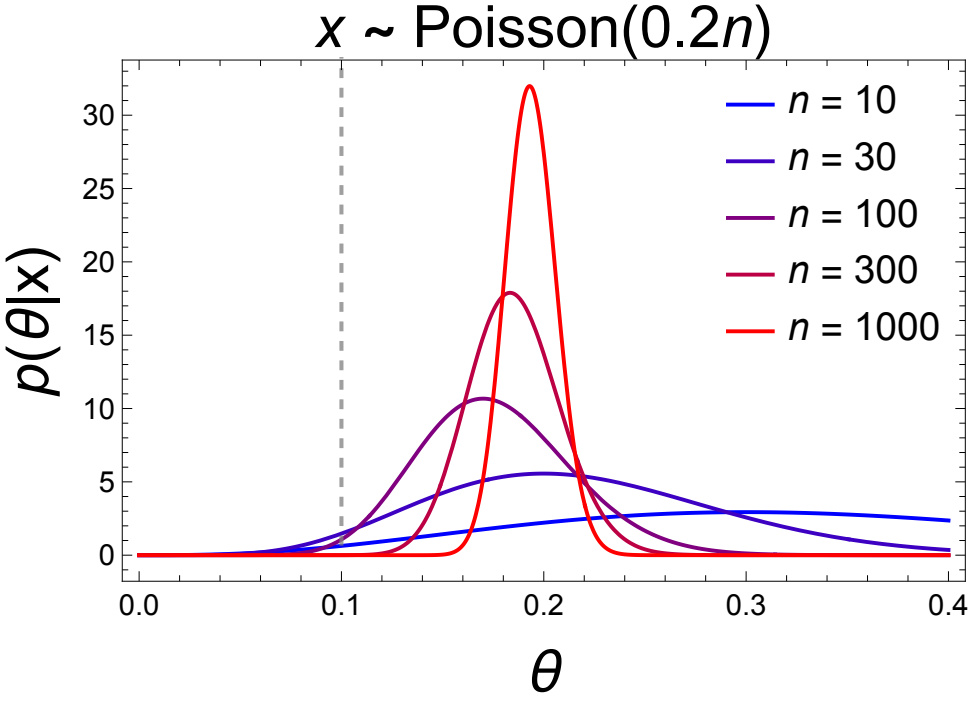
Asymptotic consistency

This closely related property holds when we assume the model is "correct", i.e., the data is actually distributed according to some underlying distribution p_t, and $p_t(x) = p(x|\theta_t)$ for some "true" parameters θ_t. Then as $n \to \infty$:

$$\theta_{\mathrm{MAP}} := \max_{\theta} \{p(\theta|x)\} \to \theta_t, \tag{165}$$

where θ_{MAP} is known as the *maximum a posteriori* (MAP) estimate.

In the above example, what happens if the data is not actually obtained from Bernoulli trials, but from a Poisson process with some average rate, say $\lambda/n = 0.2$?



If the model is incorrect, θ_{MAP} converges instead to the parameters that minimise the Kullback–Leibler divergence of p_t from p:

$$D_{\mathrm{KL}}(p_t, p) := \int dx\, p_t(x) \ln\left(\frac{p_t(x)}{p(x|\theta)}\right). \tag{166}$$

- More generally, D_{KL} is a measure of discrepancy between two probability distributions
- It is not really a statistical distance, as it is asymmetric in its arguments; $D_{\mathrm{KL}}(p, q)$ can be interpreted as the information gained from p over q

In the above example we see that $\theta_{\mathrm{MAP}} \to \lambda/n$, because the Poisson distribution also approaches the binomial distribution as $n \to \infty$. But in general, θ_{MAP} does not have a meaningful interpretation in the context of the underlying distribution p_t.

Asymptotic normality and consistency means the posterior tends to normal with the correct maximum a posteriori estimate, which justifies the use of the normal approximation.

A sketched proof of asymptotic normality

For compactness, consider $\theta \in \mathbb{R}$ first, and adopt the shorthand notation $\hat{\theta} := \theta_{\mathrm{MAP}}$, $l(\theta) := \ln p(\theta|x)$. The Taylor expansion of l about $\hat{\theta}$ is then

$$l(\theta) = l(\hat{\theta}) + \partial_\theta l(\theta)|_{\theta=\hat{\theta}}(\theta - \hat{\theta}) + \frac{1}{2}\partial_\theta^2 l(\theta)|_{\theta=\hat{\theta}}(\theta - \hat{\theta})^2 + \mathcal{O}((\theta - \hat{\theta})^3). \tag{167}$$

The constant term is essentially a normalisation factor for the posterior, while the linear term vanishes as $\hat{\theta}$ is a stationary point of the posterior by definition. Thus the log posterior around $\hat{\theta}$ is quadratic at leading order.

Exercise

Using Bayes' theorem, show that the expectation (under the data distribution) of the quadratic coefficient (times two) is related to the Fisher information by

$$\mathrm{E}[\mathcal{I}_p(\hat{\theta})] := -\mathrm{E}[\partial_\theta^2 l(\theta)|_{\theta=\hat{\theta}}] = n\mathcal{I}(\hat{\theta}) + \mathrm{const.} \tag{168}$$

By analogy to the Fisher information, $\mathcal{I}_p$ in Eq. (168) is referred to as the *observed information*. It describes the posterior (with n iid observations) rather than the likelihood.

Eq. (168) shows that the quadratic term in Eq. (167) is $\mathcal{O}(n)$. The proof also relies on certain assumed regularity conditions, one of which is the boundedness of the third derivative. This results in all higher-order terms h being $o(n)$, i.e., $\lim_{n\to\infty} h/n = 0$.

Thus for large n we have

$$\mathrm{E}[l(\theta)] = -\frac{1}{2}\mathrm{E}[\mathcal{I}_p(\hat{\theta})](\theta-\hat{\theta})^2 + \mathcal{O}((\theta-\hat{\theta})^3) = -\frac{1}{2}n\mathcal{I}(\hat{\theta})(\theta-\hat{\theta})^2 + o(n), \tag{169}$$

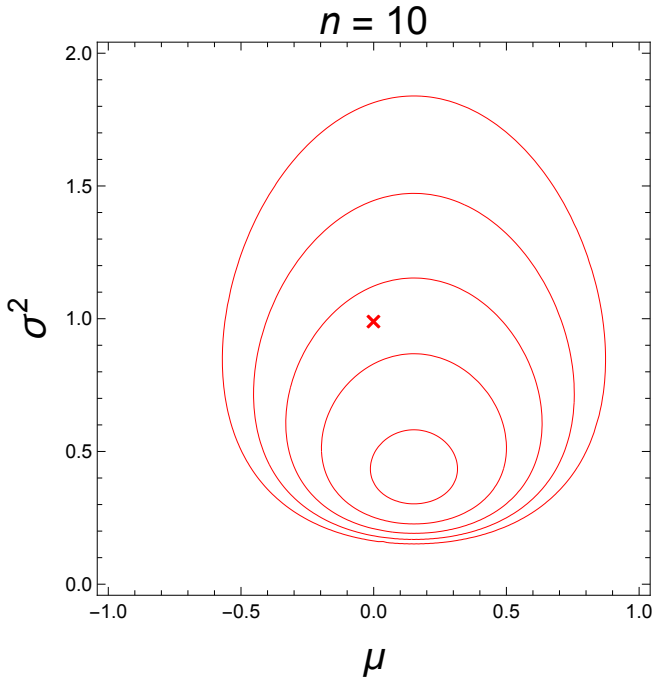
which describes the normal distribution $\mathcal{N}(\hat{\theta}, (n\mathcal{I}(\hat{\theta}))^{-1})$. Intuitively, this occurs because the likelihood precision is proportional to n, and dominates the prior precision for large n. Note also that this implies $p(\theta|x) \to \delta(\theta-\hat{\theta})$ as $n\to\infty$.

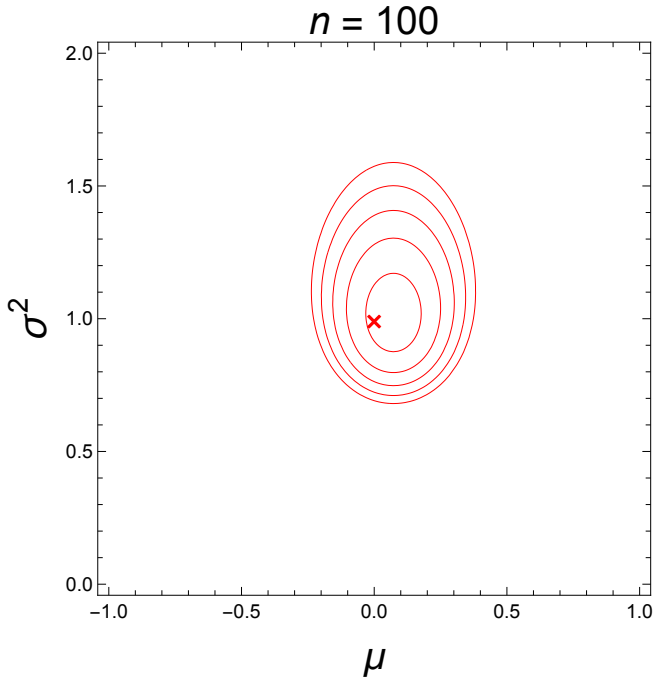
- We will not give the proof of asymptotic consistency here, but it uses similar arguments. It requires the law of large numbers (the convergence of expectations) and some basic real analysis (for continuous parameters).

The extension of the asymptotic normality result to the multivariate case is exactly analogous, and only requires a shift to vector and matrix quantities:

$$l(\theta) = -\frac{1}{2}[\theta - \hat{\theta}]^T \mathcal{I}_p(\hat{\theta})[\theta - \hat{\theta}] + \mathcal{O}(|\theta - \hat{\theta}|^3). \tag{170}$$

Here the observed information matrix is now the negative Hessian of the log posterior density, and describes its intrinsic local curvature.





Counterexamples to asymptotic normality

Asymptotic normality can fail for certain classes of posterior:

- Some parameters cannot be estimated from the data at all. This is a trivial example, as those parameters should not have been included in the first place.
- Some parameters are exactly degenerate and cannot be estimated independently, i.e., the model depends on θ_1 and θ_2 only through a fixed combination of the two. For example, $p(x|\theta) = \mathcal{N}(x|\theta_1\theta_2, 1)$. Then the posterior $p(\theta_1, \theta_2|x)$ converges to the curve $\theta_1\theta_2 = \text{const.}$ in the plane, rather than a point. This can usually be preidentified, and avoided through a suitable redefinition of the model.
- Some parameters are exchangeable, i.e., the model depends on the ordered pair (θ_1, θ_2) in the same way as it depends on (θ_2, θ_1). For example, the Gaussian mixture model $p(x|\theta) = (\mathcal{N}(x|\theta_1, 1) + \mathcal{N}(x|\theta_2, 1))/2$. Then the posterior $p(\theta_1, \theta_2|x)$ will have two modes that converge to the points $(\hat{\theta}_1, \hat{\theta}_2)$ and $(\hat{\theta}_2, \hat{\theta}_1)$. This is known as aliasing, and again can be preidentified and avoided.
- In a hierarchical model, the number of parameters might increase with n, e.g., if θ is a time series of length n that describes the drifting mean of a stochastic process. Then the posterior will not converge to a point in $\mathbb{R}^n$ (nor will any marginal in the corresponding subspace), since each x_i brings the same amount of information about its θ_i.
- With a bounded prior domain, θ_{MAP} can end up on the boundary of the prior. Then normality or even partial normality might fail. Also, an improper prior or an unbounded likelihood range might lead to an improper posterior (again trivial).

Normal Approximation in Practice

Thanks to asymptotic normality and consistency, most posterior distributions can be approximated as normal when there is enough data. This has obvious benefits:

- Ease of computing credible regions, expectations and probabilities
- Can use for importance sampling, proposals, etc. even if the approximation is not great

The procedure itself is conceptually straightforward:

1. Compute or estimate the MAP parameters θ_{MAP}
2. Analytically or numerically compute the observed information $\mathcal{I}_p = -\partial^2_\theta l(\theta)$ at θ_{MAP}
3. The normal approximation is then $\mathcal{N}(\theta_{\mathrm{MAP}}, \mathcal{I}_p^{-1})$

Example

Laplace's posterior for the probability of a female birth in Paris between 1745 and 1770 was $\mathrm{Beta}(x+1, n-x+1)$, with x/n the observed proportion. What is the normal approximation?

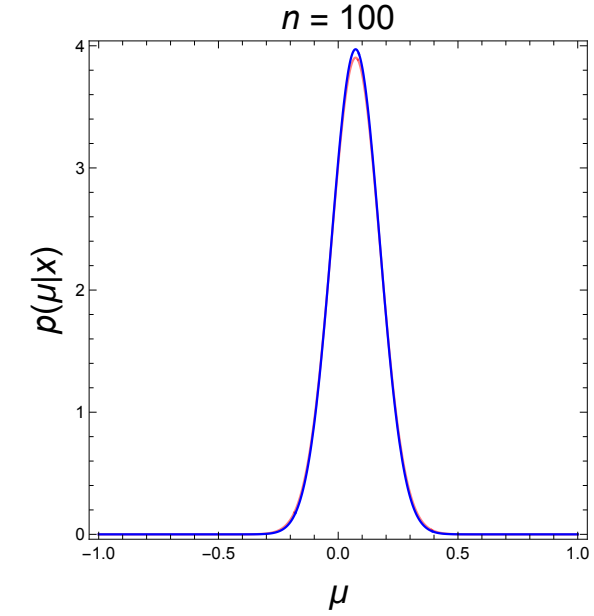
Example

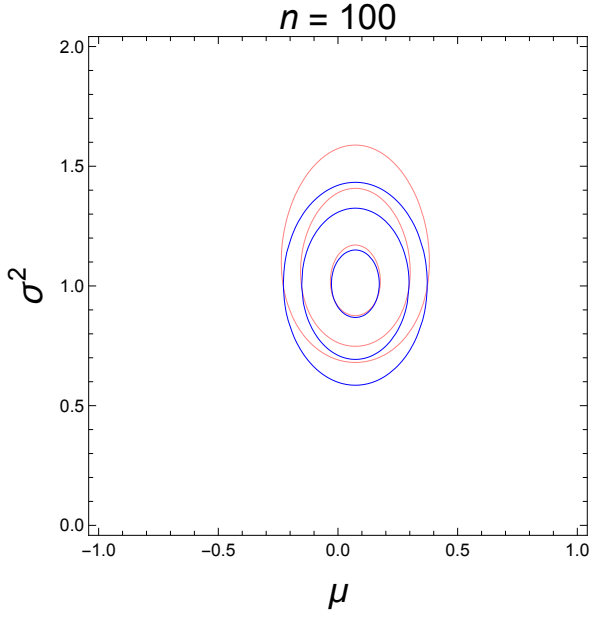
Show that the posterior in Eq. (52) for the two-parameter normal model is approximately

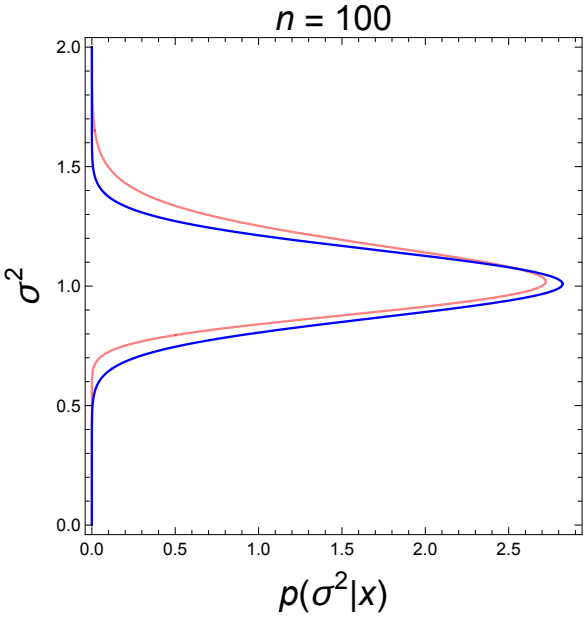
$$p(\theta_1, \theta_2|x) \approx \mathcal{N}\left(\theta_1, \theta_2 \middle| \begin{pmatrix} s_1(x) \\ s_2(x) \end{pmatrix}, \begin{pmatrix} s_2(x)/n & 0 \\ 0 & 2s_2(x)^2/n \end{pmatrix}\right). \tag{171}$$

$$p(\theta_1, \theta_2|x) \propto \theta_2^{-(n+2)/2} \exp\left(-\frac{1}{2}\frac{(n-1)s_2(x) + n(s_1(x)-\theta_1)^2}{\theta_2}\right)$$

$$s_1(x) = \frac{1}{n}\sum_i^n x_i, \quad s_2(x) = \frac{1}{n-1}\sum_i^n (x_i - s_1(x))^2.$$


n = 100
p(μ|x)
μ


n = 100
σ²
μ


n = 100
σ²
p(σ²|x)

In most realistic problems, neither θ_{MAP} nor $\mathcal{I}_p$ is analytically available. So for the normal approximation to be useful, we need techniques to obtain these two quantities with less effort then a full estimation of the posterior through sampling.

Finding θ_{MAP} is essentially an optimisation problem:

- Local maxima are "easy" to find with standard techniques, but may not even mean anything (the posterior density could be very noisy)
- Global maximum can thus be highly nontrivial to find in noisy/multimodal problems

Computing $\mathcal{I}_p$ is more generally achievable, but can also be nontrivial:

- Symbolic differentiation of the posterior density is assumed to not be an option (but even if it is, the expressions can be unwieldy to evaluate)
- Numerical differentiation can be unstable and inaccurate
- Automatic differentiation (essentially the chain rule) is not always an option

Optimisation and differentiation are of course extremely general tasks — so even in the context of the posterior density, they can be useful beyond their role in the normal approximation. For example, they are relevant to full posterior sampling through the initialisation of chains at local modes, or through gradient-based methods like HMC.

Other distributional approximations to the posterior can also be useful, should the normal approximation be too poor for a particular purpose.

One obvious thing to try is to use higher-order expansions of the log posterior density, by computing up to the third derivative (related to skewness) or the fourth derivative (related to kurtosis) of $\ln p(\theta|x)$ at θ_{MAP}.

- These are higher-order tensors for multivariate posteriors, so can be unwieldy
- The resultant distributions can be ill behaved in the tails

Another possibility is to fit non-normal distributions that are still analytically tractable:

- Split-normal and skew-normal distributions for skewness
- Generalised t distribution for excess kurtosis (heavy tails relative to normal)

More generally, the principle of fitting exact distributions to the posterior is known as variational inference, and has important applications in machine learning as well. The main idea is to minimise the Kullback–Liebler divergence or some other statistical distance between the posterior and a parametric family of distributions. Example frameworks include:

- Variational Bayes
- Expectation propagation

Computing Derivatives

Numerical differentiation

This refers to the method of approximating derivatives with finite differences, i.e., the differences between evaluations of the functions at different points.

By definition, the first-order derivative of $f(\theta)$ can be written as a forward difference:

$$f'(\theta) = \lim_{\delta \to 0} \frac{f(\theta + \delta) - f(\theta)}{\delta} = \frac{f(\theta + \delta) - f(\theta)}{\delta} + \mathcal{O}(\delta), \tag{172}$$

and equivalently as a backward or central difference:

$$f'(\theta) = \frac{f(\theta) - f(\theta - \delta)}{\delta} + \mathcal{O}(\delta), \tag{173}$$

$$f'(\theta) = \frac{(1/2)f(\theta + \delta) - (1/2)f(\theta - \delta)}{\delta} + \mathcal{O}(\delta^2). \tag{174}$$

Likewise, higher-order derivatives can be built up from finite differences:

$$f''(\theta) = \lim_{\delta \to 0} \frac{f'(\theta + \delta) - f'(\theta)}{\delta} = \frac{f(\theta + 2\delta) - 2f(\theta + \delta) + f(\theta)}{\delta^2} + \mathcal{O}(\delta), \tag{175}$$

$$f''(\theta) = \frac{f(\theta) - 2f(\theta - \delta) + f(\theta - 2\delta)}{\delta^2} + \mathcal{O}(\delta), \tag{176}$$

$$f'(\theta) = \frac{f(\theta + \delta) - 2f(\theta) + f(\theta - \delta)}{\delta^2} + \mathcal{O}(\delta^2). \tag{177}$$

Partial derivatives can be similarly approximated, e.g., the central difference

$$\partial_{\theta_1}\partial_{\theta_2} f(\theta) = \frac{(1/4)f(\theta_1+\delta_1,\theta_2+\delta_2) - (1/4)f(\theta_1+\delta_1,\theta_2-\delta_2)}{\delta_1\delta_2} + \frac{-(1/4)f(\theta_1-\delta_1,\theta_2+\delta_2) + (1/4)f(\theta_1-\delta_1,\theta_2-\delta_2)}{\delta_1\delta_2} + \mathcal{O}(\delta_1\delta_2). \tag{178}$$

The grid of points at which a function should be evaluated to compute finite differences is known as a *stencil* (often a regularly spaced grid as above).

Central-difference derivatives are more accurate given the same number of stencil points, although one may improve the accuracy of one-sided differences by increasing the number of points, e.g., the forward difference

$$f'(\theta) = \frac{-(1/2)f(\theta+2\delta) + 2f(\theta+\delta) - (3/2)f(\theta)}{\delta} + \mathcal{O}(\delta^2). \tag{179}$$

Numerical differentiation is very general and sometimes the only option, but ensuring the stability of results (to the choice of stencil) can be more trouble than it's worth.

Automatic differentiation

This method (aka autodiff) is a middle-ground between symbolic and numerical differentiation. It is suitable for any function that is a composition of elementary functions.

Consider an example function

$$f(\theta) = \theta_1 \theta_2 + \sin \theta_1, \tag{180}$$

whose gradient we wish to evaluate at some point $\theta = \hat{\theta}$.

- The symbolic approach is to work out $\partial f(\theta) = (\theta_2 + \cos \theta_1, \theta_1)$, then evaluate it at $\hat{\theta}$.
- The numerical approach is to take finite differences of $f(\theta)$ evaluated at and near $\hat{\theta}$.

