Introduction to Bayesian Inference with PyMC3

—

Lara Kattan

April 2021

Questions We'll (Begin to) Answer

- What is Bayesian estimation and inference?
- What is probabilistic programming?
- What is PyMC3 and how do I begin to use it?
  - GitHub link: https://github.com/larakattan/bayesian_inference_pymc3
  - Google colab alternative: tinyurl.com/pymc3-odsc21

Bayesian Inference

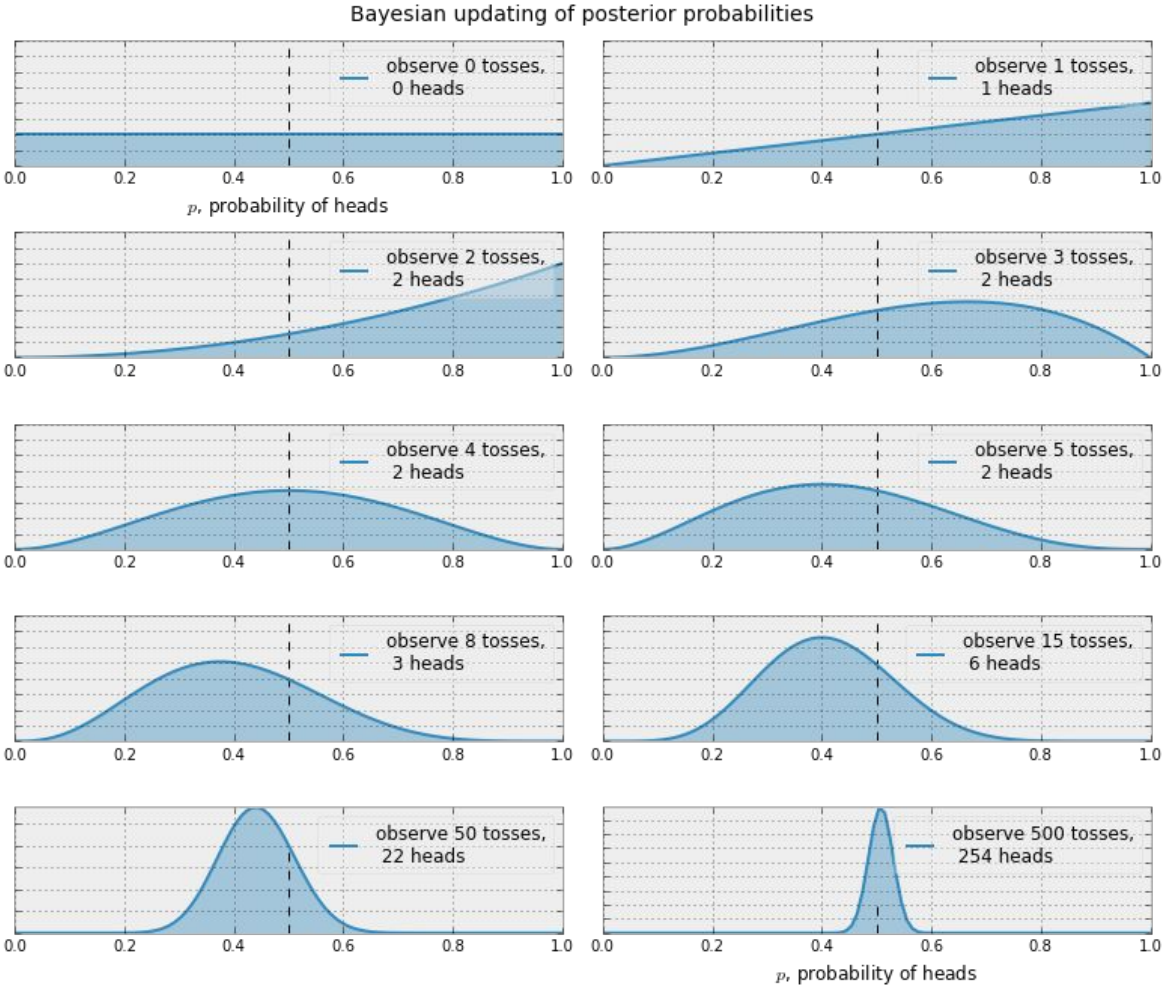
- Used to be called "inverse probability", now called the posterior distribution
  - Direct probability is now the likelihood (which is not a probability distro, technically)
  - Inverse prob: assign a probability distribution to an unknown variable
  - What's the most likely value of our parameter of interest, conditioning on the data we observe?
  - Reason from effects (observations) to causes (parameters)
- Outputs differ from traditional Frequentist statistics
  - Frequentist: point estimates of parameters and confidence intervals
  - Bayesian: posterior probability distributions on parameters
  - Why? Because what about things that happen only once; e.g. elections
  - Now: how do we go about getting these distributions?

