TWEAG

Bayesian data analysis with TensorFlow Probability

www.tweag.io

Your hosts

Simeon (presentation)

- background in computational biology
- Data Scientist at Tweag since 2019
- lives in Paris

Dorran (interactive demos & questions in chat)

- previous positions in geophysics
- Data Scientist at Tweag since 2019
- lives in Zurich

Tweag I/O

Software innovation lab and consultancy based in Paris with employees all around the world and a strong focus on open-source software.

We specialize in

- software engineering, with a focus on functional programming
- DevOps, with a focus on reproducible software systems and builds
- data science

Key industries: finance, biotech, automotive

Need help with your project? Want to work with us?

www.tweag.io

hello@tweag.io

What you're in for

This tutorial consists of alternating blocks of

- theory / example slides
- practical examples on either external websites or Google Colab notebooks. Links at

`https://github.com/tweag/tutorial-dsc-2020`

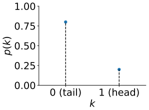
Requirements:

- a Google account (for the hands-on demos)
- elementary knowledge in probability theory and statistics

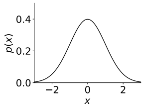
Reminder: Probabilities

Probability distributions can be...

discrete: $\text{Bernoulli}(\text{k}; \text{b}) = b^k (1-b)^{1-k}$



continuous: $\mathcal{N}(\text{x}; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{1}{2\sigma^2}(x-\mu)^2)$



Important concepts:

Conditional probability

$p(A|B)$: probability that A is true, given B is true

Joint probability

$p(A, B)$: probability that both A and B are true

Conditional joint probability

$p(A, B|C)$: probability that both A and B are true, given C is true

Bayesian vs frequentist probabilities

Example: fair coin flip with (bias $b = \frac{1}{2}$)

Frequentist probability

$p(\text{"head"}|b = \frac{1}{2})$:

$\frac{\text{\# of heads}}{\text{\# total flips}}$ for ∞ many fair coin flips

Bayesian probability

$p(\text{"head"}|b = \frac{1}{2})$:

measure of *belief* in the statement “flip results in head” given a fair coin

Prior beliefs

Assume: unknown bias b

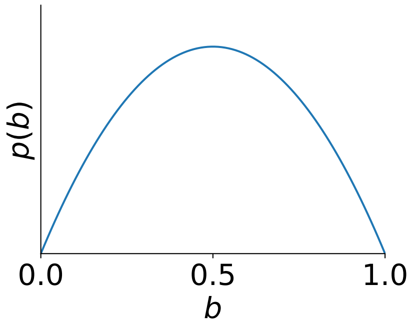
Prior probability

Encodes prior belief in b *before* flipping the coin

What is known about b?

- b is a probability: $0 \le b \le 1$
- most coins are fair

$\rightarrow$ choose prior distribution defined between 0 and 1, with maximum at and symmetric around $b = \frac{1}{2}$



Example:

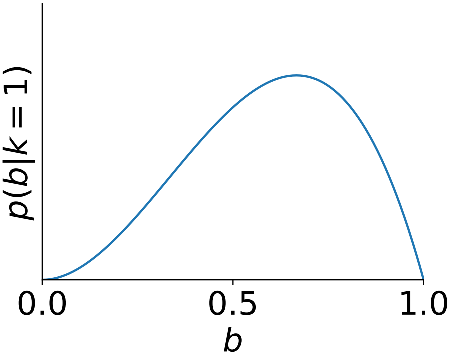
$$b \sim \text{Beta}(\alpha = 2, \beta = 2)$$

Posterior belief

Now: flip coin one time, result: head

Posterior belief

$p(b|\text{head})$: updated prior belief after obtaining new data



Update rule

Bayes' theorem

$$p(A|B) = \frac{p(B|A) \times p(A)}{p(B)}$$

(easily derived from rules for conditional probabilities)

In data analysis:

$$\underbrace{p(x|D,I)}_{\text{posterior}} = \underbrace{p(D|x,I)}_{\text{likelihood}} \times \underbrace{p(x|I)}_{\text{prior}} \; / \; \underbrace{p(D|I)}_{\text{evidence}}$$

x: model parameter (in our case: b)