Autodiff expresses the function and its gradient as a composition of functions, which allows one to evaluate the gradient exactly without having an explicit expression for it (on paper). For the above example, the function is decomposed as

$$f(\theta) = f_5(f_3(f_1(\theta), f_2(\theta)), f_4(f_1(\theta))), \tag{181}$$

where

$$\begin{aligned} f_1(\theta) = \theta_1, \quad f_2(\theta) = \theta_2, \quad f_3(f_1, f_2) = f_1 f_2, \\ f_4(f_1) = \sin f_1, \quad f_5(f_3, f_4) = f_3 + f_4. \end{aligned} \tag{182}$$

In *forward-accumulation* autodiff, we construct ∂f from the total derivatives of the f_i with respect to θ. The gradient components are evaluated separately, so here there are effectively two passes through the algorithm (we write this in vector shorthand):

$$\begin{aligned}
\partial f_1 &= (1, 0), \\
\partial f_2 &= (0, 1), \\
\partial f_3 &= f_1 \partial f_2 + f_2 \partial f_1 = (\theta_2, \theta_1), \\
\partial f_4 &= (\cos f_1) \partial f_1 = (\cos \theta_1, 0), \\
\partial f_5 &= \partial f_3 + \partial f_4 = (\theta_2 + \cos \theta_1, \theta_1).
\end{aligned} \tag{183}$$

- The RHS of the first two lines are called seed values.
- Given $f_1 = \hat{\theta}_1$ and $f_2 = \hat{\theta}_2$, we evaluate the last three lines in sequence to obtain $\partial f(\hat{\theta}) = \partial f_5$ as desired.
- Note that the explicit expressions in each of the last three lines (the second equality) are not needed in the algorithm, and only included here for illustrative purposes.

In *reverse-accumulation* autodiff, we instead construct ∂f from the total derivatives of f with respect to the f_i. The difference is not mathematical of course — only algorithmic.

- Now the gradient components are evaluated in a single pass. This sounds superior to forward accumulation, but for vector-valued functions $f : \mathbb{R}^m \to \mathbb{R}^n$, reverse accumulation requires n passes and so it can be slower if $n \gg m$.

Denoting the total derivative of f with respect to f_i by $\partial_i f$, we evaluate:

$$\begin{aligned}
\partial_5 f &= 1, \\
\partial_4 f &= \partial_5 f = 1, \\
\partial_3 f &= \partial_5 f = 1, \\
\partial_2 f &= (\partial_3 f) f_1 = \theta_1, \\
\partial_1 f &= (\partial_3 f) f_2 + (\partial_4 f) \cos f_1 = \theta_2 + \cos \theta_1.
\end{aligned} \tag{184}$$

- Similar to before, the RHS of the first line is the seed value.
- Similar to before, the explicit expressions in each of the last four lines are not needed.
- The method known as backpropagation in the training of artificial neural networks is an example of reverse-accumulation autodiff.

Autodiff works great if models are built from the ground up with elementary functions.

- In theory, it can be used for anything that can be expressed on a computer.
- In practice, this can be difficult, e.g., for an ODE/PDE solver, one needs to differentiate not just w.r.t. the states and parameters, but also all the (tuning) hyperparameters.

Optimisation vs Sampling

There is a close relationship between these two tasks: optimisation can be used to inform sampling, and sampling can be used for optimisation.

For Bayesian inference, one could in principle perform optimisation to find local/global modes of the posterior density, then perform full sampling or use the normal approximation.

- The reason is that optimisation algorithms are usually more efficient than sampling algorithms for optimisation, but only if the density is well behaved.
- In practical applications, this is not necessarily the case! Especially globally.

We will only briefly review the most common optimisation algorithms here, and only in the context of finding modes of probability densities.

General optimisation

- Newton–Raphson: Taylor-expand the objective function f about each iteration point θ and optimise the quadratic approximation, i.e., $\theta \to \theta - [\partial_\theta^2 f(\theta)]^{-1} \partial_\theta f(\theta)$
- Gradient descent/ascent: Move in direction of gradient at each iteration
- Conjugate gradient: As above, but the direction at each iteration must be conjugate to all past directions (orthogonal w.r.t. Hessian)
- Nelder–Mead: Evolve simplices ($d + 1$ vertices in d dimensions) to more optimal values using simple geometric transformations
- Stochastic algorithms: Deterministic methods such as those above can often be augmented with stochastic aspects for better robustness and generalisability

Conditional maximisation

This is a Gibbs-like method for maximising multivariate probability densities. Rather than sampling from each conditional distribution, the aim is simply to maximise each conditional density. Again, even if the conditionals are not standard, they are still one-dimensional and relatively straightforward to maximise.

Example

Starting from $(\theta_1, \theta_2) = (-2, 2)$, perform conditional maximisation on the density

$$p(\theta) = \mathcal{N}(\theta|0, \Sigma), \quad \Sigma = \begin{pmatrix} 1 & 1/2 \\ 1/2 & 1 \end{pmatrix}. \tag{185}$$

Expectation maximisation

This method can be viewed as a generalisation of conditional maximisation. Here the aim is to maximise a marginal density $p(\theta)$ rather than some joint density $p(\theta, \phi)$.

Algorithm

1. Choose a starting point θ_0
2. Expectation step: At iteration i, compute the expectation of the log joint density under the conditional density $p(\phi|\theta_{i-1})$, i.e.,

$$f(\theta|\theta_{i-1}) := \int d\phi \, \ln p(\theta, \phi) p(\phi|\theta_{i-1}) \tag{186}$$

3. Maximisation step: Compute the maximum of $f(\theta|\theta_{i-1})$ and set it as θ_i

The maximisation step can be softened to merely increase the value of $f(\theta|\theta_i)$ — this is known as generalised expectation maximisation.

- Expectation maximisation is more widely applicable than its presentation here, especially if the quantities to be marginalised over are interpreted as missing data (the original context) or latent variables (for fitting problems in machine learning).
- There, the conditional density can be defined as something tractable when setting up the problem (rather than defining a joint density with a potentially intractable conditional).

Example

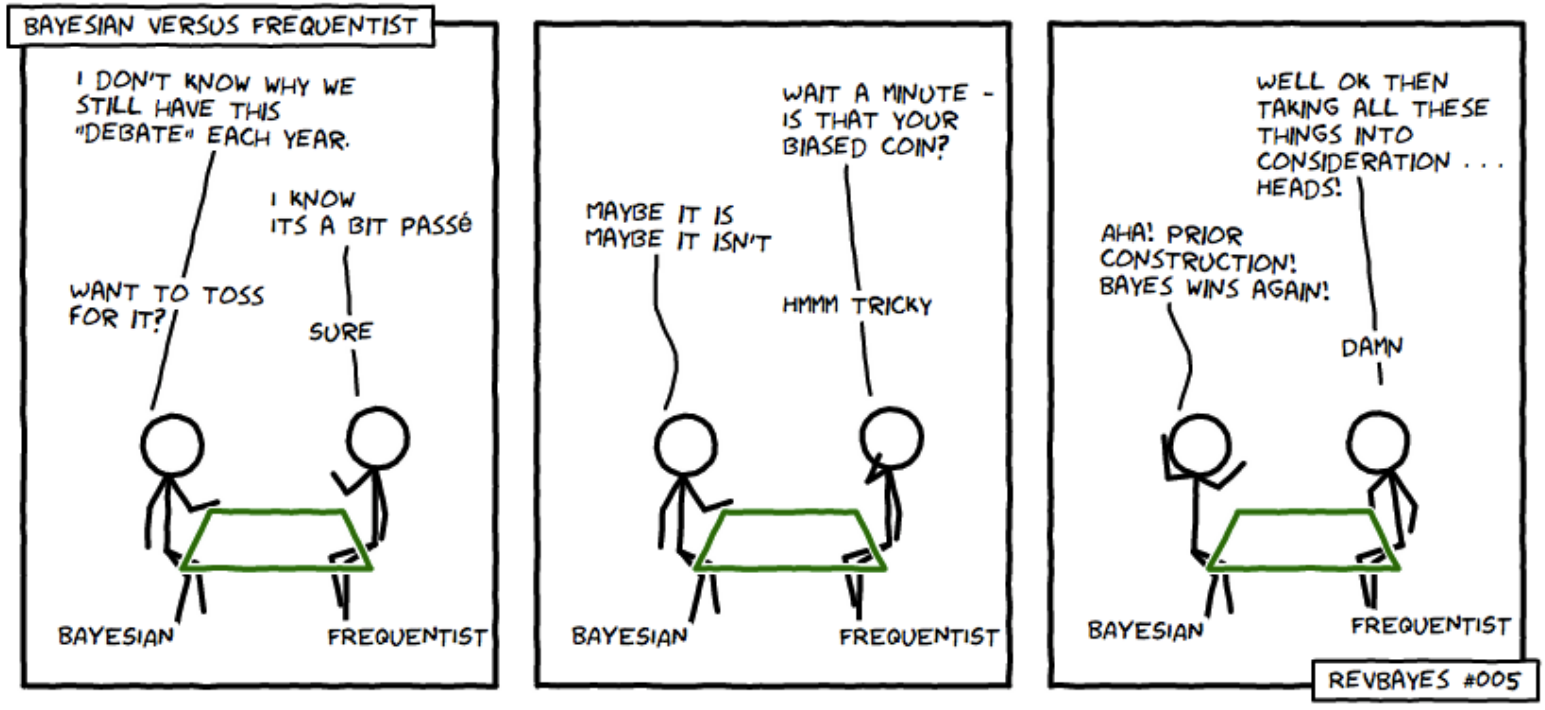
Starting from $(\theta_1, \theta_2) = (-2, 2)$, perform expectation maximisation on the marginal densities $p(\theta_1)$ and $p(\theta_2)$ for the joint density in Eq. (185).

Connections to Frequentist Statistics

Recall that frequentist statistics deals only with aleatoric uncertainty, and the interpretation of probability as frequency. This is actually present in Bayesian statistics too, and it can be useful to examine the frequency properties of Bayesian models, e.g., our study of asymptotics essentially considers an aleatoric repetition of the data/experiment as $n \to \infty$.

To be clear though: it's not as simple as Bayesian good, frequentist bad (or vice versa)!

- There are many useful methods from frequentist statistics
- The difference in results is often minimal, especially for simple models and/or large data
- Machine learning uses both frequentist and Bayesian techniques



Confidence regions vs credible regions

Confidence regions (or intervals in one dimension) are used in frequentist statistics to provide an uncertainty estimate for the parameters θ under the data distribution $p(x|\theta)$. For example, the $C\%$ confidence interval of the normal mean μ is

$$[\mu_L, \mu_R] := \left[s_1 - c\sqrt{\frac{s_2}{n}}, s_1 + c\sqrt{\frac{s_2}{n}} \right], \tag{187}$$

where s_1 and s_2 are the sample mean and variance respectively, and c is chosen such that $P(\mu_L \leq \mu \leq \mu_R) = C/100$ (under $p(s_1, s_2|\theta)$). In this setup:

- There are multiple experiments, the parameters are fixed, and the region is random.
- Interpretation: if the experiment is repeated many times, the different $C\%$ confidence regions will include the true parameters $C\%$ of the time.

Credible regions in Bayesian statistics provide an uncertainty estimate for the parameters θ under the posterior distribution $p(\theta|x)$. They are computed analogously, but now:

- There is one experiment, the parameters are random, and the region is fixed.
- Interpretation: the true parameters have a $C\%$ probability of being included in the $C\%$ credible region.

The *coverage probability* of a confidence region refers to the actual (i.e., empirically verified) proportion of the time it includes the true parameters.

- The specified confidence level $C\%$ is sometimes called the nominal coverage.

Likewise, the coverage probability of a credible region can be defined as the actual probability that the true parameters are included in the region.

- Note that this is now a frequentist notion, as we can only estimate the actual probability through aleatoric repetition of the experiment.

When does the coverage equal the nominal coverage for confidence/credible regions? In other words, when are confidence/credible regions "correct"?

For confidence regions, the assumptions that go into its construction must be correct — specifically, the assumed data distribution must match the actual data distribution.

- For example, the confidence interval of the normal mean is only guaranteed to have the correct coverage on normal data.

It's similar for credible regions, except the assumptions are now the likelihood and prior.

- The experiment is "repeated" by drawing parameters from the prior, drawing data from the corresponding likelihood, then computing the corresponding posterior.
- If this is the case, then the coverage of any credible region is also correct.
- If the parameters are not truly random (just unknown), then the prior describes epistemic uncertainty, and talking about coverage makes less sense to begin with.

As an example, let us return to the misguided "test of relativity" where π is allowed to vary in a gravitational-wave model, and estimated from actual gravitational-wave data.

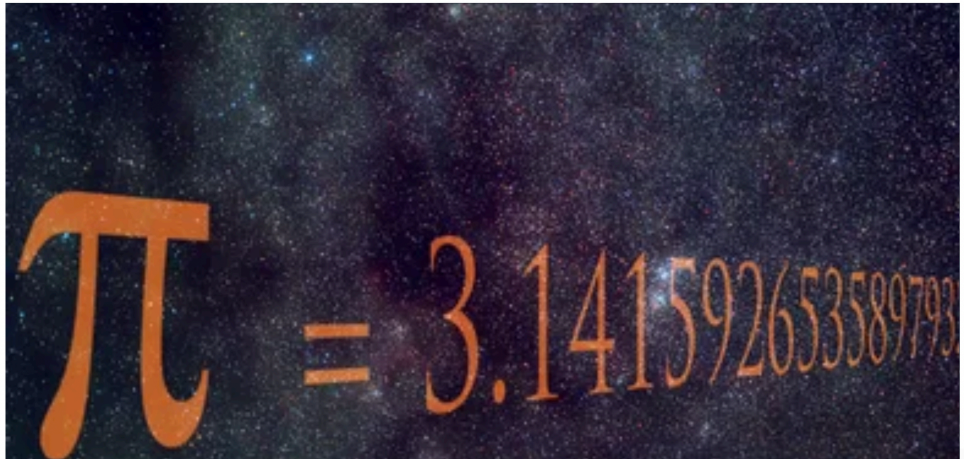
- The study computed a posterior for π that was "consistent" with the actual value. But suppose $p(\pi|x)$ was found to exclude the actual value at a high credible level, say $P(\pi \leq 3.14\ldots|x) > 99\%$. Would that imply a high-significance result?
- No, because coverage is not guaranteed. In the study, the prior on π was chosen as uniform on some large interval containing the actual value. That does not correspond to any realistic generation process for gravitational-wave data.

SPACE & PHYSICS

Pi in the Sky: General Relativity Passes the Ratio's Test

Using gravitational waves to approximate pi, physicists see no problem with Einstein's theory

By Daniel Garisto on May 19, 2020



Credit: Scientific American

Statistical tests

Null tests with p-values, where the aim is to accept or reject some point hypothesis $\theta = \theta_0$, are very popular in frequentist statistics.

They are also somewhat problematic:

- A lot of information is discarded by boiling the problem down to an accept/reject verdict
- The dichotomy between the null hypothesis and its alternative $\theta \neq \theta_0$ is often artificial, especially in continuous models
- They only address the probability of observing the data under null hypothesis, and not the probability of either hypothesis under the observation of the data

In the Bayesian approach, the best solution is to just report the full posterior (and summary statistics including probabilities), rather than give an accept/reject verdict.

Example

With observed data $x = 1$ and the normal model $\mathcal{N}(\theta, 1)$, test the hypothesis that $\theta = 0$. What does the Bayesian approach say?

- If that is still not concrete enough, the posterior odds or Bayes factor of $\theta = \theta_0$ over $\theta \neq \theta_0$ might be computed — but with care. In the above example, this cannot be done with an improper prior.

Equivalence tests are more general hypothesis tests that compute confidence regions and compare them to some equivalence bounds.

- The bounds might be defined by a hypothesis, or constructed from other data
- For one-sample tests, this is closer to Bayesian in that it moves away from point estimates, but it is still done with the data distribution instead of the posterior
- For two-sample tests, one Bayesian alternative is simply to compute two posteriors and assess their equivalence/difference

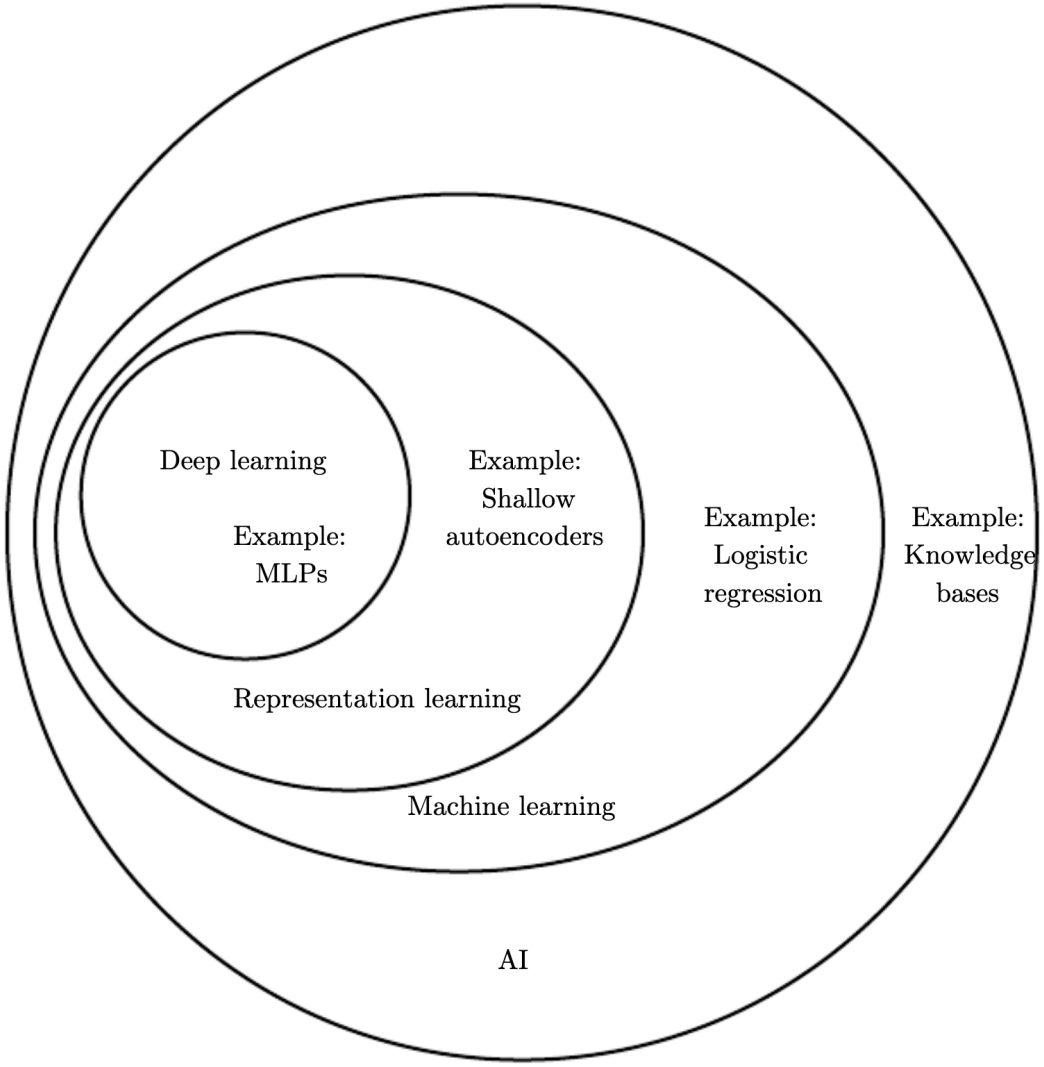
Frequentist statistical tests can also be extended to multilevel models, where the data is allowed to have a hierarchical or clustered structure, e.g., analysis-of-variance (ANOVA) tests that determine the equivalence/difference of multiple groups.

- Clearly, the Bayesian approach of hierarchical modelling is well suited to such problems!
- It's not a magic bullet, but the posterior for the hierarchical parameters provides a natural and "automatic" way of sorting/distinguishing/grouping data

12. Introduction to Machine Learning

Motivation and Background

A natural starting point is to nail down the definition of various buzzwords such as *artificial intelligence* and *deep learning*, as well as their relationship to machine learning.



Credit: Goodfellow et al., 2016

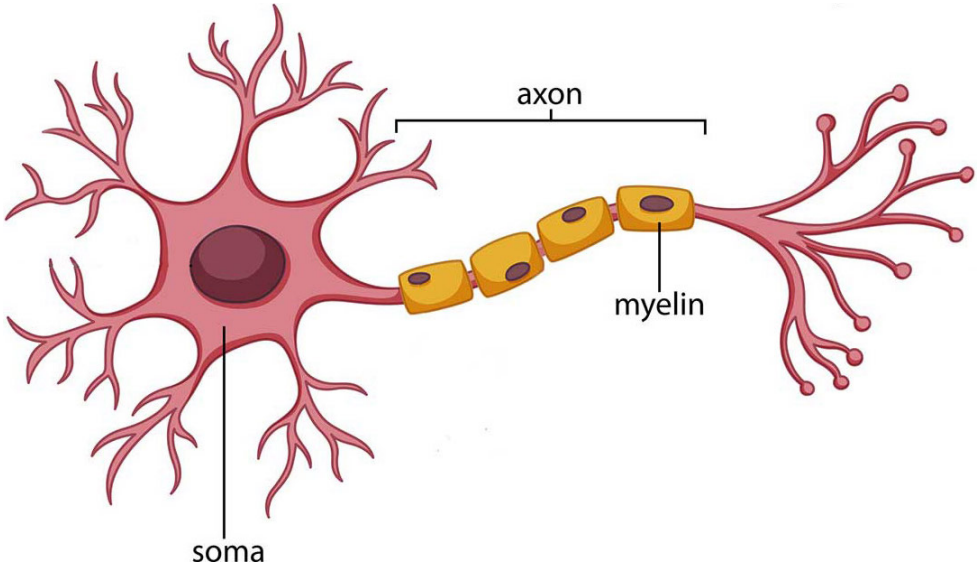
Artificial intelligence

- Not strictly a subfield of computer science, although it certainly seems that way
- Has roots and connections in neuroscience, logic, psychology and linguistics

The field dates back to the 1940s, with early mathematical models of biological learning such as the McCulloch–Pitts neuron:

$$y = \Theta\left(\sum_i w_i x_i\right), \tag{188}$$

where Θ is the Heaviside step function.