Mechanics: Updating Beliefs with Evidence

- Example from *Bayesian Methods for Hackers:*
    - You know the code you right is likely to have some bugs somewhere. So you start by testing it against a really simple case; it passes. Continue to increase the complexity of the cases you test against. The more complex cases it passes, the more you're sure that the code is bug-free. You're already thinking Bayesian!
- Never 100% sure, same as in software testing, but we can get pretty close

Bayesian Inference

Updating Priors



Source: *Bayesian Methods for Hackers*, chapter 1

Bayesian Inference vs Frequentist

As we gather an *infinite* amount of evidence, say as N→∞, our Bayesian results (often) align with frequentist results. Hence for large N, statistical inference is more or less objective. On the other hand, for small N, inference is much more *unstable*: frequentist estimates have more variance and larger confidence intervals.

This is where Bayesian analysis excels. By introducing a prior, and returning probabilities (instead of a scalar estimate), we *preserve the uncertainty* that reflects the instability of statistical inference of a small N dataset.

https://nbviewer.jupyter.org/github/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/blob/master/Chapter1_Introduction/Ch1_Introduction_PyMC3.ipynb

Bayes' Theorem

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$\Rightarrow P(A \cap B) = P(B \cap A) = P(A|B)P(B) = P(B|A)P(A)$$

$$\Rightarrow P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Bayes' Theorem

Likelihood
(prev. direct probability)

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Bayes' Theorem

Prior

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Bayes' Theorem

$$P(A|B) = \frac{P(B|A)P(A)}{\underbrace{P(B)}_{\text{Evidence}}}$$

Bayes' Theorem

$$\underbrace{P(A|B)}_{\text{Posterior}} = \frac{P(B|A)P(A)}{P(B)}$$

Posterior
(prev. Inverse probability)

Probabilistic Programming

- General term for using the high-level programming language of your choice to define and estimate a probability model and run inference
  - Language needs to have random/stochastic events as primitives
  - We can treat the distributions as actual objects and investigate them (e.g. mean or std dev)
  - Primitives can be random variables (e.g. normal distribution) or a DGP (data generating process)
  - Programming language will abstract away complexities of writing/running models
- “Programming” here means writing code, not optimization
  - From Wikipedia: “Programming in this context [optimization] ... comes from the use of program by the United States military to refer to proposed training and logistics schedules, which were the problems Dantzig studied at that time”

Probabilistic Programming

- Been around for decades, with roots in WinBUGS (Bayesian inference Using Gibbs Sampling), first released in 1997
  - Still around in the form of OpenBUGS
- More transparent than a lot of ML models (e.g. random forests) because the modeler has control over the distributions
- Better than a simulation
  - Simulation moves in one direction: get data input and move it according to assumptions of parameters and get a prediction
  - Unknown params are not distributions
  - Bayesian modeling via PP adds another direction: use the data to go back and pick one of many possible parameters as the most likely to have created the data (posterior distros)
- A way to make Bayesian inference tractable (less math)

Probabilistic Programming

> [U]nlike a traditional program, which only runs in the forward directions, a probabilistic program is run in both the forward and backward direction.
>
> It runs forward -> to compute the consequences of the assumptions it contains about the world (i.e., the model space it represents), but it also runs backward <- from the data to constrain the possible explanations.
>
> In practice, many probabilistic programming systems will cleverly interleave these forward and backward operations to efficiently home in on the best explanations.