D: data (in our case: $k = 1$)

I: prior information (often omitted to simplify notation)

Likelihood

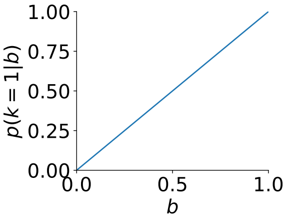
$p(D|x)$: probability of the data given fixed model parameters
$\rightarrow$ models data-generating process

In our case:

$$p(k|b) = \text{Bernoulli}(k; b) = b^k(1-b)^{k-1}$$

with

$$k = \begin{cases} 0: & \text{tail} \\ 1: & \text{head} \end{cases}$$



Evidence

$p(D) = \int \mathrm{d}x\, p(D|x)p(x)$:
normalization constant (long story...)

In our case:

$$
\begin{aligned}
p(k=1) &= \int_0^1 \mathrm{d}b\, L(k=1|b)p(b) \\
&= \int_0^1 \mathrm{d}b\;\; \mathrm{Bernoulli}(k;b) \times \mathrm{Beta}(k;\alpha=2,\beta=2)\big|_{k=1} \\
&= \int_0^1 \mathrm{d}b\;\; b^k(1-b)^{k-1}\frac{b(1-b)}{\frac{\Gamma(2)\Gamma(2)}{\Gamma(4)}}\Bigg|_{k=1} \\
&\;\vdots \\
&= \frac{1}{2}
\end{aligned}
$$

Evidence

$p(D) = \int \mathrm{d}x\, p(D|x)p(x)$:

normalization constant (long story...)

In our case:

$$
\begin{aligned}
p(k=1) &= \int_0^1 \mathrm{d}b\, L(k=1|b)p(b) \\
&= \int_0^1 \mathrm{d}b\ \ \text{Bernoulli}(k;b) \times \text{Beta}(k;\alpha=2,\beta=2)\big|_{k=1} \\
&= \int_0^1 \mathrm{d}b\ \ b^k(1-b)^{1-k} \frac{b(1-b)}{\frac{\Gamma(2)\Gamma(2)}{\Gamma(4)}}\Bigg|_{k=1} \\
&\vdots \\
&= \frac{1}{2}
\end{aligned}
$$

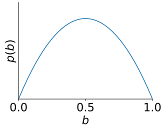
YIKES

Update rule

In our coin flip example:

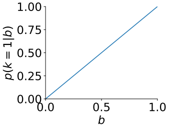
prior:

$$p(b) = \text{Beta}(b; \alpha = 2, \beta = 2) \\ \propto b(1 - b)$$



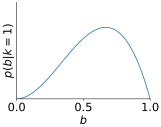
likelihood:

$$p(D|b) = \text{Bernoulli}(k = 1; b) \\ = b$$



posterior:

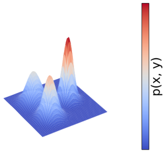
$$p(b|D) \propto p(D|b) \times p(b)/p(D) \\ = \text{Beta}(b; \alpha = 3, \beta = 2) \\ \propto b^2(1 - b)$$



– no slides for interactive stuff –

Real-world Bayesian data analysis

In real problems: non-standard, difficult posterior distributions



Probabilistic programming libraries

Allow to

- programmatically define a statistical model
- sample from arbitrary posterior distributions
- run quality checks

Examples:

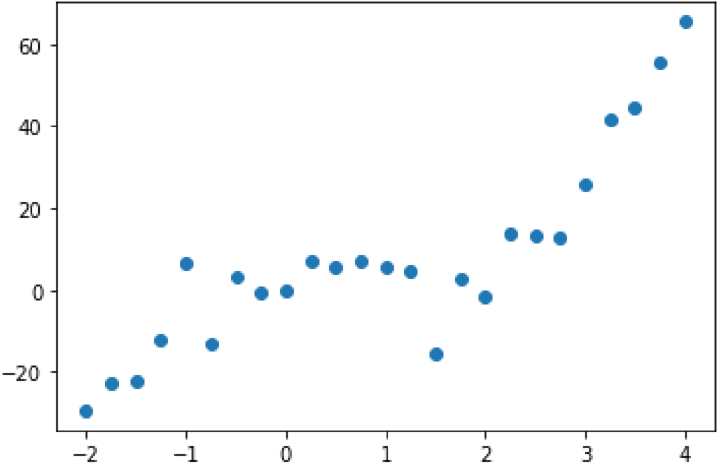
- PyMC3
- Stan
- TensorFlow Probability
- ...