Credit: Johns Hopkins Medicine

Two main early motivations:

- Understand biological intelligence on an algorithmic level
- Reproduce/mimic biological intelligence using algorithms

The former is still an active field known as computational neuroscience, while the latter is the focus of modern artificial intelligence. Connections between the two fields are weaker now, because the former is much more difficult and cannot be used to guide the latter.

Artificial intelligence is still a little broader than what we refer to as machine learning, though. Knowledge-based systems are an example: these are computer programs that use a knowledge base (hard-coded information) and a reasoning system (logical rules) to mimic intelligence.

Machine learning

How we defined it at the start of the course: using statistics, algorithms and computers to learn about a problem from data. This is admittedly rather broad (and borderline tautological).

Some better definitions:

- A form of applied statistics with increased emphasis on estimation/prediction and decreased emphasis on errors/uncertainty
- A framework where the performance of a computer program on some class of tasks can be improved with additional "experience" (relevant but not comprehensive data)

Example tasks in machine learning

- Classification: $\mathbb{R}^n \to \{1, \ldots, k\}$ (set of classes) or $\mathbb{R}^n \to \{p \in [0,1]^k : \sum_i^k p_i = 1\}$ (categorical distribution over classes)
- Regression: $\mathbb{R}^n \to \mathbb{R}^m$
- Density estimation (for inference and prediction): forecasting, imputation, synthesis, etc.
- Clustering
- Dimensionality reduction
- Others: transcription and translation, anomaly detection, data cleaning (denoising), etc.

Types of experience

- Raw unlabelled data x (unsupervised learning)
- Data augmented with labels (x, y) (supervised learning)
- Interaction with data (reinforcement learning)

Types of performance metric

- Accuracy and error rates (avoid underfitting)
- Generalisability (avoid overfitting)

Representation learning

This refers to an important subclass of machine-learning methods, where the algorithm learns the appropriate representation or "features" of the data (either specifically, or as a by-product of learning a mapping from data to target output).

- Examples that are not necessarily deep learning: clustering, dimensionality reduction

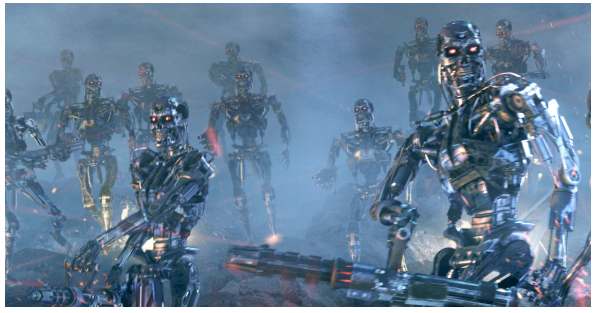
Deep learning

This is essentially hierarchical representation learning: the representations are expressed in terms of simpler representations, enabling a build-up to great complexity.

- Autoencoders do this explicitly, but any sort of multilayer neural network can also be viewed as learning representations (the outputs of individual layers)

Deep learning is so dominant these days that it is practically synonymous with artificial intelligence. But while its breakthroughs have been amazing, knowing how it works can give a better sense of what can and cannot be achieved under the current paradigms.

Winning at chess or go + deepfakes + ChatGPT $\not\to$ rise of the machines



Credit: ILM

Scope of this module

Any overview course on machine learning must necessarily be quite selective, because:

- There are far too many machine-learning tasks/methods to cover comprehensively.
- These can overlap in ways that make them difficult to categorise canonically.

This statement is twice as valid for half a course!

Thus we will focus only on some of the most (subjectively) important tasks/methods, and group them as sensibly as we can:

- Most of these can be thought of as "classical" machine learning — not in the sense of being outdated, but rather because they have become widely used as fundamental statistical/computational tools in science.
- We will not get too deep into deep learning — partly because the field has exploded over the past few decades, but mainly because its integration with science is more recent and currently still a frontier of research.

Such a treatment should hopefully give a good flavour of what machine-learning methods are available for typical scientific tasks, and provide a gateway to further self-exploration.

The Essence of Machine Learning

Virtually all methods that are used to address a given task in machine learning can be boiled down to the specification of four components:

- The data set
- The model
- The objective function, e.g., cost/loss/etc. $(-)$ or utility/profit/etc. $(+)$
- The optimisation procedure

Any of these components can be replaced largely independently from the others, which is why there are many possible methods for any particular task.

The presence of the first two components is obviously required. The last two often involve some explicit accuracy/error metric, but may be less obvious in certain contexts. For example:

- In clustering, the cost function could be the overall variance within clusters, or the number of cut edges in the partitioning of a graph
- In classification, the optimisation procedure for decision trees is quite specialised and may not be suitable for off-the-shelf optimisers

Reductionist view (redux): Isn't this just fitting a model to data?

- Yes, but machine learning is essentially the most general framework for doing so
- This unified view is useful for seeing connections among all the different methods

Other Key Concepts and Definitions

Supervised vs unsupervised learning

Generally, unsupervised learning involves learning the probability distribution of some data x (explicitly or otherwise). If x has an associated vector of labels y, supervised learning involves learning the conditional distribution $p(y|x)$ (again explicitly or otherwise).

The distinction between the two is not always clear-cut. For example:

- For $x \in \mathbb{R}^n$, we may treat the unsupervised learning of

$$p(x) = \prod_{i=1}^{n} p(x_i|x_1, \ldots, x_{i-1}) \tag{189}$$

  as $n-1$ supervised problems (plus a simpler $p(x_1)$). Likewise, the supervised learning of

$$p(y|x) = \frac{p(x,y)}{\int dy\, p(x,y)} \tag{190}$$

  might be performed by learning $p(x,y)$ with unsupervised techniques.
- When estimating the posterior density from samples, we are trying to learn the distribution $p(\{\theta_i\})$, where $\theta_i \sim p(\theta|x)$. But we also know the asymptotic posterior by definition, so this might be viewed as trying to predict $p(\theta|x)$. If we want to know it for general x, this can be done supervised (the data is $\{(x,\theta)_i\}$ and $p(\theta|x)$ is learnt).

Parametric vs non-parametric methods

"Non-parametric" is another common term. In statistical modelling, it can mean the lack of an assumed distribution or dependency on any parameter (e.g., order statistics). But more often, it broadly refers to methods with weaker assumptions on the model structure — the number and nature of parameters are flexible, and can depend or be learnt from the data.

- The normal approximation is parametric with (μ, σ)
- A histogram is non-parametric with flexible bin specifications
- Are neural networks parametric or non-parametric? It doesn't really matter

Training, validation and testing

A similar dichotomy is that between parameters and hyperparameters, which is a term that we first saw in hierarchical modelling. In the context of machine learning, hyperparameters refer to parameters that are tuned and not learnt, or at most "learnt" at a different level.

This is related to the concept of training, testing and validation data sets:

Training data	Used to learn parameters, i.e., train the model
Test data	Used to evaluate the *final* performance of the trained model
Validation data	Used to tune hyperparameters and evaluate interim performance

- Validation data is either completely distinct from training and test data, or sampled from the training data (cross-validation)
- Apart from hyperparameter tuning, validation data is also used to monitor for overfitting

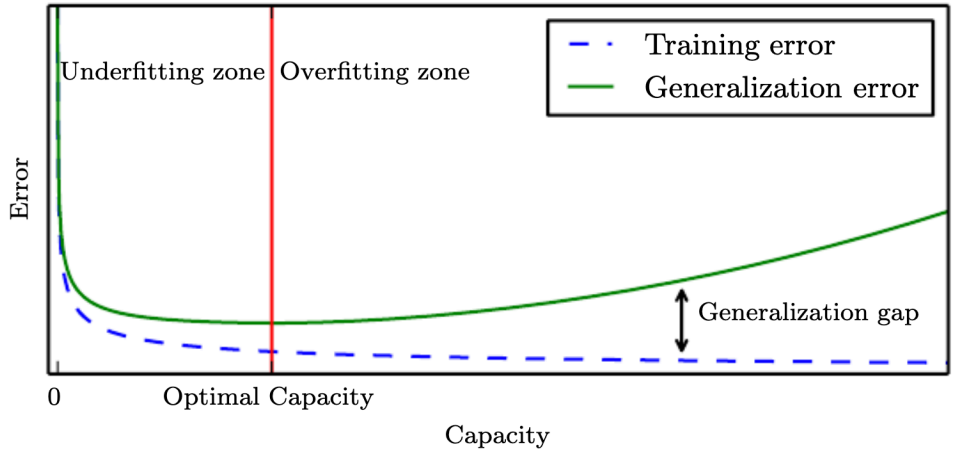
Capacity, underfitting and overfitting

A key aim in machine learning is that the model must be able to generalise well on data that has not been provided as experience (which is why knowledge-based systems are not considered "learning"). Generalisation relies on the assumption that the training and test data are iid.

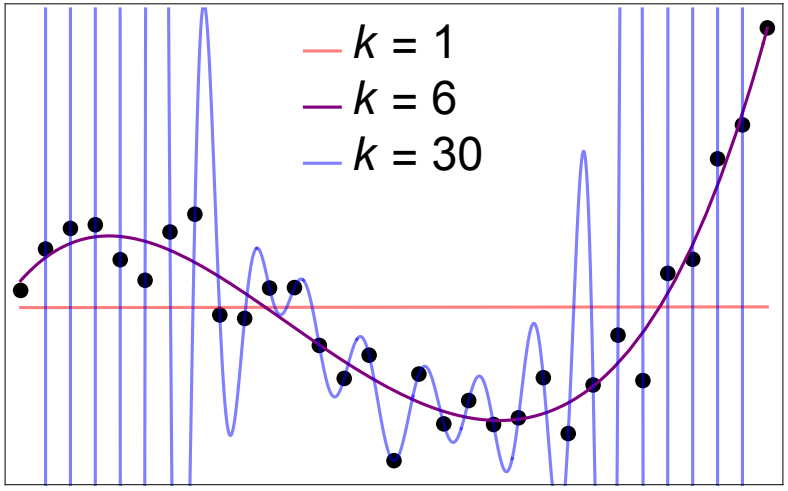
What if the test data does not have an identical or even similar distribution to the training data? In the language of function interpolation, this would correspond to extrapolation.

- Machine learning fails (generally). It may sometimes look like the most sophisticated deep-learning methods are capable of extrapolation, but I would argue it's still interpolation — we are just unable to grasp the "span" of the data in complex spaces.

The capacity of a machine-learning model is essentially the d.o.f. it has. If this is too low for a given task, the model will have a large average training error (underfitting). If it is too high, the model will have a large difference between the average test and training errors (overfitting).



Credit: Goodfellow et al., 2016



Generally, underfitting is associated with high bias and low variance in the model, while overfitting is associated with low bias and high variance. Bias and variance here can refer to both the parameters of the model as well as its output, since they are correlated.

In most machine-learning tasks, a central concern is *regularisation*: the reduction of overfitting by lowering test error but not training error. This is usually done by modifying the objective function to encourage simplicity in the model (low variance).

- Overfitting gets all the hype, but underfitting can be a bigger problem in some contexts

No free lunch

The no-free-lunch theorems by Wolpert and Macready for machine-learning and optimisation algorithms state that:

> Any two algorithms have an equivalent average performance over all possible problems.

This is related to the iid assumption from above. Essentially, even the most sophisticated machine-learning algorithm does no better than blind chance if all possible data-generating distributions are considered. If it performs well on a class of tasks, it necessarily pays for that elsewhere. In other words, machine learning is task-specific and not a universal magic box!

Curse of dimensionality

Machine learning is also challenged when the dimensionality of the data is high — basically the number of possible configurations for the data increases exponentially, and so too the amount of data that is needed for accurate generalisation.

- It's similar to why high-dimensional sampling is hard: recall the n-ball.
- The push towards deep learning was largely motivated by the curse of dimensionality.
- The term is also often used more generally to refer to other (non-learning) difficulties arising from large data sets or high-dimensional parameter spaces.

On Data Representations

Some machine-learning methods address the standalone task of learning the appropriate representation of data, i.e., without doing anything more. The most common examples are clustering (which we will touch on in the next section) and dimensionality reduction.

Dimensionality reduction is essentially the transformation of data $x \in \mathbb{R}^n$ to some reduced representation $z \in \mathbb{R}^m$, where $m < n$ can be learnt or defined.

- The raw dimensionality n can be quite high for realistic data, so the curse of dimensionality is the main motivation for dimensionality reduction.
- The reduced representation should retain most of the information in the raw data. Thus dimensionality reduction connects to the idea of data compression (which can sometimes be lossless, but is usually lossy).
- Data often has some intrinsic dimensionality d, e.g., a set of points that are constrained to lie on a curve in $\mathbb{R}^2$ has an intrinsic dimensionality of one, even though it has an overt two-dimensional representation. Dimensionality-reduction methods can allow d to be estimated, so as to choose $m \approx d$.

Dimensionality reduction is an example of unsupervised representation learning. It can done with both parametric and non-parametric methods.

- It can be viewed as a regression problem as well (but so can many other things)

Principal component analysis (PCA)

The most common and important example of a dimensionality-reduction method is that of PCA. Its underlying idea is very straightforward: the empirical distribution of the data will generally exhibit a subset of dominant directions in the data space, and the reduced representation is simply the projection of the data onto those directions.

Consider a data set with N examples $x_i \in \mathbb{R}^n$. For convenience, let us first translate $\{x_i\}$ to centre it on the sample mean $(1/N)\sum_i x_i$, which we may always do.

Next we write the data as an $N \times n$ matrix $X := [x_{ij}]$, where j indexes the n components of x_i. This is sometimes called a *design matrix*, although the same term is used in other contexts with different meanings. The sample covariance matrix is then

$$S := \frac{1}{N}\sum_{i}^{N} x_i x_i^T \equiv \frac{1}{N} X^T X. \tag{191}$$

- Note that some algorithms use the unbiased estimate of covariance (dividing by $N-1$).

The eigenvectors of S are called principal components, and the associated eigenvalues provide a natural way of ordering the components and determining which ones to retain.

Algorithm

1. Compute the sample mean and the centered data X
2. Compute the sample covariance S and its eigenvalues/eigenvectors
3. Determine the eigenvectors $u_1, \ldots, u_m$ with the m largest eigenvalues, and form the $n \times m$ matrix $U = [u_1 \cdots u_m]$
4. Compute the reduced representation as the $N \times m$ matrix $Z := XU$

- PCA was first introduced by Karl Pearson in 1901
- It has been reinvented under many different names such as the Hotelling transform; the Karhunen–Loève transform (signal processing, dynamical systems); proper orthogonal decomposition (numerical analysis, fluid dynamics); etc.
- PCA is also closely related to singular value decomposition (SVD), which generalises eigendecomposition for rectangular matrices; taking the SVD of X is now the standard way of doing PCA, as it directly gives Z without having to compute S

If desired, the data can be reconstructed in the original data space $\mathbb{R}^n$ as

$$\tilde{X} := ZU^T. \tag{192}$$

We have $\tilde{X} = X$ exactly for $m = n$, since U is an orthogonal matrix (and since standard PCA is deterministic). More generally, the reconstruction will be lossy.

In addition to dimensionality reduction, PCA can also be used as a whitening (or sphering) transform, such that the transformed data is uncorrelated and normalised, i.e., its sample covariance is the identity matrix. The transform is given by

$$W := \Lambda^{-1/2} U^T, \quad \hat{X} := X W^T, \tag{193}$$

where $m = n$ and Λ denotes the diagonal eigenvalue matrix.

Exercise

Show that $\hat{S} := (1/N)\hat{X}^T \hat{X} = I$.

More generally, any matrix W that satisfies $W^T W = S$ can also be used for whitening; e.g., $W = S^{1/2}$ (Mahalanobis whitening) or $W = L^T$ where $LL^T = S$ (Cholesky whitening).

Example

Jupyter notebook: The iris flower data set is a classic multivariate data set studied by Ronald Fisher in the 1930s. It comprises four measurements of 150 flowers from three species of iris: sepal* length, sepal width, petal length and petal width (in cm). Apply PCA to this data.

Credit: Danielle Langlois

*Sepals are actually bigger than petals in irises

A final question you might have: Why is PCA considered machine learning? Specifically, where is the objective function and optimisation procedure?

- What is being maximised is the variance of the data after projection onto some set of $m \leq n$ vectors. This also minimises bias (the error incurred by the projection).
- The optimisation is baked in — it can be shown that both the full and partial eigensystems of S are the optimal set of vectors for each value of m.

Reduced-basis methods

Another class of dimensionality-reduction methods warrants a mention here. Reduced-basis methods arise in a completely non-statistical context: that of model-order reduction in numerical analysis. But they have the same end product as PCA — a smaller orthonormal set of vectors in the data space that is determined by the data itself.

- Such methods are not standard content at all in machine-learning courses
- Still, the algorithm that is used to obtain the basis feels more explicitly like learning!

The reduced basis is constructed in an iterative fashion by searching through the data set sequentially with a *greedy algorithm* ("greedy" because it makes the locally optimal choice at each step). Each new vector that is added to the basis minimises the projection error at that step, and the stopping criterion is some explicitly defined error tolerance.

- One can alter the standard PCA algorithm to have a similar error-based criterion for choosing m, but the errors are more central in reduced-basis methods. They are monitored along the way, and even determine the basis itself via the greedy algorithm.
- The reduced basis itself depends on the choice of starting point — but for a sufficiently informative data set, its size does not vary too much. Thus the method is essentially learning the intrinsic dimensionality of the data.
- Reduced-basis methods are near-optimal but more efficient than PCA for large N, n.

Finally, both PCA and reduced-basis methods are built out of linear transformations. Is there a generalisation to nonlinear transformations? Yes: such methods are known as *autoencoders*.

13. Classification

Binary Classifiers

Classification is the supervised learning of a map from data $x \in \mathbb{R}^n$ to its target class/label $y \in \{1, \ldots, k\}$, or to a probability mass function $p(y|x)$. The PMF has k components that sum to one, i.e., its range is the probability simplex $\{p \in [0,1]^k : \sum_i^k p_i = 1\}$.

- Strictly speaking, the statistical task of classification really only refers to the former. But we will focus mainly on the latter, since it is more general and allows classification to be performed by selecting the class with the largest PMF value.

Binary classification is the case of $k = 2$. It is one of the most common classification tasks, since dichotomies are everywhere: success/failure, true/false, statement/negation, etc.

For any given training example x, we also have its associated target $y = 1$ or $y = 2$. This can be represented in what is known as *one-hot* form: $y = (1, 0)$ or $y = (0, 1)$.

- For k classes, the form comprises k bits, of which only one is "hot" (nonzero)
- The one-hot form also has a convenient interpretation here as the "true" probability mass function $p(y|x)$, if y is regarded as a random variable

The aim then is to construct a model $\hat{y}(x)$ that outputs a vector on the probability 2-simplex, such that $\hat{y}(x) \approx y$ in one-hot form (for all training examples). This is known as a soft classifier, as opposed to a hard classifier that restricts its output to one-hot form.

- Soft classifiers perform better when the classes are less distinct, or when the underlying data generation process is smooth. This is often the case in realistic problems.
- Soft classifiers are also more informative (not just k bits).

Logistic regression

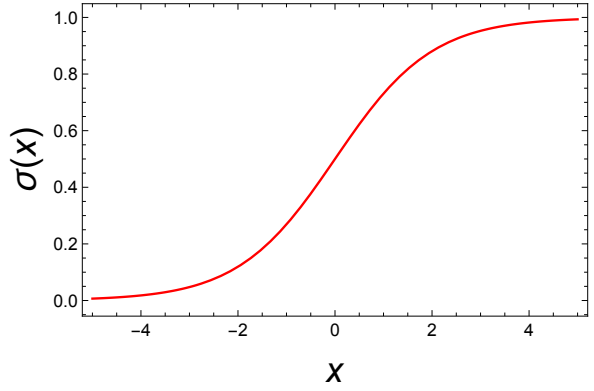
First consider an arbitrary linear (or really affine) transformation $x \mapsto Wx + b$, which is a natural starting point and building block in constructing more complex maps.

- We restrict to a scalar output here, and it is always possible to fold b into W by augmenting x with constant unit components, so: $x \mapsto w \cdot x$

The output of this transformation is unconstrained on $\mathbb{R}$, but we would like to map into $[0, 1]$ instead when describing probabilities. In logistic regression, this is done via the logistic function

$$\sigma(x) := \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x} = \frac{1}{2}\left(1 + \tanh\left(\frac{x}{2}\right)\right). \tag{194}$$

- This is aka the expit function; it is the inverse of the logit function from Eq. (12), so logistic regression is also referred to as logit regression or the logit model.
- The logistic function is the most well-known example of a *sigmoid* function, and so the two terms are often used interchangeably.



Exercise

Show that $\sigma(-x) = 1 - \sigma(x)$.

Putting it all together, the logit model is

$$\hat{y}(x) := (\sigma(w \cdot x), \sigma(-w \cdot x)), \tag{195}$$

or simply $\hat{y}(x) \equiv P(y = (1, 0)|x) = \sigma(w \cdot x)$, which contains all the required information.

We now need an objective function to optimise (over the model parameters w). Recall that the Kullback–Liebler divergence in Eq. (166) describes the discrepancy between two probability distributions. We want our model $\hat{y}$ to approximate y, so let us minimise

$$D_{\mathrm{KL}}(y, \hat{y}) = \sum_i^k y_i \ln\left(\frac{y_i}{\hat{y}_i}\right) = -\sum_i^k y_i \ln \hat{y}_i + \text{const.}, \tag{196}$$

where the y_i-only term does not depend on w. Since $k = 2$, the sum in the RHS evaluates to

$$L(y, \hat{y}) := -(y_1 \ln \hat{y}_1 + y_2 \ln \hat{y}_2) = -(y_1 \ln \sigma(w \cdot x) + (1 - y_1) \ln \sigma(-w \cdot x)). \tag{197}$$

The function $L(y, \hat{y})$ is known as the cross-entropy loss for classification (extending to general k as well). It is minimised over the model parameters w for all training examples — i.e., the actual quantity that is minimised is its average or sum over the training set.

Exercise

Show that $L(y, \hat{y})$ is minimised when $y = \hat{y}$.

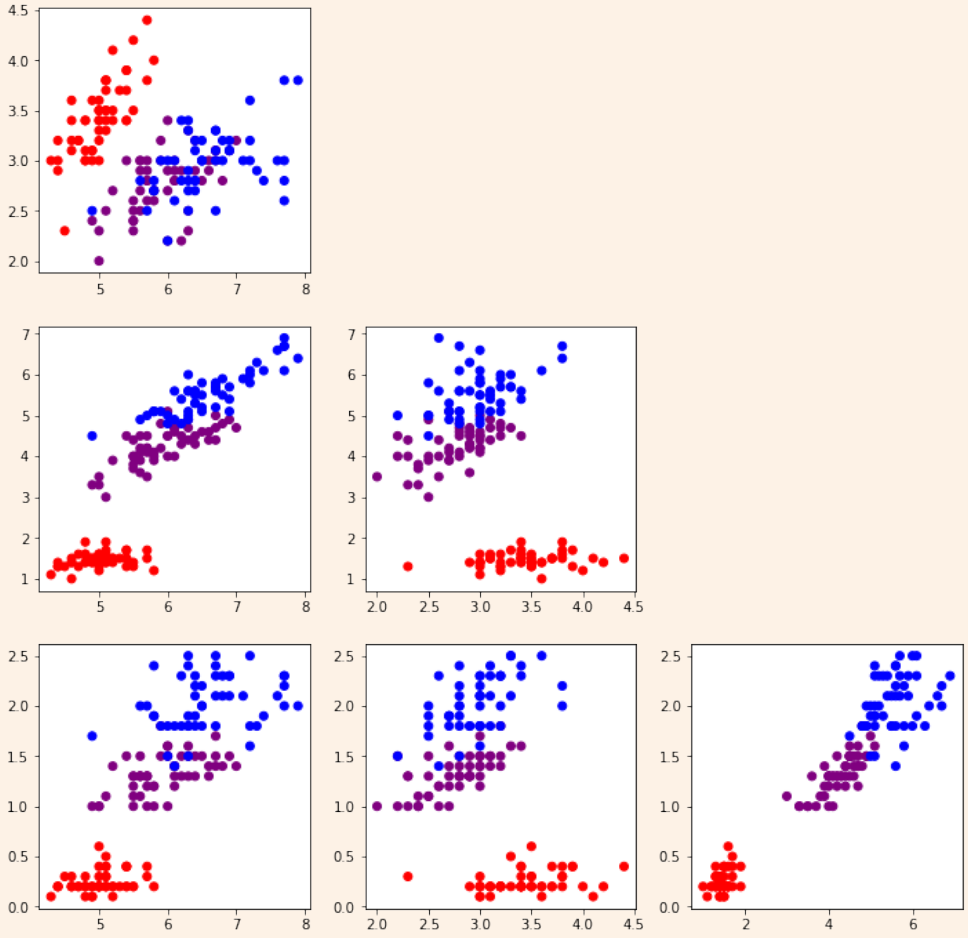
This is a transcendental equation in w, so it's as far as we can go analytically. Numerical optimisation is required, using either general algorithms or more tailored ones.

Practicalities (also a preview for more general machine-learning methods):

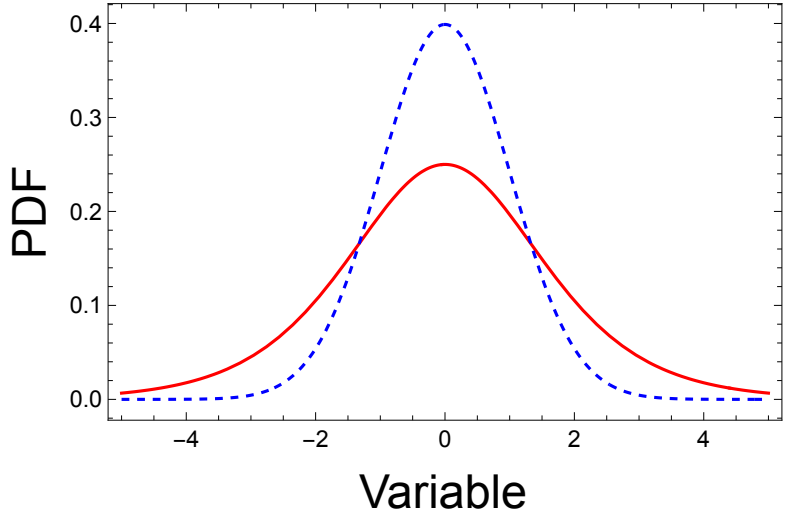
- Transforming the data to location ≈ 0 and scale ~ 1 is common and can help
- Randomly shuffling the data is also common (but not relevant here)
- The bias/intercept term b is often restored for more capacity
- A regularisation term can be added to the loss, to reduce variance in the weights w
- Optimisation with random start points is often needed to deal with multimodal loss functions (although the binary cross-entropy loss here is convex)

Example

Jupyter notebook: Apply logistic regression to the iris flower data set, and construct three binary classifiers that predict whether any particular iris flower (given its sepal and petal measurements) is of a certain species (setosa/versicolour/virginica).



The logistic function is also the cumulative distribution function of a probability distribution known as the logistic distribution (with location zero and scale one). This looks like a normal distribution with heavier tails:



Thus the logit model $P(y = (1, 0)|x) = \sigma(w \cdot x)$ is equivalent to

$$P(\epsilon \leq w \cdot x) = P(w \cdot x - \epsilon \geq 0), \tag{198}$$

where $\epsilon \sim \text{Logistic}(0, 1)$. Alternatively for $P(y = (0, 1)|x) = \sigma(-w \cdot x)$:

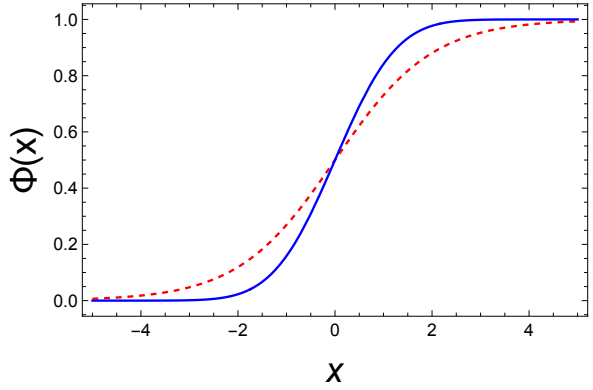
$$P(\epsilon \leq -w \cdot x) = P(w \cdot x + \epsilon \leq 0). \tag{199}$$

In this picture, logistic regression can be viewed as: using a linear model $w \cdot x$ with logistically distributed errors ϵ to classify x, based on the probability that $w \cdot x - \epsilon$ is greater than zero.

Other linear binary classifiers

The above picture allows us to generalise to other "linear" models $f(w \cdot x)$ with different error distributions. If the normal CDF Φ is used in place of σ, this yields the *probit model*, which is a linear model with normally distributed errors.

- There is no a priori reason to say one model is more "correct" than another (unless data generation explicitly uses a known model), and the results they yield are often quite similar
- The logit model is generally less sensitive to outliers and thus more robust, due to the heavier tails of the logistic distribution



In machine learning, f is called an *activation* function, while its inverse is known as a *link* function in the statistics literature, e.g., a logistic activation is equivalent to a logit link/model.

- In artificial neural networks, individual neurons are precisely the composition of an activation function with a linear transformation

Validating binary classifiers

To assess the performance of a binary classifier, we typically examine its so-called *confusion matrix* and various derived quantities (with multiple names for some quantities).

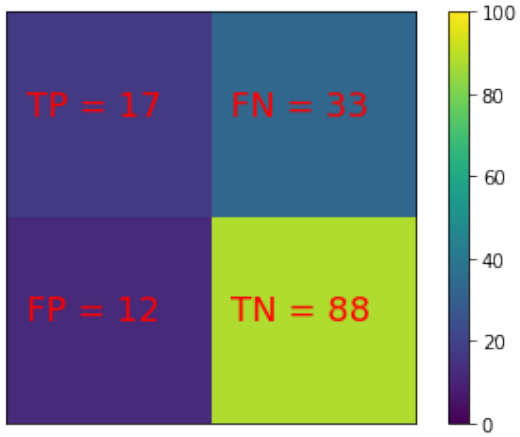
Some presentations can be quite confusing (thus the name, maybe) but it may help to know up front that everything is fully specified by just four numbers: size of data set, number of actual positives, number of predicted positives, and number of true (correctly predicted) positives.

The essential quantities:

- S: Size of data set
- P: Number of actual positives
- N: Number of actual negatives
- $\hat{\mathrm{P}}$: Number of predicted positives
- $\hat{\mathrm{N}}$: Number of predicted negatives
- TP: Number of true positives (aka hits)
- FP: Number of false positives (aka Type I errors, false alarms)
- FN: Number of false negatives (aka Type II errors, misses)
- TN: Number of true negatives

The confusion matrix refers to

$$\begin{pmatrix} \mathrm{TP} & \mathrm{FN} \\ \mathrm{FP} & \mathrm{TN} \end{pmatrix} \tag{200}$$



Important derived quantities:

P/S: Prevalence (of actual positives)

$(\mathrm{TP}+\mathrm{TN})/\mathrm{S}$: Accuracy

$\mathrm{TPR}=\mathrm{TP}/\mathrm{P}$: True positive rate (aka recall, sensitivity, detection probability, hit rate)

$\mathrm{FPR}=\mathrm{FP}/\mathrm{N}$: False positive rate (aka Type-I-error rate, false-alarm probability)

$\mathrm{FNR}=\mathrm{FN}/\mathrm{P}$: False negative rate (aka Type-II-error rate, miss rate)

$\mathrm{TNR}=\mathrm{TN}/\mathrm{N}$: True negative rate (aka specificity)

$\mathrm{PPV}=\mathrm{TP}/\hat{\mathrm{P}}$: Positive predictive value (aka precision)

$\mathrm{FDR}=\mathrm{FP}/\hat{\mathrm{P}}$: False discovery rate

$\mathrm{FOR}=\mathrm{FN}/\hat{\mathrm{N}}$: False omission rate

$\mathrm{NPV}=\mathrm{TN}/\hat{\mathrm{N}}$: Negative predictive value

The true/false positive/negative rates are properties of the classifier and not the data. They are independent from the prevalence, which is a property of the data and not the classifier.

Quantities like the accuracy and the predictive values are prevalence-dependent. Careless application of statistics can often lead to the *base-rate fallacy*, in which prevalence-dependent conclusions are drawn without considering the prevalence.

Example

If you test negative on a particular Covid test with $\mathrm{TPR} = 0.7$ and $\mathrm{TNR} = 0.99$, what is the probability that you are actually positive if the prevalence is i) 0.2; ii) 0.8?

The confusion matrix and its derived quantities do not fully describe the performance of a binary classifier that learns the PMF, because the threshold probability value that is used to distinguish the two classes is arbitrary.

- For example, a classifier that consistently assigns probability values $P(y = 1) \approx 0.5$ to examples with $y = 1$ and values $P(y = 1) \approx 0.25$ to examples with $y = 2$ is actually performing much better than one would assess it to be with a fixed threshold of 0.5.

This is addressed by examining the *receiver operating characteristic* (ROC) of the classifier.

- Developed during WWII to measure a radar receiver operator's ability to correctly detect enemy aircraft from their radar signals
- Consider a strictly monotonic sequence of threshold values, each of which gives a corresponding TPR and FPR
- The ROC is simply the plot of TPR against FPR on the unit square
- The ROC can be summarised by the area under it: a value of 1 corresponds to perfect performance, while a value of 0.5 corresponds to no discrimination (pure chance)

Example

Jupyter notebook: Assess the validity of the three logit classifiers for the iris flower data set.

Multiclass Classifiers

The logit model for binary classification in Eq. (195) can be written more explicitly as

$$\hat{y}(x) = \left(\frac{e^{w_1' \cdot x}}{1 + e^{w_1' \cdot x}}, \frac{e^{w_2' \cdot x}}{1 + e^{w_2' \cdot x}} \right), \tag{201}$$

where $w_2' = -w_1'$.

Exercise

Show with a suitable transformation $(w_1', w_2') \to (w_1, w_2)$ that

$$\hat{y}(x) = \left(\frac{e^{w_1 \cdot x}}{e^{w_1 \cdot x} + e^{w_2 \cdot x}}, \frac{e^{w_2 \cdot x}}{e^{w_1 \cdot x} + e^{w_2 \cdot x}} \right). \tag{202}$$

From the form of Eq. (202), it is clear how to generalise the model to its multiclass variant:

$$\hat{y}(x) := \left(\frac{e^{w_1 \cdot x}}{\sum_i^k e^{w_i \cdot x}}, \ldots, \frac{e^{w_k \cdot x}}{\sum_i^k e^{w_i \cdot x}} \right), \tag{203}$$

where the k components sum to one by definition. This vector-valued function is known as the *softmax* function, and is so named because it is a smooth approximation to the argmax function (whose output is "hard" as it is in one-hot form):

$$\operatorname{argmax}(w_1 \cdot x, \ldots, w_k \cdot x) = (0, \ldots, 0, 1, 0, \ldots, 0), \tag{204}$$

where the index of the single nonzero component is that of the largest $w_i \cdot x$.

- The softmax function is often denoted by σ as well, i.e., $\hat{y}(x) = \sigma(w_1 \cdot x, \ldots, w_k \cdot x)$.
- As an example, $\sigma(-1, 2, 3) = (0.01, 0.27, 0.72) \approx (0, 0, 1) = \operatorname{argmax}(-1, 2, 3)$.

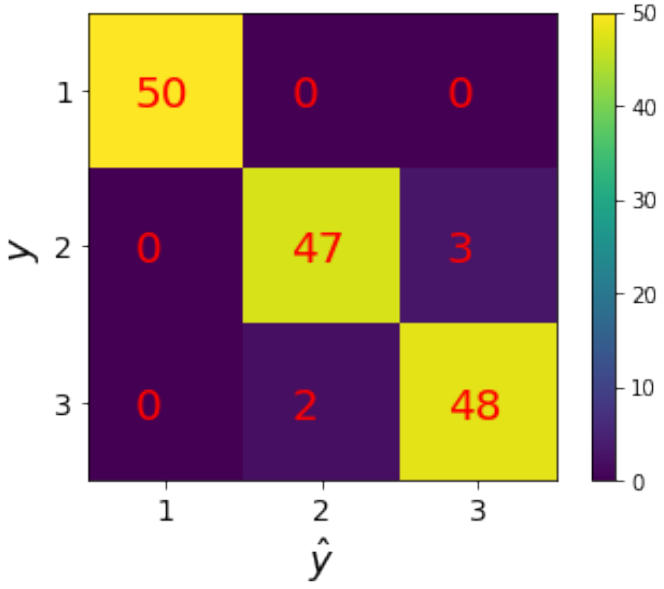
Finally, the cross-entropy loss function for multiclass classification is given via Eq. (196) as

$$L(y, \hat{y}) := -\sum_i^k y_i \ln \hat{y}_i. \tag{205}$$

Note that there are now kn weights to optimise over (or $k(n+1)$ if a bias was included). However, there are only $(k-1)n$ d.o.f., so the softmax formulation is underconstrained — the model is invariant under the addition of the same arbitrary constant to each $w_i \cdot x$.

The output of a k-class classifier can be viewed as solutions to k binary classification problems (or $k-1$ independent ones), so its performance can in principle be assessed by all of the quantities that are used to validate binary classifiers.

- In practice this is overkill, and we may usually just examine the overall accuracy (fraction of predictions that are correct) or the generalised confusion matrix
- The output itself is a point on the $(k-1)$-dimensional probability simplex, so is somewhat more troublesome to visualise in bulk when $k > 3$



Example

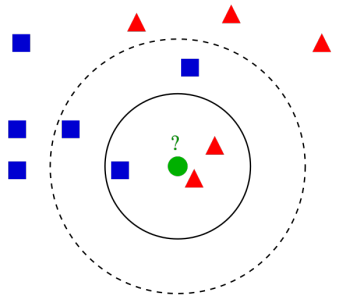
Jupyter notebook: Apply multiclass logistic regression to the iris flower data set.

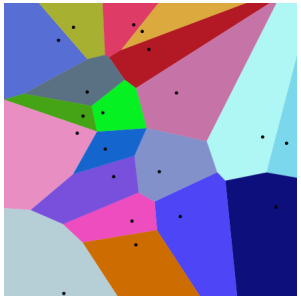
Other Common Classifiers

k nearest neighbours

Given a labelled training set $\{(x, y)_i\}$ and some metric on the data space (e.g., Euclidean for $x \in \mathbb{R}^n$), the k-nearest-neighbour algorithm simply classifies x under the most common class of the k training examples that are nearest to x. Ties are broken randomly or with different k.

- When $k = 1$, it is known as the nearest-neighbour algorithm, and is essentially a partition of the data space known as a *Voronoi diagram*
- Voronoi diagrams are generalisable in principle to $k > 1$, but these are nontrivial to compute and are seldom visualised directly





The standard version of k nearest neighbours doesn't really have parameters that are learnt.

- Can learn k (or more generally, weights for the entire training set) as well as the metric, but these extensions fall under *kernel methods*
- Can also be used for regression (take the average of k continuous targets)

Pros and cons:

- Very simple to interpret and implement
- Scales poorly with dimensionality and size of training set

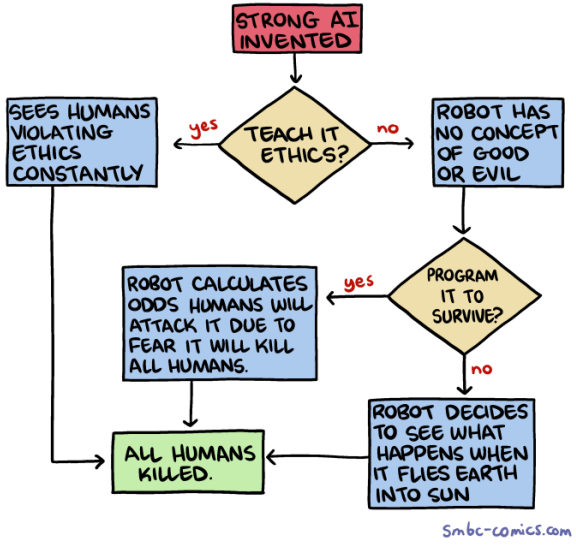
Decision trees

Decision trees follow the same principle as k nearest neighbours of partitioning the data space, and assigning the same output to points in the same region (equivalence class). The main difference is that the partition is built more flexibly in decision trees, by way of recursive partitioning — essentially, a flowchart that maps to the target space.

- Each split is usually (but not necessarily) binary; different paths can have same terminus
- Many extensions, e.g., random forests (essentially cross-validation for decision trees)
- Can also be used for regression (piecewise-constant function on a partition of the continuous target space)

Pros and cons:

- Interpretable like k nearest neighbours, but with more capacity (thus prone to overfitting)
- Training involves optimisation over space of all partitions! Requires specialised methods to train, and new data requires training from scratch



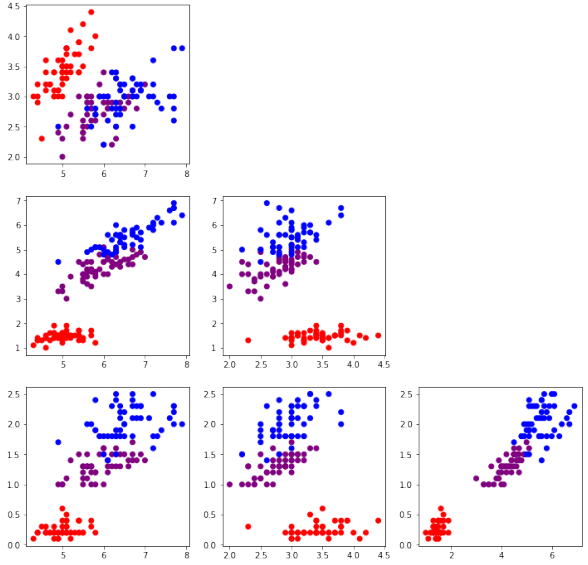
Classification vs Clustering

There is much conceptual overlap between the tasks of classification and clustering. For example, when we constructed a multiclass classifier for the iris flower data set, the end product was also an explicit grouping of the data into three clusters (in this case, associated with the three known classes). However, their main distinction can be summarised as:

Classification is the supervised learning of how to assign data to a set of given classes, while clustering is the unsupervised learning of what the classes are for a given data set.

This gives rise to significant practical differences in the algorithms for each task:

- There is no explicit training in clustering, and no need for validation in the usual sense
- Clustering is more "open-ended", so its results can be less precise or robust



The majority of the most important clustering algorithms fall under three broad classes:

Centroid These methods cluster data points based on their distance from a set of centroid (mean) points, which do not have to belong to the data set and can be learnt. Crucially, however, such methods generally require the number of clusters to be specified.

Density These methods cluster data points based on how densely they are distributed (with respect to some distance metric). They can learn the number of clusters, but the density-based criterion that is used as a cutoff definition for a cluster can be arbitrary. Examples include DBSCAN and OPTICS.

Connectivity These methods cluster data points based on how connected they are to one another (again with respect to some metric). They are generally quite powerful as they can be used to construct a hierarchy of clusters (i.e., at different scales), but are also more computationally expensive.

We will largely neglect clustering in the machine-learning part of this course — indeed, the overall focus is mostly on supervised-learning methods. However, we will briefly cover one of the simplest and most common algorithms: k-means clustering.

k-means clustering

This algorithm is the archetypal example of centroid-based clustering, and many other such methods are simply extensions of k-means. The idea is again very straightforward: to find a partition of the data into k subsets, such that the overall variance of all subsets is minimised. These subsets are then fully specified by their respective means.

- The algorithm is not to be confused with k nearest neighbours, although there are conceptual similarities in the sense of partitioning the data space. For example, the clusters for a set of k means agree precisely with the Voronoi diagram for the nearest-neighbour algorithm ($k = 1$) if it is trained on that set of points.

Formally, we may define the loss function for a partition $\mathcal{P} = \{S_1, \ldots, S_k\}$ of the data set S as

$$L(\mathcal{P}) := \sum_{i}^{k} \sum_{x \in S_i} |x - m_i|^2 = \sum_{i}^{k} |S_i| \mathrm{Var}[S_i], \tag{206}$$

where the m_i are the means of $x \in S_i$. (We abuse the notation $|\cdot|$ here to denote both the L2 norm of a vector and the cardinality of a set.) The aim in k-means clustering is to minimise $L(\mathcal{P})$ over all partitions, which again is generally difficult to do both accurately and efficiently.

However, there are simple practical algorithms that can achieve this optimisation approximately. The standard version is known as naive k-means, or Lloyd's algorithm.