Real-world Bayesian data analysis: recipe

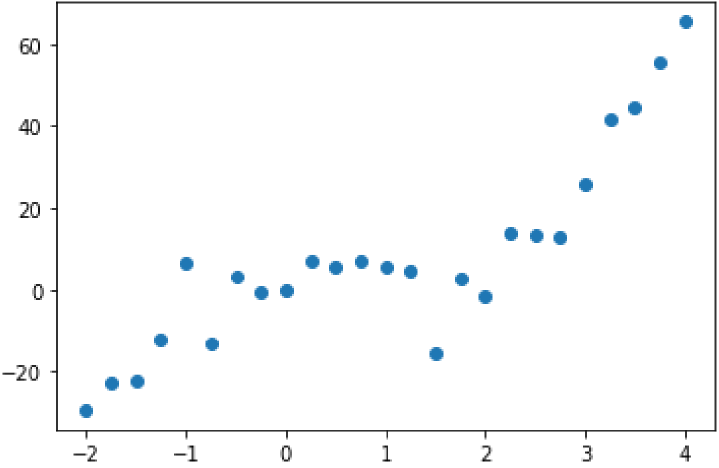
Steps to solve real-world problems:

0. Look at and think about data:
   - What kind of process generated data?
   - What do I want to learn: clusters? Approximating function? class labels?
1. formulate likelihood (statistical model for data-generating process)
2. formulate prior distributions for likelihood parameters
3. set up model in probabilistic programming library
4. sample from posterior distribution
5. perform quality checks

Step 0: formulating the problem and understanding the data

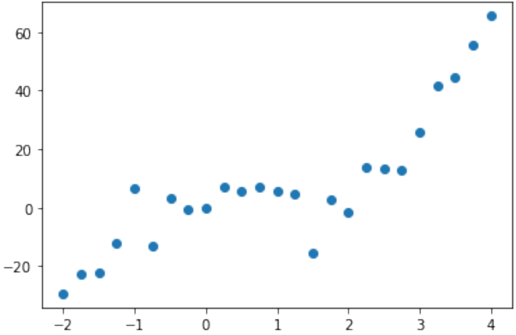


Step 0: formulating the problem and understanding the data



Let’s do polynomial regression!

Step 1: formulating the likelihood



Often: measured data = idealized data + noise

idealized data: $\hat{f}(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$

noise: normally distributed

Likelihood parameters: $\vec{\beta}, \sigma$

Step 2: formulating prior distributions

Incorporate all available information

- previous experiments
- ballpark estimation from the data
- common sense

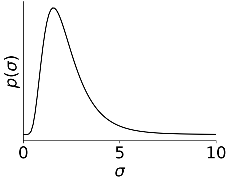
Warning

Avoid introducing strong bias!

Noise σ:

- positive quantity $\rightarrow$ $p(\sigma)$ with positive support
- from data: not too big

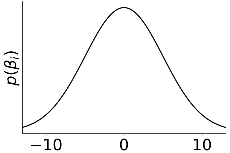
e.g., $\sigma \sim$ Lognormal$(\mu = 2, \sigma = \frac{1}{2})$



Coefficients β:

- not too extreme values ¯_(ツ)_/¯

Something broad and “uninformative”, e.g. $\mathcal{N}(0, 5)$



Step 3: code model in probabilistic programming library

– see interactive demo –

Step 4: sample with appropriate inference algorithm

Most popular and powerful class: Markov chain Monte Carlo (MCMC)

Continuous parameters:

Hamiltonian Monte Carlo (HMC) very efficient

Discrete parameters:

e.g., Metropolis-Hastings

Combination of both:

Gibbs sampling

Another popular option: variational inference (VI)

MCMC introduction blog posts:

`www.tweag.io/blog`

Approximation workhorse: Metropolis-Hastings

Markov chain

Random process with

$$p(x_{i+1}|x_i, x_{i-1}, \ldots, x_1) = p(x_{i+1}|x_i)$$

$\rightarrow$ a Markov chain has no “memory”

In some conditions: converges to a unique invariant distribution $\pi(x)$

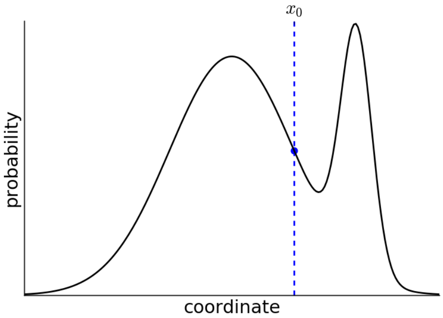
Metropolis-Hastings algorithm

Construct Markov chain with invariant distribution $\pi(x) = p(x)$:

1. starting at state, x_i, propose a new state x^*_{i+1} from $q(x^*_{i+1}|x_i)$
2. calculate acceptance probability p_{acc}
3. draw $u \sim \mathcal{U}(0, 1)$
4. if $u < p_{\text{acc}}$: $x_{i+1} = x^*_{i+1}$, else $x_{i+1} = x_i$

Metropolis (-Hastings)

Initialize with any state x_0



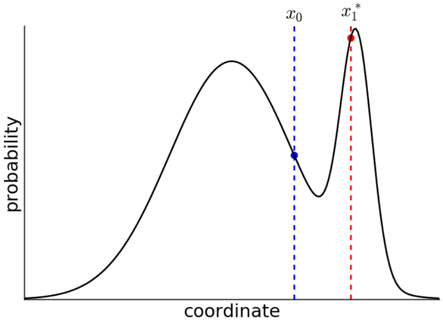
Sequence of states:
(x_0)

Metropolis et al., J. Chem. Phys (1953); Hastings, Biometrika (1970)

Metropolis (-Hastings)

Initial state: x_0

1. calculate a proposal state x_1^* by randomly perturbing x_0



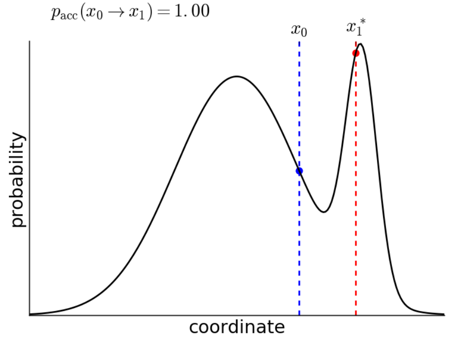
Metropolis et al., J. Chem. Phys (1953); Hastings, Biometrika (1970)

Metropolis (-Hastings)

Initial state: x_0

1. calculate a proposal state x_1^* by randomly perturbing x_0
2. calculate acceptance probability

$$p_{\text{acc}} = \min\left(1, \frac{p(x_1^*)}{p(x_0)}\right)$$



Metropolis et al., J. Chem. Phys (1953); Hastings, Biometrika (1970)

Metropolis (-Hastings)

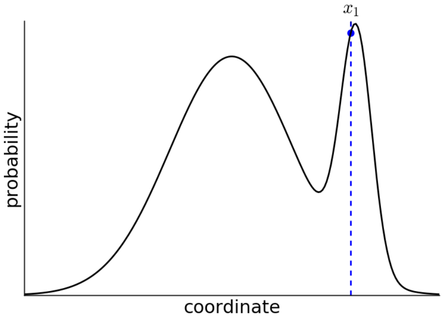
Initial state: x_0

1. calculate a proposal state x_1^* by randomly perturbing x_0
2. calculate acceptance probability

$$p_{\mathrm{acc}} = \min\left(1, \frac{p(x_1^*)}{p(x_0)}\right)$$

3. with probability p_{acc}, accept proposal state x_1^* as the next state x_1, else copy x_0

Sequence of states:
(x_0, x_1)