Algorithm

1. Choose a starting set of k means $m_1, \ldots, m_k$
2. At each iteration, determine the mean that is nearest to each data point (typically with respect to the Euclidean metric)
3. Partition the data into k subsets $S_1, \ldots, S_k$ with the same nearest mean
4. Compute the means of each subset for the next iteration

Note that some subsets might be empty at Step 3, if the initial means are not chosen appropriately. One possible solution is simply to not update the corresponding mean.

Pros and cons:

- Very simple to interpret and implement; very widely used
- Large variety of more robust and/or efficient extensions
- Naive k-means can be quite sensitive to choice of initial means; may not converge
- A big limitation of many centroid-based methods is the need to specify k; however, one can consider multiple k, or extensions that learn k

Example

Jupyter notebook: Apply k-means clustering to the iris flower data set.

14. Regression

Regression vs Classification

Like classification, regression also involves the construction of a map from data x to target y.

- Classification: $\mathbb{R}^n \to \{1, \ldots, k\}$ (or $\mathbb{R}^n \to \{p \in [0,1]^k : \sum_i^k p_i = 1\}$)
- Regression: $\mathbb{R}^n \to \mathbb{R}^m$

So is regression just classification with a continuous codomain? Not quite. Both are statistical tasks, and actually describe the conditional distribution of y given x.

- In classification, $y|x$ is modelled as a Bernoulli/categorical random variable
- In regression, $y|x$ obeys some continuous distribution (e.g., normal in linear models)

It is this probabilistic description that allows us to directly map to the PMF of $y|x \in \{1, \ldots, k\}$ in classification. Can we extend this to learning the PDF of $y|x \in \mathbb{R}^m$ in regression?

- Not with naive fine-graining! $k \not\to \infty$
- But yes, because the conditional distribution is already specified by the model, although its representation can generally be implicit and/or non-parametric
- For example, most regression neural networks simply give the conditional mean $\mathrm{E}[y|x]$, but can be modified to output $p(y|x)$ instead
- This connects to inference (of $y|x$) and density estimation (of $p(y|x)$)

As an aside: what is the distinction between regression and *interpolation*?

- Interpolation can be viewed as a specific application of regression
- Regression is probabilistic while interpolation can be non-probabilistic, e.g., cubic splines

Linear Models

We will focus our study of regression on linear* models and their numerous generalisations.

Linear Least Squares

In *ordinary least squares*, each observed/training data point x is an n-vector whose components are called *regressors*. The corresponding observed/training target y is scalar-valued ($m = 1$), and is called the *response* when viewed as $y(x)$ or $y|x$.

The regression model in ordinary least squares is simply

$$\hat{y}(x) := w \cdot x, \tag{207}$$

where the components of the vector w are the weights/parameters of the model. We include a bias term by augmenting $x \to x \oplus 1$, such that w has $d := (n + 1)$ components.

- Note that the standard literature on regression uses the notation $\beta \equiv w$.

The model describes an n-dimensional hyperplane in $\mathbb{R}^n \times \mathbb{R}$ with respect to the components of x. When $n = 1$ this is just a line, and is referred to as *simple linear regression*. For general $n > 1$, the method is known as *multiple linear regression*.

*Unfortunately the term "linear" is so overloaded that it is on the verge of losing all meaning, as you will see.

For a training set $\{(x, y)_i\}$ of size $N \geq d$, we may form the $N \times d$ design matrix

$$X := \begin{pmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_N & 1 \end{pmatrix}, \tag{208}$$

and write the N-vector output of the model as

$$\hat{Y}(X) := Xw. \tag{209}$$

We want to choose w such that $\hat{Y}$ approximates the N-vector of targets Y. In ordinary least squares, this is done by minimising the residual sum of squares (hence "least squares"):

$$L(Y, \hat{Y}) := |Y - \hat{Y}|^2. \tag{210}$$

This is also commonly known as the L2 or squared-error loss in machine learning. Its minimisation in this case has a simple analytical solution, because the model is actually linear.

Exercise

Show that $L(Y, \hat{Y})$ is minimised when

$$w = (X^T X)^{-1} X^T Y. \tag{211}$$

The *Gram matrix* $X^T X$ is invertible when X has full rank, i.e., when its columns are linearly independent (since $N \geq d$). Equivalently, regressors should not be perfectly correlated. Even if X has full rank, $X^T X$ may still be ill-conditioned, but data transformations can help.

Ordinary least squares assumes the residuals/errors $y_i - \hat{y}_i$ are iid normal random variables (with zero mean). In *weighted least squares*, the errors remain independent but are allowed to have different variances — this is referred to as heteroscedasticity. In *generalised least squares*, the errors have a fully general covariance matrix Σ (which must be specified).

The upshot is that the general solution for linear least squares can be written as

$$w = (X^T W X)^{-1} X^T W Y \tag{212}$$

where $W := \Sigma^{-1}$ is either proportional to I, generally diagonal, or fully general.

There are two common summary statistics that can be used to perform goodness-of-fit tests for linear least squares (and far more generally as well).

Coefficient of determination:

$$R^2 := 1 - \frac{|Y - \hat{Y}|^2}{|Y - \bar{Y}|^2}, \tag{213}$$

where $\bar{Y}$ is the sample mean of Y.

- This can be interpreted as the proportion of the variation in Y that can be predicted by its relationship with X. If $\hat{Y} = Y$, then $R^2 = 1$ and the fit is exact. If $\hat{Y} = \bar{Y}$, then $R^2 = 0$ and the fit is bad. Note that R^2 can be negative!
- It is less meaningful for weighted and generalised least squares, but analogue quantities can be defined with transformations based on W.

Reduced chi-squared statistic (not defined for $N = d$):

$$\chi^2_\nu := \frac{1}{\nu}[Y - \hat{Y}]^T W [Y - \hat{Y}], \tag{214}$$

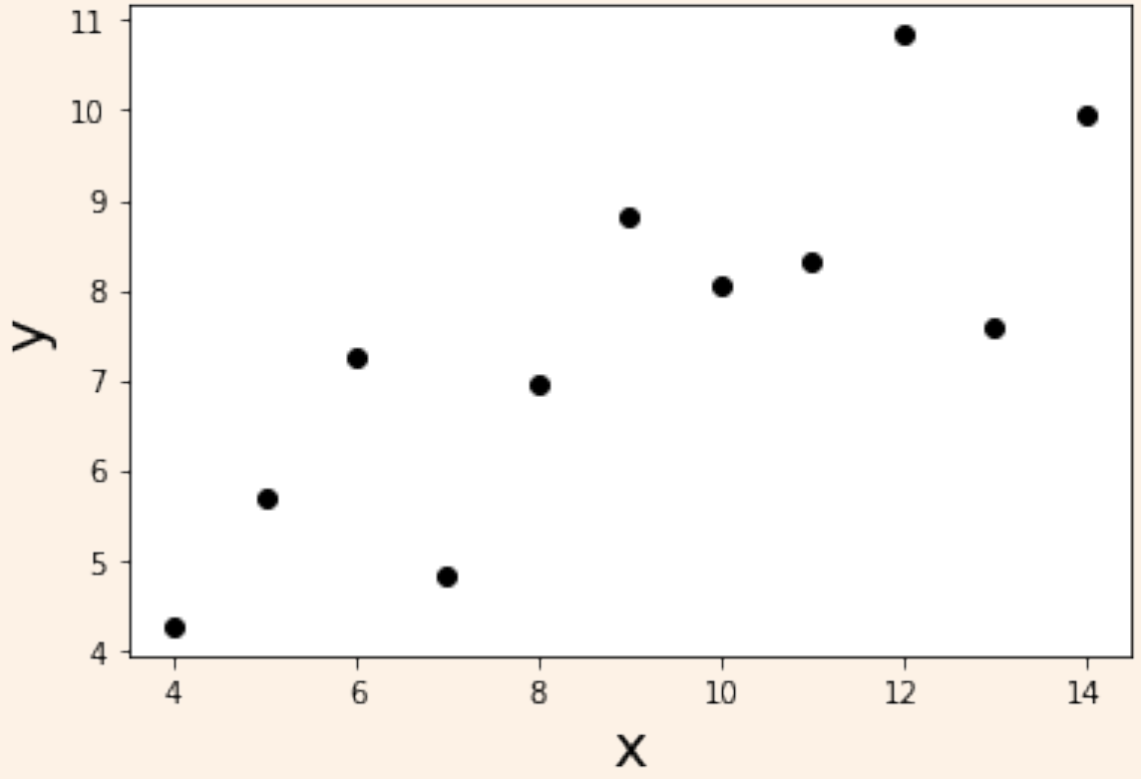
where $\nu := N - d$ is the number of d.o.f. in the fit.

- For data with estimated errors, $\chi^2_\nu \approx 1$ indicates that the model is good (in terms of both fit and errors), while $\chi^2_\nu < 1 / \chi^2_\nu > 1$ indicates overfitting/underfitting respectively.
- In ordinary least squares, the errors are often unspecified a priori ($W = I$). Then χ^2_ν can be used as an a posteriori estimate of the error variance.

Example

Jupyter notebook: Apply ordinary least squares to the following data set:

$$x = (10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5),$$
$$y = (8.04, 6.95, 7.58, 8.81, 8.33, 9.96, 7.24, 4.26, 10.84, 4.82, 5.68). \quad (215)$$



Validating regression models

A pervasive theme throughout this course has been:

Don't just rely on black-or-white tests with simple summary statistics.

Example

Jupyter notebook: Apply ordinary least squares to the following data set:

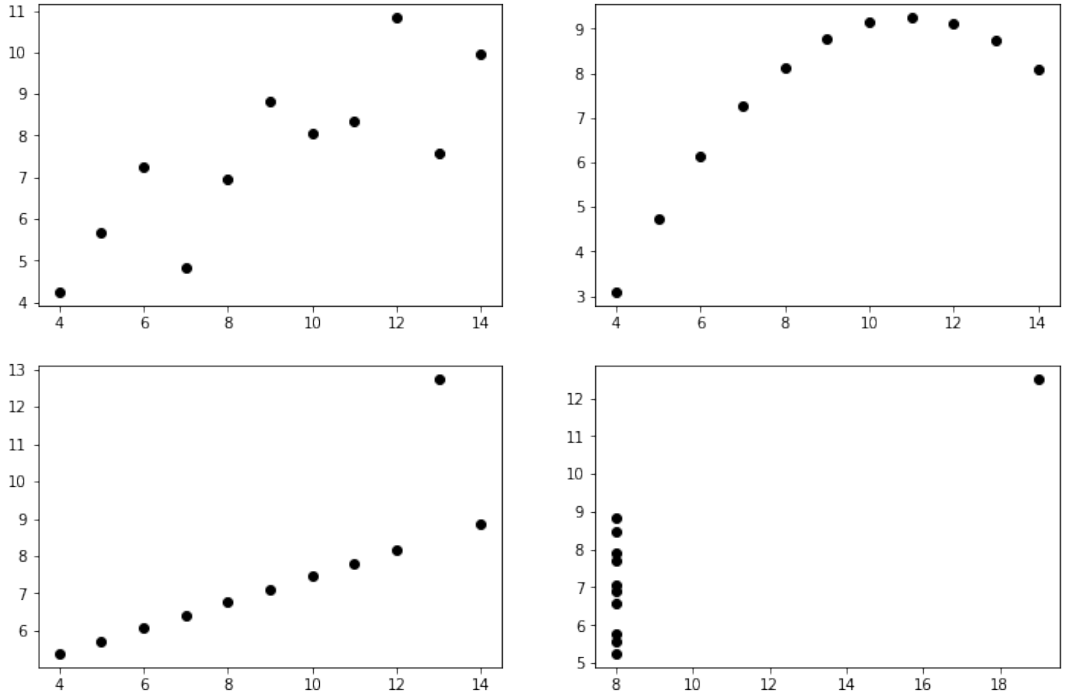
$$\begin{aligned} x &= (10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5), \\ y &= (7.46, 6.77, 12.74, 7.11, 7.81, 8.84, 6.08, 5.39, 8.15, 6.42, 5.73). \end{aligned} \tag{216}$$

One way to do better is through residual analysis: either direct visualisation (plots/histograms) of the residuals $Y - \hat{Y}$, or something more quantitative like a quantile–quantile (Q–Q) plot.

- A Q–Q plot is used to graphically compare two probability distributions (theoretical or empirical). For some set of quantile probabilities $q \in [0, 1]$, plot the q-th quantile of the distributions against each other.
- In the case of linear least squares, the quantiles of the residuals can be plotted against the exact quantiles of the normal distribution $\Phi^{-1}(q)$.

Cross-validation can also be used: in the context of regression, it can be as simple as leaving out a small validation set, and comparing statistics for the training and validation sets.

The data sets in Eq. (215) and Eq. (216) are members of Anscombe's quartet — a classic example of four data sets that have near-identical summary statistics but are qualitatively very different. They were introduced to highlight the importance of graphical analysis.



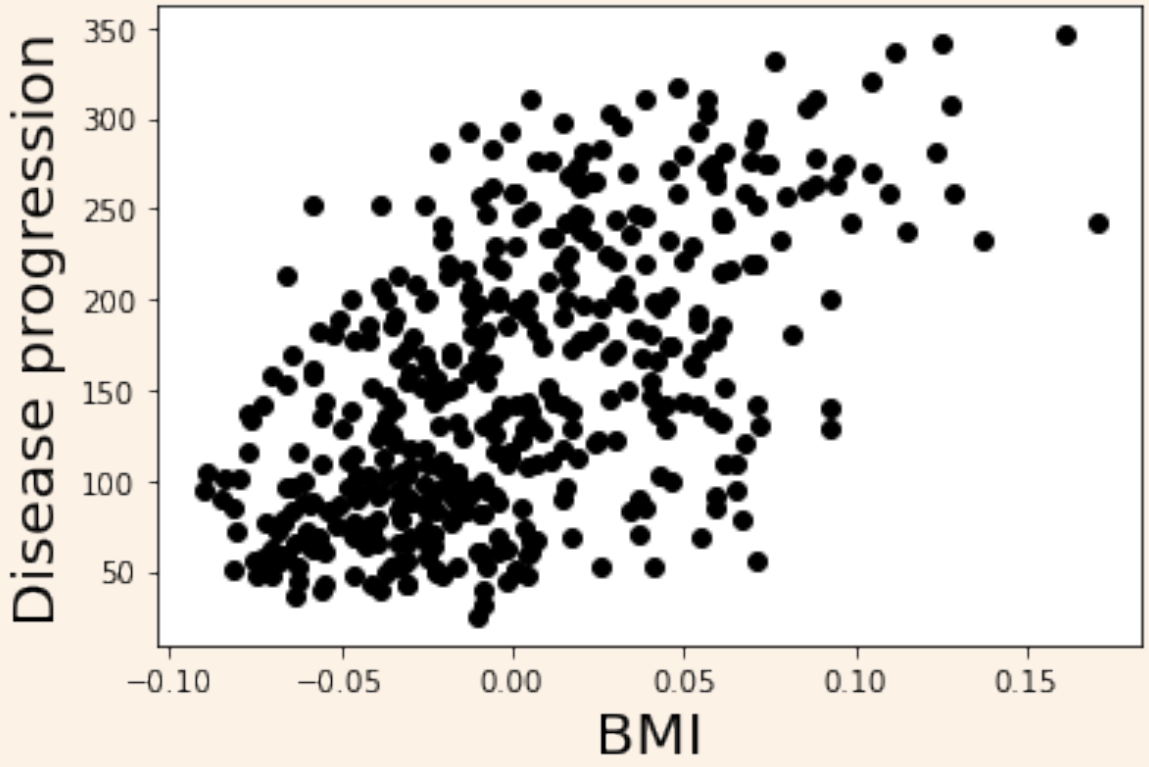
Graphical analysis is not the same as doing statistics purely visually, i.e., the derogatory "chi-by-eye". The point is that quantitative + qualitative is the best approach.

Example

Jupyter notebook: Apply weighted least squares to the data set in Eq. (216).

Example

Jupyter notebook: In a study of $N = 442$ diabetes patients, data was obtained for $n = 10$ regressors (age, sex, body mass index, average blood pressure and six blood serum measurements), along with the response of interest (some quantitative measure of disease progression after a year). Apply ordinary least squares to this data set.



General linear models

This is a catch-all term that might be used for various extensions of linear models. Crucially, in all of these methods, the model remains linear with respect to its parameters w.

The standard use of the term is to refer to *multivariate linear regression*, where the target $y \in \mathbb{R}^m$ is now vector-valued. There are then $m(n+1)$ instead of $n+1$ parameters, but everything is still linear and so the extension from multiple linear regression is straightforward.

We may also consider methods like *polynomial regression* as general linear models (if not linear models outright, according to standard definitions). This can be confusing, because such models are not linear with respect to the raw data x.

- For example, in quadratic regression, each row of the design matrix X is $(x^2, x, 1)$
- The regressors are then correlated but not perfectly, so $X^T X$ is generally still invertible

Example

Jupyter notebook: Apply quadratic regression to the second data set in Anscombe's quartet.

In general, any basis of regressors can be used. Beyond the polynomial basis $\{x^j\}$, other examples include the Fourier basis $\{e^{ijx}\}$, radial bases $\{f(|x - c_j|)\}$, etc.

Regularisation

There are two main motivations for introducing regularisation into linear least squares.

- Conditioning: The Gram matrix typically becomes ill-conditioned as d grows large. Intuitively, as more regressors are considered, it becomes more likely for the two most correlated ones to have a high correlation — causing the corresponding columns of X to be almost proportional, which in turn makes $X^T X$ near-singular. This sort of poor conditioning cannot be solved with simple data transformations.
- Overfitting: Even if poor conditioning does not cause numerical problems, it generally leads to large variance in the model parameters w, and thus overfitting. When there are as many independent d.o.f. as data points ($d = N$), the fit becomes exact.

One regularisation method is *ridge regression*. Consider the least-squares loss function in Eq. (210), but with an additional term that penalises large values for the components of w:

$$L'(Y, \hat{Y}) := |Y - \hat{Y}|^2 + \lambda |w|^2, \tag{217}$$

where $|\cdot|$ denotes the L2 norm $||\cdot||_2$ (on $\mathbb{R}^N$ for the first term, and on $\mathbb{R}^d$ for the second).

- The minimisation of $L'(Y, \hat{Y})$ can also be viewed as the minimisation of $L(Y, \hat{Y})$ under the constraint $|w|^2 = \text{const.}$, with Lagrange multiplier λ
- In practice, we use λ as a tunable hyperparameter

Exercise

Show that $L'(Y, \hat{Y})$ is minimised when

$$w = (X^T X + \lambda I)^{-1} X^T Y. \tag{218}$$

Example

Jupyter notebook: Apply polynomial ridge regression to the first Anscombe data set.

Another popular regularisation method is *lasso regression* (lasso is actually an acronym for "least absolute shrinkage and selection operator"). Here, the regularisation term in Eq. (217) is replaced with $\lambda||w||_1$, where the L1 norm $||\cdot||_1$ of a vector is simply the sum of the absolute values of its components.

- Lasso regression does not have a closed-form solution for w, except in special cases
- The solution for w is generally sparse, which effectively plays the role of regressor selection

Generalised Linear Models

Consider now the (unfortunately named) generalised linear model

$$\hat{Y}(X) := f^{-1}(Xw) = \mathrm{E}[Y|X], \tag{219}$$

where f is the link function. This contains all general linear models, in which f is the identity.

We said earlier that general linear models assume normally distributed errors $Y - \hat{Y}$, but nothing in the model looks explicitly normal. Where is normality imposed?

- Somewhat subtle, but it is implicitly assumed through the choice of loss function! The errors don't have to be normal, just that the model is optimal when they are.

Exercise

Show that minimising the squared error between Y and $\hat{Y}$ with the identity link function is equivalent to maximising the conditional probability of Y given X under the normal model

$$p(Y|X) = \mathcal{N}(Y|Xw, \sigma^2 I). \tag{220}$$

By choosing different link functions f (that are not linear transformations), we end up with a much broader class of models that are no longer linear in their parameters w.

- Recall that we first encountered link functions in the context of classification, e.g., the logit link (logistic regression) or the probit link (probit model). More broadly, all linear classifiers are subsumed in the framework of generalised linear models.

When fitting generalised linear models, the objective function is canonically the log-probability of $Y|X$ (essentially, the log-likelihood with data $Y|X$ and parameters w — we will examine this connection to Bayesian statistics shortly). Equivalently, we minimise:

$$L(Y, \hat{Y}) := -\ln p(Y|\hat{Y}(X)), \tag{221}$$

where the conditional distribution p is assumed, and is parametrised by its mean $\hat{Y} = \mathrm{E}[Y|X]$ (possibly along with other parameters that can be assumed or estimated, e.g., σ^2).

In practice:

- The focus of regression is more often on fitting $\mathrm{E}[Y|X]$, rather than the full statistical model $p(Y|X)$. Thus the choice of f is often practical rather than fundamental.
- For example, if the range of $\mathrm{E}[Y|X]$ is the real line, the identity link generally works well with a good choice of regressors (with respect to which $\mathrm{E}[Y|X]$ is approximately linear). In binary classification, $\mathrm{E}[Y|X] \in [0, 1]$, so we have to choose $f : (0, 1) \to (-\infty, \infty)$.
- If the aim is just to fit $\mathrm{E}[Y|X]$, the choice of loss function L may also not have a big impact. For example, the squared-error loss usually gives similar results to the cross-entropy loss when used in logistic regression.

Poisson regression is a simple example of a generalised linear model that can be used for discrete count-type data. The conditional distribution of $Y|X$ is a Poisson distribution with mean $\lambda(X) \in (0, \infty)$. Its canonical link function is the logarithm, i.e.,

$$\hat{Y}(X) := [\exp(w \cdot x_i)] \equiv \exp(Xw). \tag{222}$$

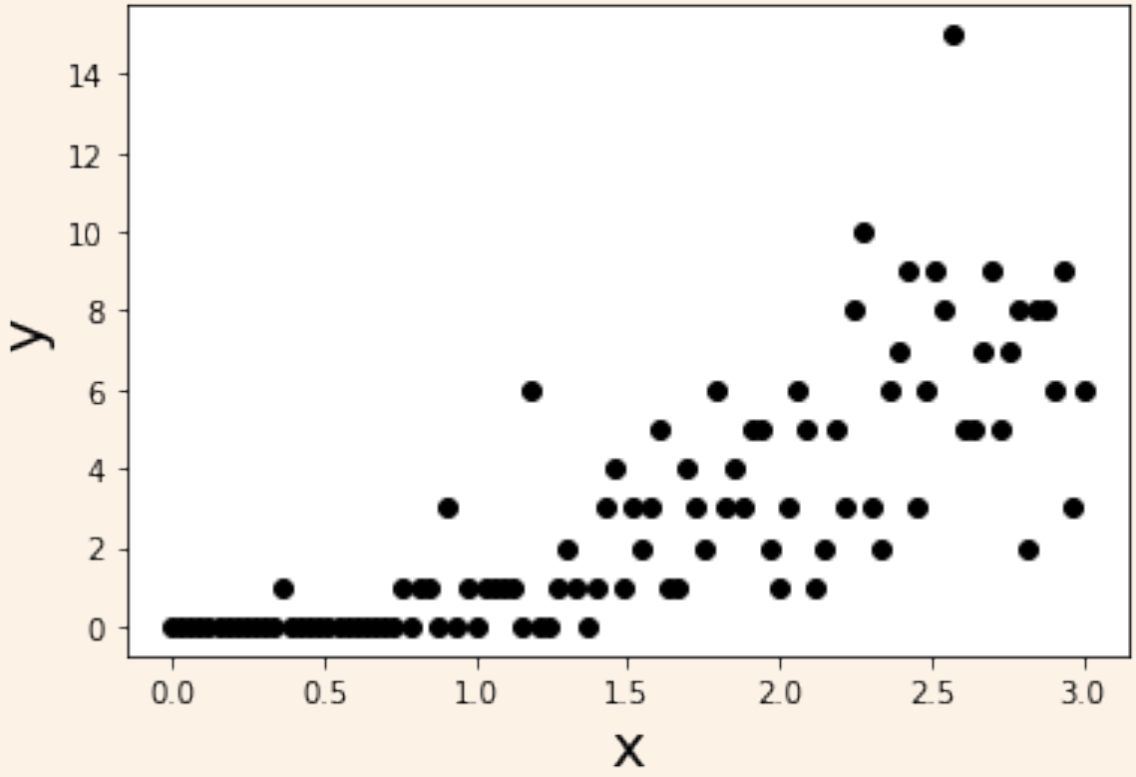
Exercise

Show that the loss function in Eq. (221) is

$$L(Y, \hat{Y}) = \sum_i (\exp(w \cdot x_i) - (w \cdot x_i) y_i). \tag{223}$$

Example

Jupyter notebook: Apply Poisson regression to the data set plotted below.



Finally — in light of this framework, it might be somewhat valid to think of neural networks simply as recursive and interconnected generalised linear models. But they are often referred to as nonlinear, so perhaps that is as far as the definition of "linear" stretches.

Bayesian approach to regression

Note that the normal form for a general linear model in Eq. (220) can be written more explicitly as a Bayesian likelihood in our usual notation:

$$L(\theta|X,Y) = p(Y|X,\theta) = \mathcal{N}(Y|X\theta_1, \theta_2 I), \tag{224}$$

where $\theta_1 \equiv w$ is a d-vector and $\theta_2 \equiv \sigma^2$ is a positive scalar. Recall that the interpretation of the likelihood is as a function of θ given X, Y (the fixed data set).

- When performing standard regression with a general linear model, we are typically concerned only with $\mathrm{E}[Y|X]$, and thus only with θ_1 and its least-squares solution

$$\hat{\theta}_1 := (X^T X)^{-1} X^T Y. \tag{225}$$

- We can however use the likelihood formulation to estimate all the parameters in our model, by maximising the log-likelihood over θ. The final regression model is effectively specified by the MLE, which we already solved for when examining Eq. (220).
- This approach can extend to the full multinormal model (where $\theta_2 \equiv \Sigma$ is the entire covariance matrix), or even any generalised linear model (where only θ_1 contribute to $\mathrm{E}[Y|X]$ via $f^{-1}(X\theta_1)$, while θ_2 are any remaining d.o.f. that do not).

The likelihood formulation thus reveals a clear path to the fully Bayesian approach. We simply specify priors for the model parameters, and proceed as we would in an inference problem.

In full generality, we may always estimate the posterior $p(\theta|X,Y)$ by generating and analysing samples from it, using the methods covered earlier in the course. The final regression model may then be specified by the posterior mean — or by the MAP estimate, which accounts for prior information (but corresponds to the MLE if flat priors are used).

For general linear models (which are normal or multinormal), there is also the option of solving semi-analytically for $p(\theta|X,Y)$ with simple or conjugate priors. We already have all the tools to do this, from our study of the normal and multinormal Bayesian models. For example, for the likelihood in Eq. (224) with the uninformative prior

$$\pi(\theta) \propto \theta_2^{-1}, \tag{226}$$

we may perform a conditional–marginal factorisation to obtain

$$\theta_1|\theta_2,X,Y \sim \mathcal{N}\left(\hat{\theta}_1, \theta_2(X^TX)^{-1}\right), \quad \theta_2|X,Y \sim \text{Inv-}\chi^2_\nu\left(\frac{|Y-X\hat{\theta}_1|^2}{\nu}\right). \tag{227}$$

The Bayesian approach takes more work, but allows for *probabilistic regression* — instead of point estimates, we now have posteriors and all the benefits they provide. For example:

- The model can make probabilistic predictions, which may be useful in certain contexts
- One can test hypotheses such as whether the weight of a particular regressor is statistically inconsistent with zero, and thus has a contribution to the response

15. Kernel Methods

The "Kernel Trick"

Consider a general training set $\{(x, y)_i\}$ of size N, where $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$. Construct the $N \times n$ data matrix X whose rows are x_i, and the $N \times m$ target matrix Y whose rows are y_i.

Note that we have been using X to denote the design matrix so far (since the data matrix itself is a viable design matrix), but here we extend to a general definition for the design matrix:

$$\Phi := \begin{pmatrix} \phi(x_1) \\ \vdots \\ \phi(x_N) \end{pmatrix} \equiv \begin{pmatrix} \phi_1 \\ \vdots \\ \phi_N \end{pmatrix} \equiv \phi(X), \tag{228}$$

where the vector-valued function $\phi : \mathbb{R}^n \to \mathbb{R}^d$ is called a *feature map*.

- For example, in quadratic regression we had $\phi(x) = (x^2, x, 1)$
- More generally, d can be less than n (and it often is in machine learning)

A general linear model with the least-squares solution may then be written as

$$\hat{y}(x) := \phi(x)^T \hat{w}, \quad \hat{w} := (\Phi^T \Phi)^{-1} \Phi^T Y, \tag{229}$$

where the matrix $\Phi^T \Phi$ is assumed to be invertible (or regularised).

- When evaluated on the training data, we write $\hat{Y}(X) = \Phi \hat{w}$
- When evaluated on a new data point x_*, we write $\hat{y}(x_*) = \phi_*^T \hat{w}$

Exercise

Show that

$$\hat{w} = \Phi^T(\Phi\Phi^T)^{-1}Y := \Phi^T\tilde{Y}, \tag{230}$$

where $\tilde{Y}$ is an $N \times m$ matrix.

Thus for any data point x_*, we have

$$\hat{y}(x_*) = \phi_*^T\Phi^T\tilde{Y} = \sum_i^N \phi_*^T\phi_i\tilde{y}_i := \sum_i^N k(x_*, x_i)\tilde{y}_i = [k(x_*, x_i)]^T\tilde{Y}, \tag{231}$$

where the $\tilde{y}_i$ are the rows of $\tilde{Y}$. The two-point scalar-valued function k is known as a *kernel*. It is a measure of similarity, and takes on its maximal value when its arguments are equal.

What is the point of all this algebraic manipulation?

- In the original formulation, we define the feature map $\phi : \mathbb{R}^n \to \mathbb{R}^d$ and evaluate it explicitly for x_*, then apply the linear transformation $\hat{w} : \mathbb{R}^d \to \mathbb{R}^m$
- In the kernel formulation, if $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is directly available, we evaluate it N times and use the results as weights for the $\tilde{y}_i$
- Which is better as $d \to \infty$?

The formulation of classification/regression in terms of k is known colloquially as the kernel trick, and methods that utilise kernels are referred to broadly as kernel methods.

In the least-squares example we just saw, the kernel is explicitly the Euclidean inner product of the features of x. More generally, a valid kernel is defined by any inner product on the *feature space* (the codomain of ϕ), which can be $\mathbb{R}^d$ or an infinite-dimensional Hilbert space.

Example

For $x \in \mathbb{R}$, what is a feature map corresponding to $k(x, x') = xx' - 1$? What is a feature map corresponding to $k(x, x') = \exp\left(-(x - x')^2/2\right)$?

Crucially, kernel methods bypass the explicit definition of ϕ by specifying valid k directly. The result that allows us to do this is known as Mercer's theorem. Essentially, the inner-product condition is satisfied as long as the kernel is positive semi-definite (PSD), i.e., for any set of vectors $\{x_1, \ldots, x_N\} \in \mathbb{R}^n$, the matrix $K = [k(x_i, x_j)]$ is PSD:

1. K is symmetric
2. $z^T K z \geq 0$ for all $z \in \mathbb{R}^N$

Some examples of common kernels

- Linear: $k(x, x') = x^T x'$
- Cosine: $k(x, x') = x^T x' / (|x||x'|)$
- Polynomial: $k(x, x') = (x^T x' + c)^p$ where $c \geq 0, p > 0$
- Gaussian: $k(x, x') = \exp\left(-c|x - x'|^2\right)$ where $c > 0$

How do we know that an arbitrary kernel is PSD? We can't possibly form and check all possible matrices K. However, the set of all PSD kernels on $\mathbb{R}^n \times \mathbb{R}^n$ is closed under addition, multiplication, and limits. For any PSD kernels $k_1, k_2, \ldots$, the following are also PSD kernels:

- $k_i + k_j$
- $k_i k_j$
- $\lim_{i \to \infty} k_i$ if the limit exists

Exercise

Show that the polynomial kernel is PSD.

Pros and cons of kernel methods:

- Avoid explicit construction of feature maps; non-parametric and thus more flexible
- Computational costs generally scale poorly with size of training set N

Examples of kernel methods:

- Kernel density estimation (unsupervised learning)
- Support vector machines for classification and regression
- Gaussian process regression

Support Vector Machines

The earliest version of the support vector machine (SVM) algorithm was introduced by Vapnik and Chervonenkis in the 1960s. There have been many extensions since, and modern SVMs are accurate learning algorithms that reached peak popularity in the 1990s (but have since been overtaken by deep neural networks).

- One distinctive feature of SVMs is that they are sparse models, which helps to combat the poor N-scaling problem during model evaluation.

The linear SVM binary classifier

We will examine the simplest SVM first: a linear binary classifier that maps data $x \in \mathbb{R}^n$ to target $y \in \{-1, 1\}$ (note the relabelling of the first target class).

Consider first the linear function $x^T w + b$ for some $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$, whose level sets describe $(n-1)$-dimensional hyperplanes in $\mathbb{R}^n$.

- For example, $x^T w + b = \text{const.}$ is a line in the plane when $n = 2$.
- Note that $y = x^T w + b$ would describe an n-dimensional hyperplane in $\mathbb{R}^n \times \mathbb{R}$, but here we are looking only at the data space.

Now for a given training set $\{(x, y)_i\}$, is it possible to find a hyperplane that cleanly separates all the x_i with $y_i = -1$ from all the x_i with $y_i = 1$?

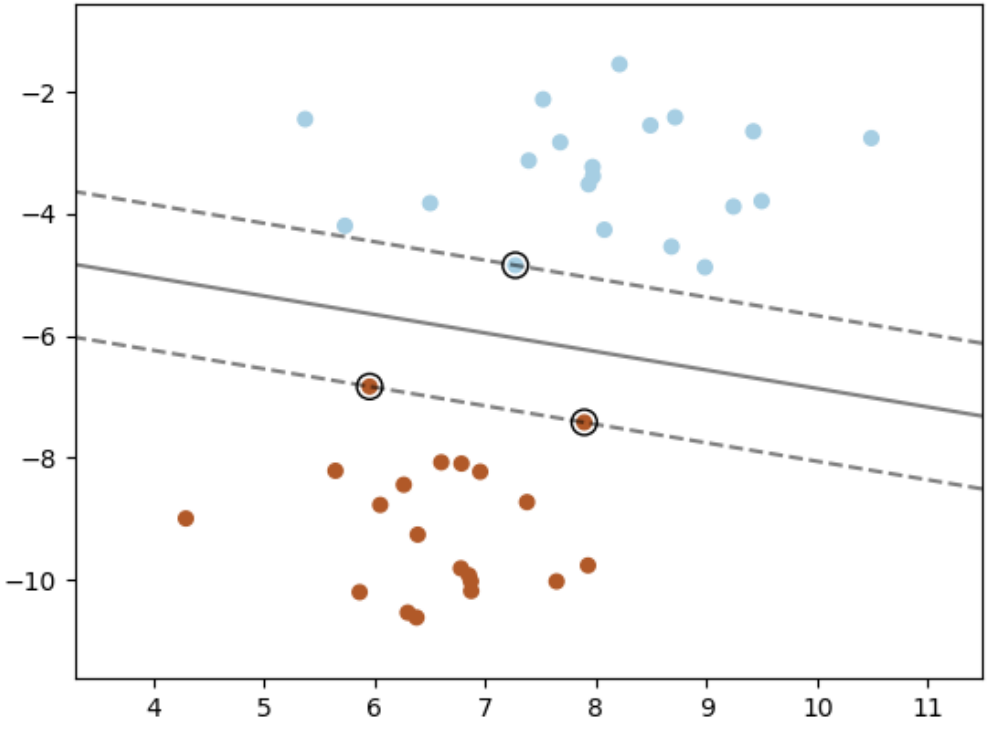
- Sometimes, but not generally. If such a hyperplane exists, we refer to the data as *linearly separable*. Let us assume linear separability for now.

The aim in an SVM is to find model parameters w, b for $\{(x, y)_i\}$ such that:

1. The two hyperplanes $x^T w + b = -1$ and $x^T w + b = 1$ (as well as all hyperplanes in between) cleanly separate the two classes of data. More precisely, $x_i^T w + b \leq -1$ if $y_i = -1$ and $x_i^T w + b \geq 1$ if $y_i = 1$. Equivalently, we require for all i that

$$y_i(x_i^T w + b) \geq 1. \tag{232}$$

2. The hyperplanes themselves have a maximal *margin*, i.e., they are as far apart as possible.



Exercise

Show that the margin between the two hyperplanes is given by $2/|w|$.

Formally, the trained SVM is thus the solution to an optimisation problem with N constraints:

$$\min_{w,b} \frac{1}{2}|w|^2, \quad y_i(x_i^T w + b) \geq 1. \tag{233}$$

This is known as a "hard-margin" SVM, which admits a unique solution $\hat{w}, \hat{b}$ if and only if the data is linearly separable (and no solution otherwise).

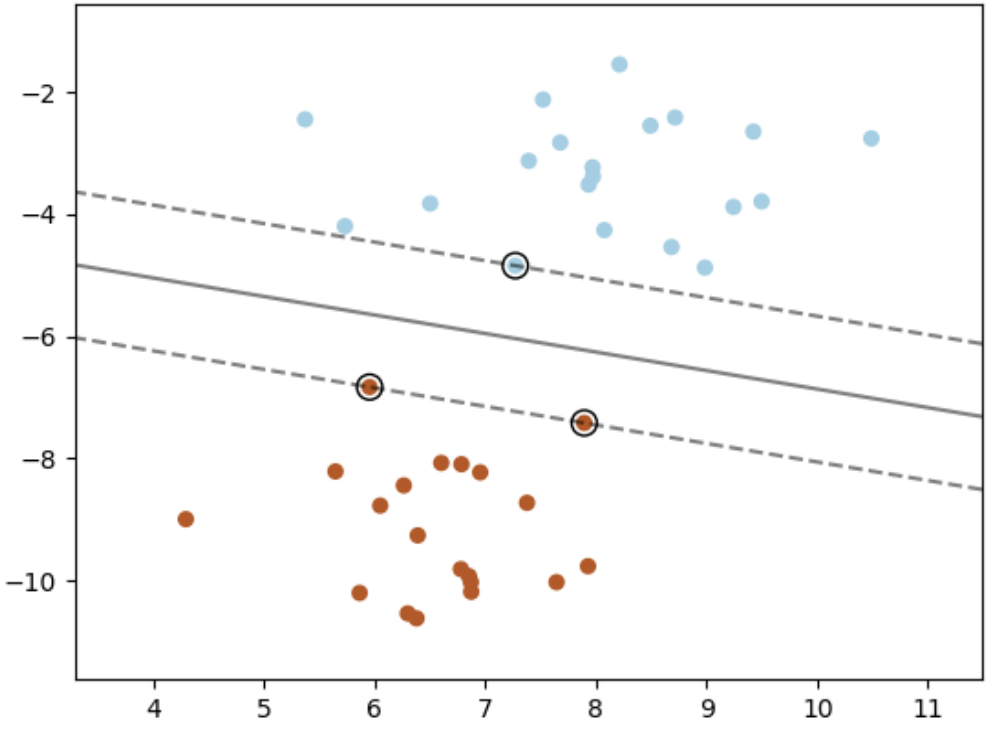
- We will not delve into the theory/practice of constrained optimisation for the purposes of this course, but such problems can generally be solved with standard numerical libraries.

Note that by construction, the hyperplanes will necessarily pass through a number of data points x_i — these are the *support vectors* that the method is named for.

- The number of support vectors depends on the structure of the training set (intuitively, it depends on the convex hulls of the grouped data). There can be as few as two (in binary classification), and as many as the size of the training set itself.

The central bisector $x^T w + b = 0$ is known as the maximum-margin hyperplane, and implicitly defines the final model $\hat{y}$ with trained parameters $\hat{w}, \hat{b}$:

$$\hat{y}(x_*) := \operatorname{sgn}(x_*^T \hat{w} + \hat{b}). \tag{234}$$



Kernel SVMs

In kernel SVMs, the linear transformation $x^T w + b$ is effectively replaced by a more general transformation in terms of some (implicitly defined) feature map: $\phi(x)^T w' + b$, where w' now has some arbitrary dimensionality d. As before, we will use the kernel formulation to completely avoid defining ϕ and d in the first place: $\phi(x)^T \phi(x') \to k(x, x')$.

- The hard-margin SVM can be solved for in the form of Eq. (233) only if ϕ is defined, so in order to generalise to kernels, we need to do a bit of extra work.

In the well-known method of Lagrange multipliers, optimising $f(z)$ subject to N constraints $g_i(z) = 0$ is done by finding stationary points of the Lagrangian $L(z, \lambda) := f(z) + \lambda^T g(z)$.

If we wish to optimise $f(z)$ subject to N *inequality* constraints $g_i(z) \leq 0$, this can be achieved in the same way, but with two conditions on the multipliers (now denoted μ):

1. $\mu_i \geq 0$ for all i ("dual feasibility")
2. $\mu_i g_i(\hat{z}) = 0$ for all i, where $\hat{z}$ is a local optimum ("complementary slackness")

These conditions (plus the constrained optimisation problem itself) are known as the Karush–Kuhn–Tucker (KKT) conditions, and we will refer to μ as KKT multipliers.

Returning to Eq. (233), we see that:

- $f(w, b) \equiv |w|^2/2$
- $g_i(w, b) \equiv 1 - y_i(x_i^T w + b)$

Thus the Lagrangian is

$$L(w, b, \mu) := \frac{1}{2}|w|^2 + \sum_i^N \mu_i (1 - y_i(x_i^T w + b)). \tag{235}$$

Exercise

For a stationary point $(\hat{w}, \hat{b}, \hat{\mu})$ of the Lagrangian, show that

$$\hat{w} = \sum_i^N \hat{\mu}_i y_i x_i, \quad \sum_i^N \hat{\mu}_i y_i = 0. \tag{236}$$

By substituting Eqs (236) into the Lagrangian, the optimisation problem of Eq. (233) can ultimately be cast in terms of μ as

$$\max_\mu \left(\sum_i^N \mu_i - \frac{1}{2} \sum_{i,j}^N \mu_i y_i (x_i^T x_j) y_j \mu_j \right), \quad \mu_i \geq 0, \quad \sum_i^N \mu_i y_i = 0. \tag{237}$$

In the language of optimisation, Eq. (233) is in primal form, and Eq. (237) is its dual.

The final linear SVM model can be rewritten as

$$\hat{y}(x_*) = \mathrm{sgn}(x_*^T \hat{w} + \hat{b}) = \mathrm{sgn}\left(\sum_i^N \hat{\mu}_i y_i (x_i^T x_*) + \hat{b} \right), \tag{238}$$

using the first of Eqs (236). Recall that $y_i = x_i^T \hat{w} + \hat{b}$ if and only if x_i is a support vector, which allows $\hat{b}$ to be solved for in terms of $\hat{\mu}$ for any support vector.

Crucially, note that:

- The model is sparse, because any x_i that is not a support vector has

$$g_i(\hat{w}, \hat{b}) = 1 - y_i(x_i^T \hat{w} + \hat{b}) \neq 0, \tag{239}$$

  which in turn forces the corresponding $\hat{\mu}_i$ to be zero from the condition of complementary slackness — i.e., only the support vectors are required during model evaluation.
- The dual optimisation problem involves a quadratic function in μ, and is much easier to solve efficiently than the primal problem.

Finally, the dual form of the SVM optimisation problem makes it explicitly clear how to incorporate kernels in SVMs. Eq. (237) depends on the data x only through the quantity $x_i^T x_j$, which is essentially $\phi(x_i)^T \phi(x_j)$ with the identity feature map.

In a kernel SVM, then, we instead solve the very similar optimisation problem

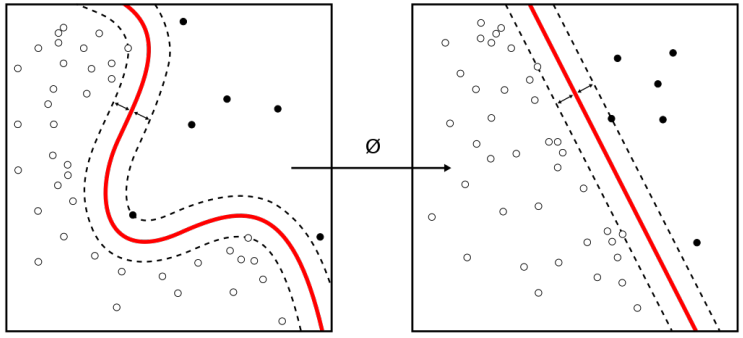
$$\max_{\mu} \left(\sum_{i}^{N} \mu_i - \frac{1}{2} \sum_{i,j}^{N} \mu_i y_i k(x_i, x_j) y_j \mu_j \right), \quad \mu_i \geq 0, \quad \sum_{i}^{N} \mu_i y_i = 0, \tag{240}$$

for some choice of kernel k.