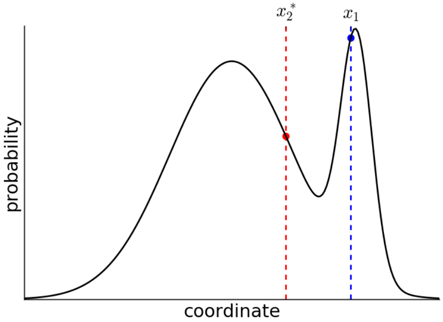


Metropolis et al., J. Chem. Phys (1953); Hastings, Biometrika (1970)

Metropolis (-Hastings)

Current state: x_1

1. calculate a proposal state x_2^* by randomly perturbing x_1



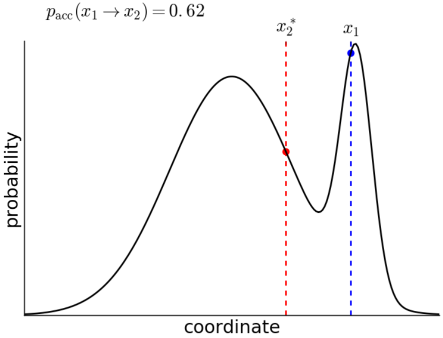
Metropolis et al., J. Chem. Phys (1953); Hastings, Biometrika (1970)

Metropolis (-Hastings)

Current state: x_1

1. calculate a proposal state x_2^* by randomly perturbing x_1
2. calculate acceptance probability

$$p_{\text{acc}} = \min\left(1, \frac{p(x_2^*)}{p(x_1)}\right)$$



Metropolis et al., J. Chem. Phys (1953); Hastings, Biometrika (1970)

Metropolis (-Hastings)

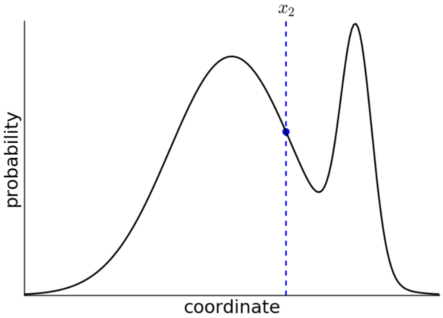
Current state: x_1

1. calculate a proposal state x_2^* by randomly perturbing x_1
2. calculate acceptance probability

$$p_{\text{acc}} = \min\left(1, \frac{p(x_2^*)}{p(x_1)}\right)$$

3. with probability p_{acc}, accept proposal state x_2^* as the next state x_2, else copy x_1

Sequence of states:
(x_0, x_1, x_2)



Metropolis et al., J. Chem. Phys (1953); Hastings, Biometrika (1970)

Metropolis (-Hastings)

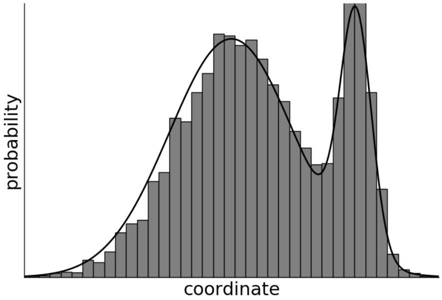
Current state: x_1

1. calculate a proposal state x_2^* by randomly perturbing x_1
2. calculate acceptance probability

$$p_{\text{acc}} = \min\left(1, \frac{p(x_2^*)}{p(x_1)}\right)$$

3. with probability p_{acc}, accept proposal state x_2^* as the next state x_2, else copy x_1

Sequence of states:
$(x_0, x_1, x_2, \ldots, x_n)$



Metropolis et al., J. Chem. Phys (1953); Hastings, Biometrika (1970)

Step 5: quality checks & debugging

How to check whether sampling was successful?
Effective sample size, $\hat{R} \rightarrow$ interactive demo

Model debugging strategy:

Prior predictive check

Use prior samples to generate data points y_k:

$$p(y_i) = \int \mathrm{d}\vec{\beta}\mathrm{d}\sigma \, L(y_k|\vec{\beta},\sigma)p(\vec{\beta})p(\sigma)$$

$\rightarrow$ interactive demo

Algorithm:

1. draw samples from prior
2. for each sample β_i, σ_i, you get a likelihood $L(y_k|\beta_i,\sigma_i)$
3. draw samples from $L(y_k|\beta_i,\sigma_i)$
4. compare if new data points y_k look $\approx$ actual data