The final model becomes

$$\hat{y}(x_*) := \operatorname{sgn}(\phi(x_*)^T \hat{w} + \hat{b}) = \operatorname{sgn} \left(\sum_{i}^{N} \hat{\mu}_i y_i k(x_i, x_*) + \hat{b} \right), \tag{241}$$

where $\hat{b}$ is solved for analogously to the linear model.



Other extensions

There are many extensions to the linear SVM binary classifier. The most important ones are:

- Soft-margin SVMs, where additional variables ζ_i are introduced into the optimisation problem to account for data that is not linearly separable. In primal form, we have

$$\min_{w,b,\zeta} \left(\frac{1}{2}|w|^2 + C\sum_i^N \zeta_i \right), \quad y_i(x_i^T w + b) \geq 1 - \zeta_i, \quad \zeta_i \geq 0, \tag{242}$$

  where the freedom in ζ_i allows support vectors to fall within the margin, but with a penalty (C is a tunable hyperparameter that controls the overall strength of the penalty).
- Multiclass SVMs (usually built from multiple binary classifiers, but not always).
- Regression SVMs, where the optimisation problem becomes

$$\min_{w,b} \frac{1}{2}|w|^2, \quad |y_i - (x_i^T w + b)| \leq \epsilon, \tag{243}$$

  with ϵ a tunable hyperparameter that controls the overall prediction error.
- Probability estimates can be computed through external means, e.g., cross-validation.

Example

Jupyter notebook: Apply an SVM classifier to the iris flower data set.

Gaussian Process Regression

Standard linear and kernel SVMs are extensions of general linear models that focus purely on the geometric formulation of the regression problem (minimising the overall distance between the training targets Y and the model predictions $\hat{Y}$). Thus they do not have an explicit probabilistic interpretation, and do not directly provide probability estimates.

- However, we saw earlier that general linear models can also be treated in a probabilistic way using the likelihood formulation (maximising the conditional probability of the training targets Y given the training data X).

Gaussian process regression (GPR) is a kernel-based method on that end of the spectrum; it was introduced by Krige and Matheron in the 1960s, and is also known as *kriging*. Like SVMs, GPR is widely used as well — but usually more in the context of (Bayesian) statistical modelling rather than in machine-learning applications.

Distinctive features:

- GPR is essentially Bayesian prediction; it relies on the posterior solution to a multinormal model for the weight parameters, but the prior is updated/trained as hyperparameters
- On the flip side — GPR models are generally not sparse and can be computationally unwieldy, although there are extensions that mitigate this somewhat

Consider the likelihood formulation of the general linear model from Eq. (224), but in terms of the data features $\Phi(X)$ rather than the raw data X:

$$L(\theta|X,Y) = p(Y|X,\theta) = \mathcal{N}(Y|\Phi\theta, \sigma_n^2 I), \tag{244}$$

where we will treat the variance of the model as a hyperparameter (denoted now by σ_n^2).

- Equivalently, we are assuming that for all $(x,y)_i$ in the training set, the predictions of the fitted model $\hat{y}_i \equiv \hat{y}(x_i) := \phi_i^T \hat{\theta}$ have iid errors:

$$y_i - \hat{y}_i \sim \mathcal{N}(0, \sigma_n^2), \tag{245}$$

  and so σ_n^2 has an interpretation as a "noise" variance.

In GPR, we assume a conjugate normal prior for the model parameters:

$$\pi(\theta) = \mathcal{N}(0, \Sigma_\pi), \tag{246}$$

such that the posterior has an analytical solution. This can be obtained using our more general treatment of the multinormal model, but here things are much simpler as the covariance matrix of the data distribution is fixed, and proportional to the identity.

Following (the multivariate analogue of) the derivation of Eqs (36) and (37) for the normal model with known variance, the posterior distribution is

$$\theta|X,Y \sim \mathcal{N}\left(\sigma_n^{-2}A^{-1}\Phi^T Y, A^{-1}\right), \tag{247}$$

where the posterior precision

$$A := \sigma_n^{-2}\Phi^T\Phi + \Sigma_\pi^{-1} \tag{248}$$

is the sum of the likelihood and prior precisions.

In GPR, the main focus is not so much on the parameter posterior itself, but rather the posterior predictive distribution. In other words, the model is used to make a probabilistic prediction $p(\hat{y}|x_*, X, Y)$, rather than the usual point prediction $\hat{y}(x_*) = \mathrm{E}[\hat{y}|x_*, X, Y]$.

The posterior predictive distribution is given by

$$\begin{aligned} p(\hat{y}|x_*, X, Y) &= \int d\theta\, p(\hat{y}|x_*, \theta)p(\theta|X, Y) \\ &= \mathcal{N}\left(\hat{y}|\sigma_n^{-2}\phi_*^T A^{-1}\Phi^T Y, \phi_*^T A^{-1}\phi_*\right), \end{aligned} \tag{249}$$

again from the fairly standard (if tedious) convolution of two Gaussians.

We are now in the position to apply the kernel trick, so as to avoid working with the explicit feature map ϕ (and to allow its implicit infinite-dimensional generalisation).

Exercise

Show that

$$\sigma_n^{-2} A^{-1} \Phi^T = \Sigma_\pi \Phi^T (K + \sigma_n^2 I)^{-1}, \tag{250}$$

where $K = \Phi \Sigma_\pi \Phi^T$.

Thus the mean of the posterior predictive distribution is

$$\mathrm{E}[\hat{y} | x_*, X, Y] = \sigma_n^{-2} \phi_*^T A^{-1} \Phi^T Y = [\Phi \Sigma_\pi \phi_*]^T (K + \sigma_n^2 I)^{-1} Y. \tag{251}$$

A slightly longer calculation for the posterior predictive variance yields

$$\mathrm{Var}[\hat{y} | x_*, X, Y] = \phi_*^T \Sigma_\pi \phi_* - [\Phi \Sigma_\pi \phi_*]^T (K + \sigma_n^2 I)^{-1} [\Phi \Sigma_\pi \phi_*]. \tag{252}$$

We may then write the posterior predictive distribution as

$$\hat{y}|x_*, X, Y \sim \mathcal{N}\left(k_*^T(K+\sigma_n^2 I)^{-1}Y, k_{**} - k_*^T(K+\sigma_n^2 I)^{-1}k_*\right), \tag{253}$$

where

$$K = \Phi\Sigma_\pi\Phi^T = [\phi_i^T\Sigma_\pi\phi_j], \quad k_* = \Phi\Sigma_\pi\phi_* = [\phi_i^T\Sigma_\pi\phi_*], \quad k_{**} = \phi_*^T\Sigma_\pi\phi_*. \tag{254}$$

Finally, recall that Σ_π is the covariance matrix of the normal prior distribution. Thus it is positive-definite, and its associated quadratic form is a valid inner product. This allows us to replace the quadratic form with a general kernel k, and define

$$K := [k(x_i, x_j)], \quad k_* := [k(x_i, x_*)], \quad k_{**} := k(x_*, x_*). \tag{255}$$

Eq. (253) with (255) is essentially the full Bayesian solution to the regression problem — there is no optimisation over the model "parameters" θ, because we have already marginalised over the parameter posterior! We turn this into a learning problem by updating the prior (which is encoded in the specification of k), so as to maximise the conditional probability of Y given X.

But first: In the context of Gaussian processes, k is referred to as a *covariance function*. What is a Gaussian process, and how does it relate to the above derivation?

The Gaussian-process formulation of GPR

A Gaussian process can be viewed as an infinite-dimensional generalisation of the multinormal distribution (infinite number of variables $N \to \infty$, not number of features d).

- More precisely, it is defined as a countable set of variables $\{y_i\}$ such that any finite subset $\{y_1, \ldots, y_N\}$ has a Gaussian joint distribution.
- The index i of the process can describe any abstract space, not just time.

In GPR, the aim is to fit such a process to a given finite set $\{y_1, \ldots, y_N\}$, which allows predictions to be made about any y_* not in the set. As in the case of general regression, the problem can be cast not in terms of the y_i but some covariates x_i, which is then essentially a fit to the underlying relation $\hat{y}(x)$ between x and y.

We will only consider zero-mean Gaussian processes, which are fully specified by their covariance function k. The starting assumption is that $\hat{y}(x)$ is described by such a process:

$$\hat{y}_i \equiv \hat{y}(x_i) \sim \mathcal{GP}(k). \tag{256}$$

The covariance function describes the covariance between any pair of $\hat{y}_i$. For any finite (ordered) set $\hat{Y} \equiv (\hat{y}_1, \ldots, \hat{y}_N)$, we have:

$$\hat{Y} \sim \mathcal{N}(0, [k(x_i, x_j)]). \tag{257}$$

- Why does this still work when fitting data with a nonzero mean? One way to think of it is that the zero-mean process is only a Bayesian prior for $\hat{y}$ (in the same way that the model parameters were assigned a zero-mean prior in the linear-model formulation).

Note now that for some $\hat{y}_* \sim \mathcal{GP}(k)$ not in $\hat{Y}$, we can write the joint distribution of $(\hat{Y}, \hat{y}_*)$ as

$$\begin{pmatrix} \hat{Y} \\ \hat{y}_* \end{pmatrix} \sim \mathcal{N}\left(0, \begin{pmatrix} K & k_* \\ k_*^T & k_{**} \end{pmatrix}\right), \tag{258}$$

with K, k_*, k_{**} as given in Eq. (255).

In a regression problem, we do not have access to the idealised targets $\hat{y}_i$ that are exactly described by a Gaussian process, but rather the actual training targets y_i. If we make the same assumption in Eq. (245) that $y_i - \hat{y}_i$ is iid with zero mean and variance σ_n^2, we have:

$$\begin{pmatrix} Y \\ \hat{y}_* \end{pmatrix} \sim \mathcal{N}\left(0, \begin{pmatrix} K + \sigma_n^2 I & k_* \\ k_*^T & k_{**} \end{pmatrix}\right). \tag{259}$$

With the joint distribution in hand, it is straightforward (but again tedious) to obtain the conditional distribution of $\hat{y}_*|Y$, which is also Gaussian:

$$\hat{y}_*|Y \sim \mathcal{N}\left(k_*^T(K + \sigma_n^2 I)^{-1} Y, k_{**} - k_*^T(K + \sigma_n^2 I)^{-1} k_*\right). \tag{260}$$

This is identical to Eq. (253) (after restoring the implicit conditioning on x_*, X).

To sum up GPR: The probabilistic prediction for $\hat{y}_*$ is a normal distribution that depends on x_* (through k_* and k_{**}), X (through K and k_*), Y (only in the mean), and the choice of k.

Training the Gaussian process

As we saw earlier, choosing a kernel k is essentially equivalent to specifying a Bayesian prior for the implicit model parameters θ. This prior can then be "updated" to better fit the data.

In GPR, we maximise the probability of the training set $Y|X$ under the model (as we do for generalised linear models; see Eq. (221)). We shall call this quantity the hyperlikelihood*, since the model is now parametrised by some hyperparameters φ that describe the kernel. For example, $\varphi := (\sigma_y^2, \ell^2)$ for the kernel $k(x, x') = \sigma_y^2 \exp\left(-(1/2)|x - x'|^2/\ell^2\right)$.

Exercise

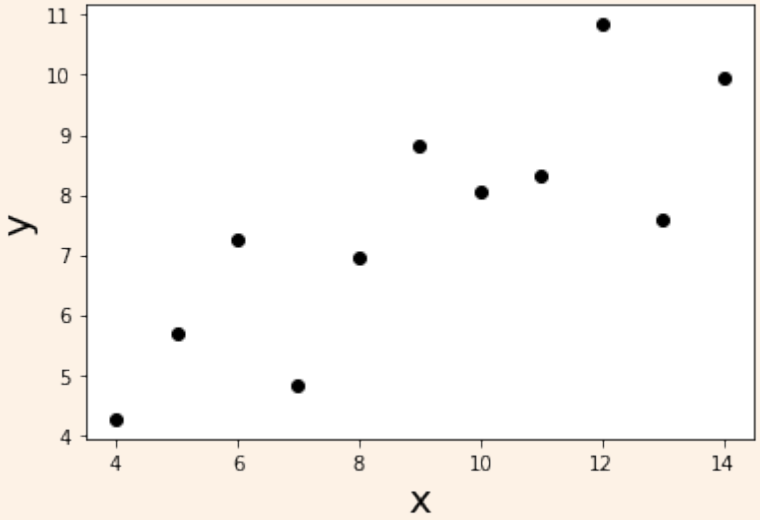
From Eq. (259), write down the logarithm of the hyperlikelihood $p(Y|X, \varphi)$.

- Note that the noise hyperparameter σ_n^2 actually describes the original likelihood in Eq. (244), rather than the prior. Depending on the application context, it can either be assumed and held fixed, or learnt from the training set as well.

*In standard treatments, $p(Y|X, \phi)$ is referred to as the marginal likelihood: "marginal" because $\hat{y}_*$ is marginalised over in the joint distribution (259), and "likelihood" even though the ϕ are technically hyperparameters. Consequently, it is sometimes also called the evidence.

Example

Jupyter notebook: Apply GPR to the first Anscombe data set.



Extensions of GPR

- Sparsity assumptions/approximations: Simple GPR models scale poorly with the size N of the training set, so a large variety of methods have been developed to mitigate this; e.g., sparse covariance functions, sparse methods for inverting $(K + \sigma_n^2 I)$ approximately, assumptions of sparsity on $(K + \sigma_n^2 I)^{-1}$.
- Gaussian process classification: This can be done, for example, by augmenting GPR with a logit link as in a generalised linear model.

16. Artificial Neural Networks

Multilayer Perceptrons

In this course, our treatment of artificial neural networks will focus mainly on multilayer perceptrons (MLPs), which are the simplest example of deep neural networks.

- They are important both historically (as the predecessor to more sophisticated networks) and in a modern context (as building blocks in such networks).
- They are also invaluable for introducing broader concepts/techniques in deep learning, and they provide a conceptual bridge between deep learning and some of the classical machine-learning methods we have seen so far.

MLPs are sometimes referred to as feedforward neural networks, but the latter term has a broader meaning. They are also called dense or fully connected neural networks, especially in the context of more complex networks that contain MLPs in their *architecture* (structure).

We saw earlier that kernel methods allow us to add complexity to the model by increasing the effective number of features (potentially to infinity), without specifying them explicitly.

- MLPs also add complexity through a large number of features that are not specified explicitly — the features are instead built up hierarchically through the composition of many interconnected functions, and learnt from the data.
- Like kernels, a choice of architecture must still be made, but this is far less constraining than engineering the features directly.

From generalised linear models to artificial neural networks

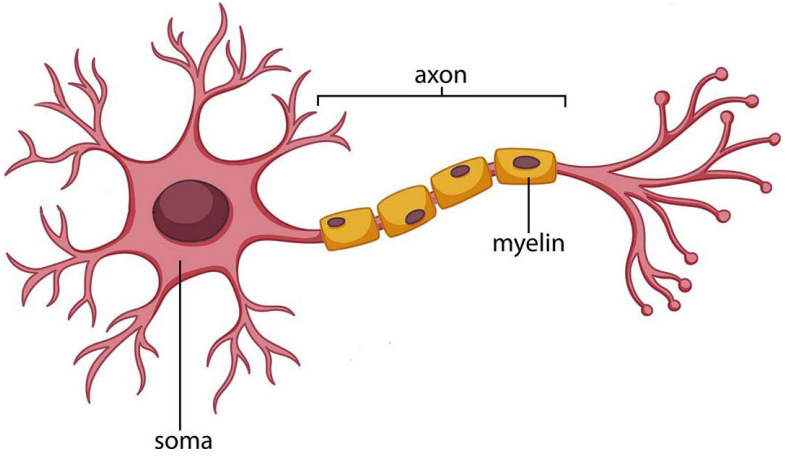
Recall from Eq. (219) that a generalised linear model (with non-identity link function) is simply

$$a(x) := \sigma(z(x)) := \sigma(x^T w + b), \tag{261}$$

where $a \equiv \hat{y}$ is the output; $\sigma \equiv f^{-1}$ is the activation function (not just the logistic function, although that too is a valid activation); and the bias term b has been explicitly restored.

In an artificial neural network, what we refer to as a *neuron/node* is essentially just a generalised linear model — with certain requirements on the choice of activation function, and without considering most of the underlying statistical framework.

- The term "perceptron" historically refers to the McCulloch–Pitts model, where the activation function is the Heaviside step function, and the number of neurons is one



Credit: Johns Hopkins Medicine

How do we construct a network of such neurons? Instead of directly setting $a \equiv \hat{y}$ as in a generalised linear model, let us add a *hidden layer* of neurons in between the input (the data vector x) and the final output (the estimate $\hat{y}$ of the associated target vector y). The number of neurons in the layer is called its *width*.

For a hidden layer of width d_1, we can write the layer output as $a_1 = \sigma_1(z_1)$, where

$$z_1 \equiv [z_{1j_1}] := [x^T w_{1j_1} + b_{1j_1}] \equiv W_1 x + B_1, \quad j_1 = 1, \ldots, d_1. \tag{262}$$

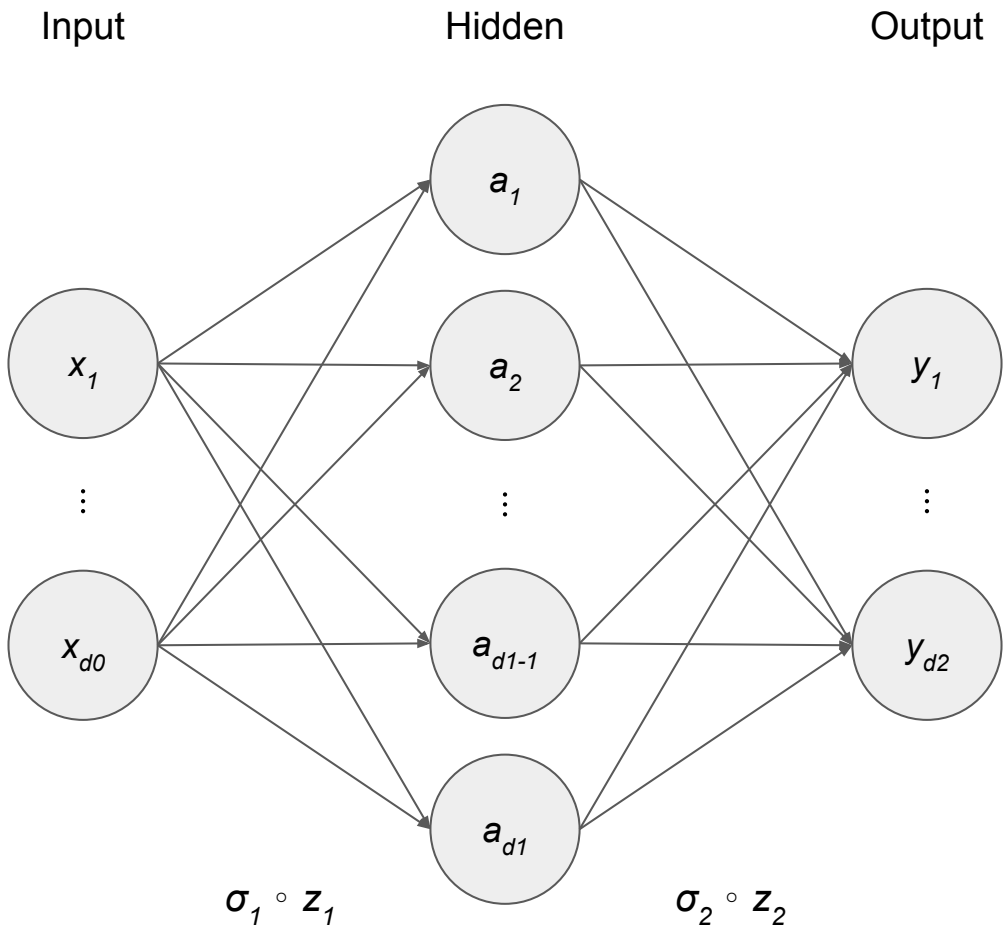
By definition, W_1 is a $d_1 \times d_0$ matrix (where d_0 is the dimensionality of x), while B_1 is a d_1-vector. The vector-valued activation function σ_1 usually also has dimensionality d_1.

Next, we pass the layer output a_1 as input to the final output $\hat{y} = \sigma_2(z_2)$:

$$z_2 \equiv [z_{2j_2}] := [a_1^T w_{2j_2} + b_{2j_2}] \equiv W_2 a_1 + B_2, \quad j_2 = 1, \ldots, d_2, \tag{263}$$

where W_2 is a $d_2 \times d_1$ matrix (where d_2 is the dimensionality of y), and B_2 is a d_2-vector.

- Note that there is actually nothing preventing us from choosing a different activation function for each neuron in the hidden layer, since σ_1 can act component-wise on z_1. However, this is very seldom done, as it introduces arbitrary d.o.f. for marginal gains.
- The activation functions σ_1 and σ_2 generally do differ, because the range of the latter is constrained by the form of y. For example, if y takes values on the probability simplex, then so too must σ_2 (e.g., the softmax function).


Input
Hidden
Output
a_1
x_1
a_2
y_1
x_{d0}
a_{d1-1}
y_{d2}
a_{d1}
$\sigma_1 \circ z_1$
$\sigma_2 \circ z_2$

We now expand the model further by adding multiple hidden layers between the input and the final output. The number of hidden layers in a network, plus one, is called its *depth*. Any network with depth > 2 is generally referred to as a deep neural network.

For a network of depth D, we may define $a_0 := x$ and $a_D := \hat{y}$, which leads to the following minimalist description of an MLP:

$$a_i = \sigma_i(z_i), \quad z_i = W_i a_{i-1} + B_i, \quad i = 1, \ldots, D, \tag{264}$$

where W_i is a $d_i \times d_{i-1}$ matrix and B_i is a d_i-vector.

- Again, there is nothing preventing us from allowing $\sigma_1, \ldots, \sigma_{D-1}$ to differ, but they seldom do in a standalone MLP.

Choosing the network architecture

Generally, specifying the architecture of an artificial neural network can seem more like an art than a science — although there are some guiding theoretical principles, architecture design in practice usually involves a lot of trial and error.

- In the simplest case of an MLP, choices about the depth D, the layer widths d_i, and the activation functions σ_i are required.
- Specialised architectures for certain types of data/task (e.g., convolutional neural networks) can have even more hyperparameters.

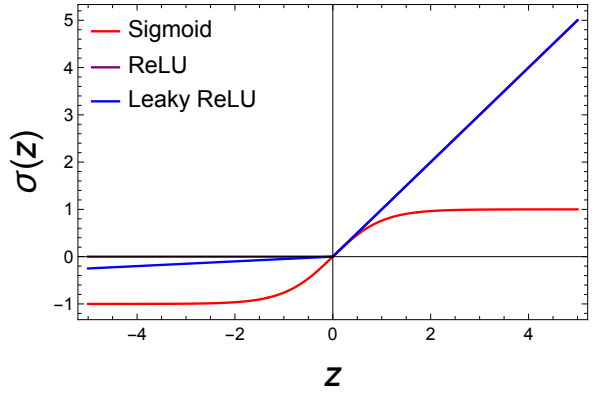
In most cases, MLPs only utilise two different activation functions — one for the output layer (σ_D), and a common one for all of the hidden layers ($\sigma_1 = \ldots = \sigma_{D-1}$).

Output activation functions

- Linear (identity): $\sigma(z) = z$
- Sigmoid (usually logistic; for mapping to interval $(0, 1)$): $\sigma(z) = 1/(1 + \exp(-z))$
- Softmax (for mapping to probability simplex): $\sigma(z) = [\exp(z_j)/\sum_j \exp(z_j)]$
- Other alternatives can be determined by the task, i.e., the form of the target vector y

Hidden activation functions

- Sigmoid (usually hyperbolic tangent; less common in MLPs now): $\sigma(z) = \tanh z$
- ReLU ("rectified linear unit"; larger gradients): $\sigma(z) = \max\{0, z\}$
- Leaky ReLU (gradient everywhere nonzero; not smooth): $\sigma(z) = \max\{0, z\} + \epsilon \min\{0, z\}$
- Many other alternatives



There is even less theoretical guidance for choosing the depth and width(s) of an MLP.

- In practice, these are often simply dictated by computational considerations such as memory and the complexity of operations.

One relevant and well-known result is known as the *universal approximation theorem*:

A feedforward neural network with linear output and at least one hidden layer of width d can approximate any function and its derivatives to arbitrary accuracy, as $d \to \infty$.

- The theorem guarantees that a large-enough MLP can represent any desired function from data to target, but not that it can actually learn the function.
- There is no guidance on how large the MLP should be for a given function.

However, empirical studies show that prioritising depth in a neural network can actually reduce the total number of neurons required to represent a given function.

- Consider two MLPs: one with n hidden layers of width $N \gg n$, and one with N hidden layers of width n. With all other factors equal, the second deeper network is more likely to perform better in terms of both representation and generalisation accuracy.

Training and Validating Neural Networks

Loss functions

Given some set of training examples (x, y), training an artificial neural network involves minimising a suitable loss function over the parameters of the network. In the case of an MLP, the parameters are the weight matrices W_i and the bias vectors B_i.

As in the case of generalised linear models, neural networks can be fitted through the maximum-likelihood principle, which gives a loss function similar to Eq. (221):

$$L := \langle -\ln p_{\text{model}}(y|\hat{y}(x)) \rangle, \tag{265}$$

where p_{model} is the probability of $y|x$ as described by the network, and $\langle \cdot \rangle$ denotes the average over the training set ($\approx$ the expectation under the true distribution p_{data} of $y|x$).

This general expression allows a unique loss function to be derived for any given network/task, and also subsumes the most common loss functions used in deep learning:

- For classification, $\hat{y}$ usually describes a PMF over k classes, and the maximum-likelihood loss reduces to the cross-entropy loss for classification (Eq. (205)):

$$L = \left\langle -\sum_i^k y_i \ln \hat{y}_i(x) \right\rangle \tag{266}$$

  with y in one-hot form, since

$$p_{\text{model}}(y|x) \propto \prod_i^k \hat{y}_i(x)^{y_i}. \tag{267}$$

- For regression, $\hat{y}$ usually describes the conditional mean of $y|x$ under p_{model}. If $p_{\text{model}} = \mathcal{N}(\hat{y}, \sigma^2 I)$, the maximum-likelihood loss reduces to the L2 loss (Eq. (210)):

$$L = \left\langle |y - \hat{y}(x)|^2 \right\rangle, \tag{268}$$

  where y is now allowed to be vector-valued.

Apart from the maximum-likelihood principle, one may also use loss functions that provide convergence to conditional properties of p_{data} itself (as approximated by the training set):

- The conditional median of $y|x$ under p_{data} is learnt by minimising the L1 loss

$$L = \langle ||y - \hat{y}(x)||_1 \rangle. \tag{269}$$

- The conditional mean of $y|x$ under p_{data} is learnt by minimising the L2 loss.

Exercise

Show that $\hat{y}(x; \theta) = \langle y|x \rangle$ minimises the L2 loss.

Backpropagation

With the loss function $L(W_i, B_i)$ specified, it can in principle be minimised using any optimisation algorithm. However, methods that utilise the loss gradients $\partial_{(W_i,B_i)} L$ are almost always used, because gradient information is readily obtainable for most network architectures.

- One extremely popular example is the stochastic gradient-based algorithm ADAM (adaptive moment estimation)

The training algorithm begins with a forward "propagation" of the training set through the network: given a set of training examples (x, y) and some initialisation values for (W_i, B_i), the set of predictions $\hat{y}$ and thus L is computed (along with all intermediate quantities (z_i, a_i)). Backpropagation (aka backprop) is then performed to compute the gradients $\partial_{(W_i,B_i)} L$, which are used to update the values (W_i, B_i) for the next training *epoch* (iteration).

Backprop is simply reverse-accumulation autodiff. First consider the gradient (scalar-by-vector derivative) of L with respect to z_D in the output layer, which can be computed as:

$$\partial_{z_D} L = \partial_{a_D} L \cdot \partial_{z_D} \sigma_D . \tag{270}$$

- $\partial_{a_D} L$ is the scalar-by-vector derivative of L with respect to $a_D = \hat{y}$ (analytically known), evaluated at the current value of a_D (computed in forward pass)
- $\partial_{z_D} \sigma_D$ is the vector-by-vector derivative of σ_D with respect to z_D (analytically known), evaluated at the current value of z_D (computed in forward pass)

With $\partial_{z_D} L$ in hand, the gradient of L with respect to z_i in each of the hidden layers can then be computed recursively as:

$$\partial_{z_i} L = \partial_{z_{i+1}} L \cdot \partial_{z_i} z_{i+1} = \partial_{z_{i+1}} L \cdot (W_{i+1} \cdot \partial_{z_i} \sigma_i), \tag{271}$$

again using the current values of W_{i+1} and z_i.

Finally, we may compute the desired gradients of L with respect to the weights and biases of the network. The former "gradients" are really scalar-by-matrix derivatives (i.e., $\partial_{W_i} L$ is a $d_i \times d_{i-1}$ matrix), and can be computed as:

$$\partial_{W_i} L = \partial_{z_i} L \otimes a_{i-1}, \tag{272}$$

where $\otimes$ denotes the outer product. The latter gradients are given simply by:

$$\partial_{B_i} L = \partial_{z_i} L. \tag{273}$$

Note:

- There is a suppressed dimension in the backprop equations above, since the (z_i, a_i) are actually arrays of vectors for all the training examples; more generally, they can have arbitrary dimensionality and are referred to as *tensors* (not the maths/physics definition!)
- The equations explicitly highlight the role of the activation function in gradient-based optimisation, e.g., a Heaviside activation is unsuitable as the gradients vanish everywhere

Training and validation workflow

There are many considerations that go into training and validating an artificial neural network (again, it can be more art than science); only some of the essential ones are presented here.

Training, validation and test sets

- All of the available data should be partitioned into at least a training set (examples that the network is trained on) and a test set (examples that the network is assessed on, and has never seen during the training stage).
- Another common practice is to add a validation set to the partition (examples that are used to tune the hyperparameters of the network and training procedure).
- Confusingly, a validation set is sometimes used not for tuning, but to assess a preliminary network before being used to train a final network. In this scenario, it can be viewed as the test set for the preliminary network, or part of the training set for the final network. The actual test set might then be dropped, depending on the context.
- A reasonable split is 80% training, 10% validation and 10% test — but there is plenty of freedom here. Again, it depends on the context.

Preprocessing of data

- Normalise the data if appropriate, e.g., transform to location ≈ 0 and scale ~ 1 (this generally lowers variance and improves learning)
- Randomly shuffle the training examples (this is relevant because of batching).

Initialisation of parameters

- The initial values for (W_i, B_i) should be normalised as well. The aim is again variance reduction, but also to mitigate issues such as vanishing or exploding gradients.
- A common practice is to initialise the weights in each layer randomly (uniform or normal) with location ≈ 0 and scale $\approx 1/\sqrt{d_i}$, and to initialise the biases to zero.

Training hyperparameters

- Apart from the hyperparameters of the network, there are also several key quantities in the training procedure that need to be tuned.
- Note that all of the loss functions for neural networks are essentially sums over the training examples. One may then consider sums over subsets of the training examples (which are different estimates of the same expectation), and iterate over them; this is called *batching* or minibatching. At the other extreme, one may also iterate over individual examples.
- Tuning the batch size is crucial. Large batch sizes are more efficient as they lead to more accurate estimates of the loss gradients, but can be limited by memory issues. Small batch sizes lead to noisier and thus slower training, but can have a regularising effect.
- The *learning rate* is also important — it is basically the step size for the optimiser. Some methods like ADAM have adaptive rates, but the initial rate may still need to be tuned.

Example

Jupyter notebook: Train an MLP classifier on the iris flower data set.

Regularisation techniques

Artificial neural networks are high-capacity models with many d.o.f., so they are notoriously prone to overfitting when training data is limited. There are many techniques to combat this at all stages of the workflow; some important examples are described here.

Weight decay

- Add a regularisation term like $|W|^2$ (as in ridge regression) or $||W||_1$ (as in lasso regression), where W denotes all the weights to be suppressed in size; e.g., the weights in the output layer would not generally be suppressed in a regression problem.

Early stopping

- When the validation loss reaches a minimum and starts to rise again, stop training and return the network (parameters) to its state at the minimum.

Dropout

- Randomly "turn off" some number of neurons at each iteration, i.e., set their outputs to zero (nodes in the input layer can be included as well). This can be regularising as it lowers the instantaneous capacity of the network, and adds noise to the training.

Data augmentation

- Artificially increase the number of training examples by transforming the data in ways that the network should be invariant to (e.g., translating/rotating/scaling images that are to be classified), or by adding different noise realisations.

Beyond Multilayer Perceptrons

Autoencoders

An autoencoder learns a (typically compressed) representation $z \in \mathbb{R}^m$ for data $x \in \mathbb{R}^n$, by attempting to generate recreated data $\tilde{x} \in \mathbb{R}^n$ from z.

- It is effectively two functions: the encoder $z = E(x)$ and the decoder $\tilde{x} = D(z)$
- The representation z is often referred to as the code, or as latent variables
- Applications include dimensionality reduction, information retrieval, image denoising

Undercomplete autoencoders ($m < n$):

- These are the archetypal example of an autoencoder. They can be trained with just a simple loss function that penalises discrepancy between x and $\tilde{x}$, e.g., $L = \langle |x - \tilde{x}|^2 \rangle$.
- If the decoder is fully linear and the loss function is the L2 loss, an undercomplete autoencoder actually learns the subspace spanned by the principal components in PCA.
- In principle, one might imagine a network with enough capacity to map every training example to a different point on the real line, and vice versa. This might be great for compression, but not so much for learning useful representations.

Overcomplete autoencoders ($m \geq n$):

- With just a discrepancy-type loss, these are likely to simply learn the identity function. However, they can still be trained with regularisation to learn useful representations.
- This is done by adding a loss term that imposes constraints on z directly, e.g., $\langle |z|^2 \rangle$.

Convolutional neural networks (CNNs)

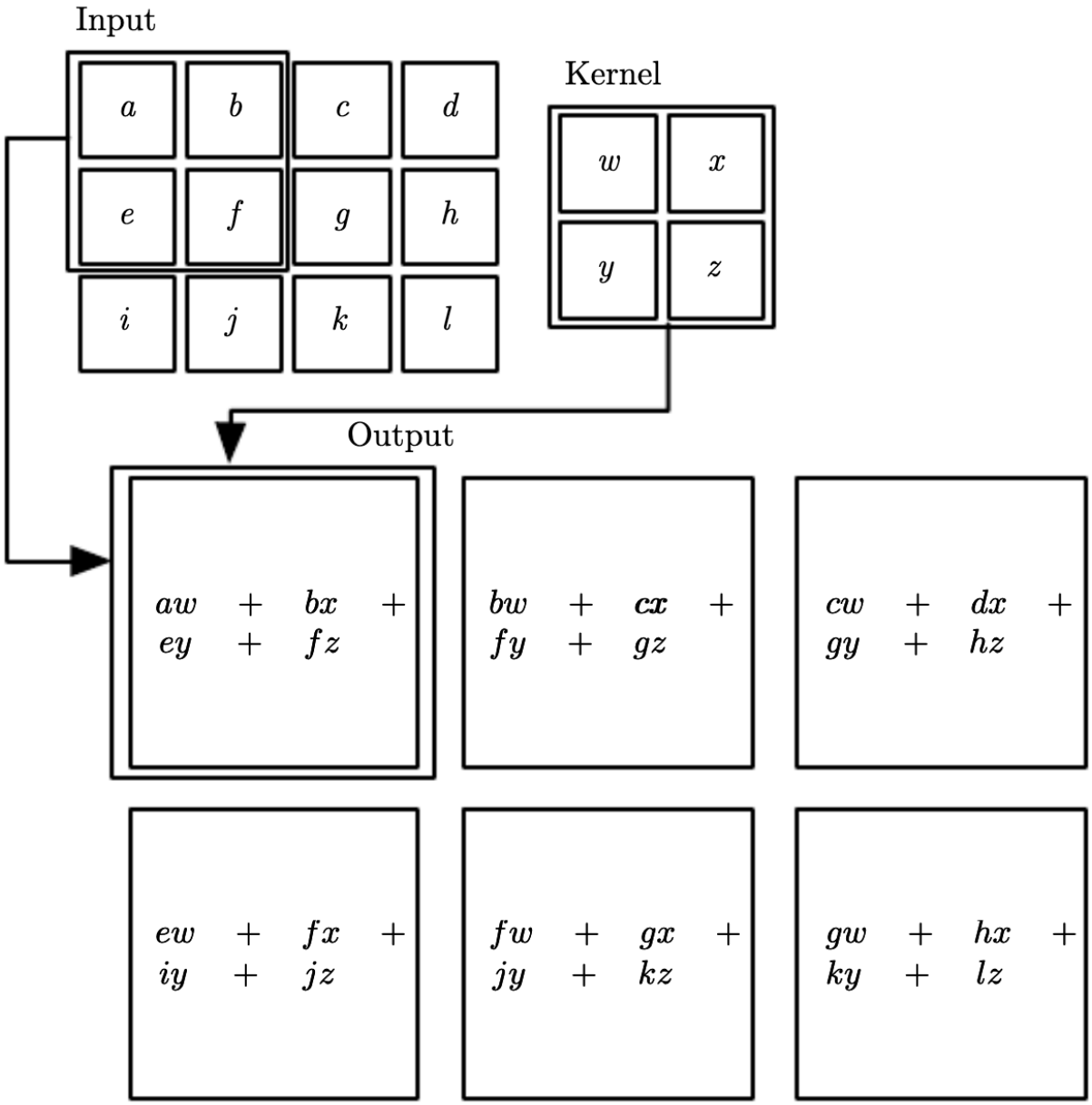
A CNN is a specialised network architecture for learning grid-like data (e.g., time series in one dimension, or images in two dimensions), by leveraging spatial correlations in the data. It is inspired by human vision, and is the most successful example of biologically inspired learning.

The essence of a CNN is the *convolutional layer*, which contains three types of operations.

1. Convolution stage: An affine transformation that acts only on a small subset of the layer input is applied across (convolved with) the entire input. The transformation is called a convolution kernel, and its weights and biases are the layer parameters.
2. Detector stage: This is just the application of a nonlinear, component-wise activation function (typically ReLU or its variants) to the result of the convolution stage.
3. Pooling stage: The result of the detector stage is split into smaller subsets and down-sampled according to some rule (commonly *max pooling*, which takes the maximum of each subset). The subsets can overlap. This keeps the network small, and helps to make the layer output invariant to small translations of the layer input.

Note:

- The convolution operation is actually cross-correlation, if the kernel is not "flipped" (which gives nice formal properties). There is often no practical difference.
- The different stages are commonly treated as layers themselves, allowing further customisation. CNNs also often contain fully connected layers (as in MLPs).
- All operations are differentiable, so the loss functions for training MLPs also apply here.



Credit: Goodfellow et al., 2016

Recurrent neural networks (RNNs)

An RNN is a specialised network architecture for learning sequence-like data (e.g., sentences, or general time series) by mimicking memory — i.e., accounting for arbitrarily long-term dependencies of any point in the sequence on its history up to that point. It plays an important role in natural language processing (speech/character recognition, translation, ChatGPT, etc.).

The key to incorporating "memory" in the architecture is to include a hidden "layer" h that is itself a sequence. In a fully recurrent network, each node $h^{(t)}$ in h is connected not just to the input $x^{(t)}$ at the corresponding point in the sequence, but also the previous node $h^{(t-1)}$.

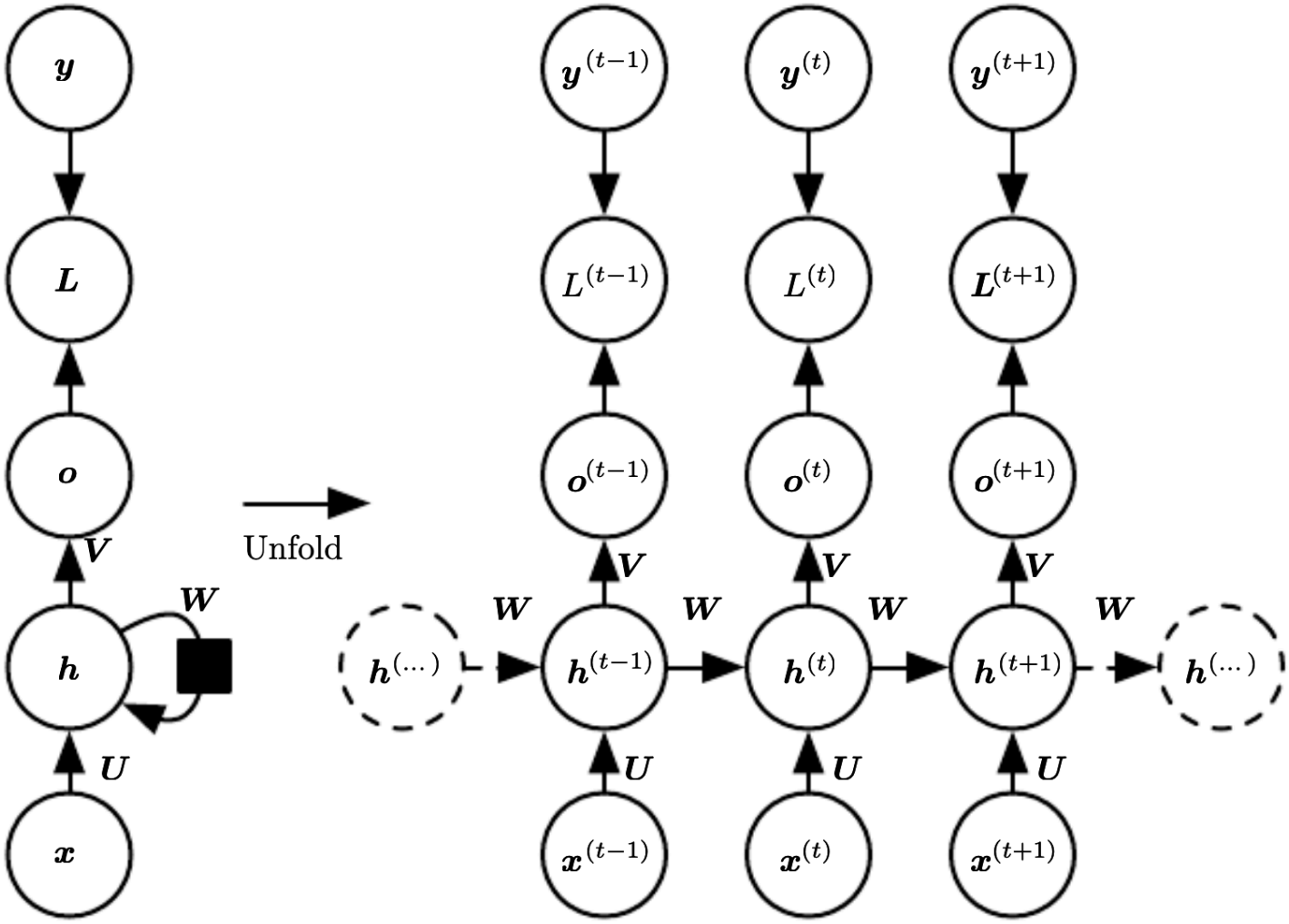
- These connections are again built out of affine transformations and activation functions:

$$h^{(t)} = \sigma\left(Ux^{(t-1)} + Wh^{(t-1)} + B\right). \tag{274}$$

- The output of the RNN is then the sequence

$$\hat{y}^{(t)} = \sigma_{\text{out}}\left(Vh^{(t)} + C\right), \tag{275}$$

  whose discrepancy from the corresponding training target $y^{(t)}$ is described by the loss function $L^{(t)}$ and used to train the network.
- As in CNNs, where a common kernel is applied across all locations, RNNs also share (U, V, W) across all times; this keeps the network small and improves generalisation.
- There are many RNN variants. A well-known one is *long short-term memory*, which additionally learns when to "forget" dependencies.



Credit: Goodfellow et al., 2016