Cronin, Beau. "Why Probabilistic Programming Matters." 24 Mar 2013. Google, Online Posting to Google forum. 24 Mar. 2013. https://plus.google.com/u/0/107971134877020469960/posts/KpeRdJKR6Z1

Bayesian Inference via Probabilistic Programming

- Solving Bayes' theorem in practice requires taking integrals
- If we don't want to do that, we need to use numerical solution methods
  - Theano to the rescue!
- Lots of development in terms of new methods of sampling
  - Markov Chain Monte Carlo and Hamiltonian Monte Carlo
    - If some math satisfied (detailed balance equation), guaranteed to get the posterior distro in the limit
  - Then NUTS, No U-Turn Sampler, which removes having to specify a step size
    - Gelman: "NUTS uses a recursive algorithm to build a set of likely candidate points that spans a wide swath of the target distribution, stopping automatically when it starts to double back and retrace its steps"
  - Variational inference
    - Make distributions similar to each other
    - Optimization, not sampling, so more appropriate for big data

Solutions via Markov Chain Monte Carlo

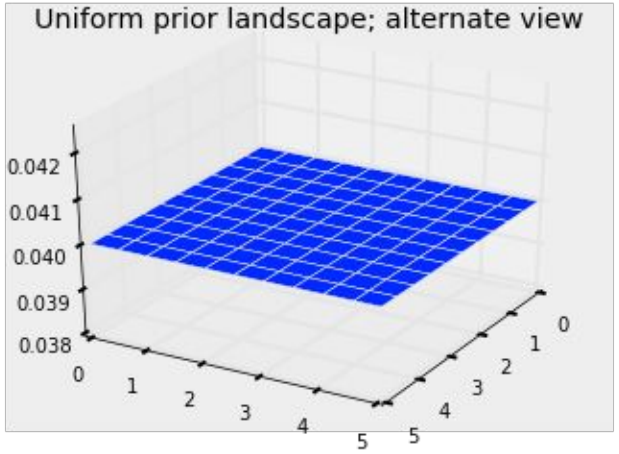
- From *Bayesian Methods for Hackers*:

When we setup a Bayesian inference problem with N unknowns, we are **implicitly creating an N dimensional space for the prior distributions to exist in**.

Associated with the space is an additional dimension, which we can describe as the *surface*, or *curve*, that sits on top of the space, that reflects the *prior probability* of a particular point. The surface on the space is defined by our prior distributions.

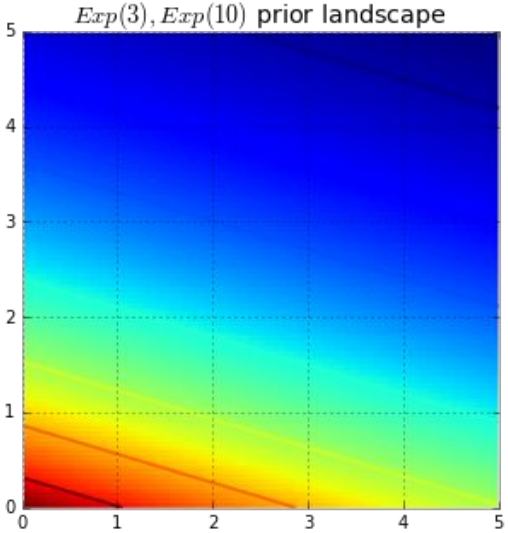
For example, if we have two unknowns p1 and p2, and priors for both are Uniform(0,5), the space created is a square of length 5 and the surface is a flat plane that sits on top of the square (representing that every point is equally likely).

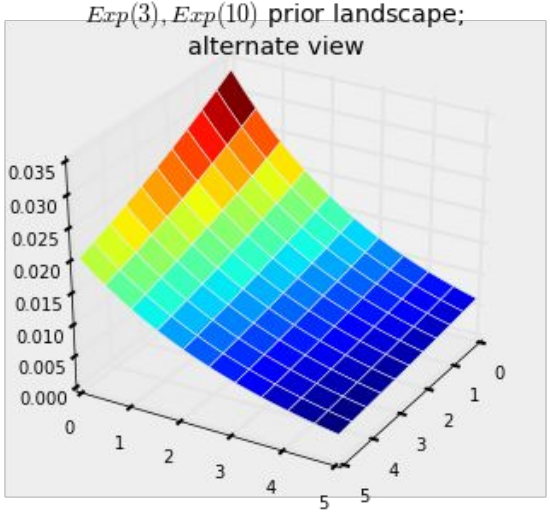
Solutions via Markov Chain Monte Carlo



Source:
https://nbviewer.jupyter.org/github/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/blob/master/Chapter3_MCMC/Ch3_IntroMCMC_PyMC3.ipynb

Solutions via Markov Chain Monte Carlo



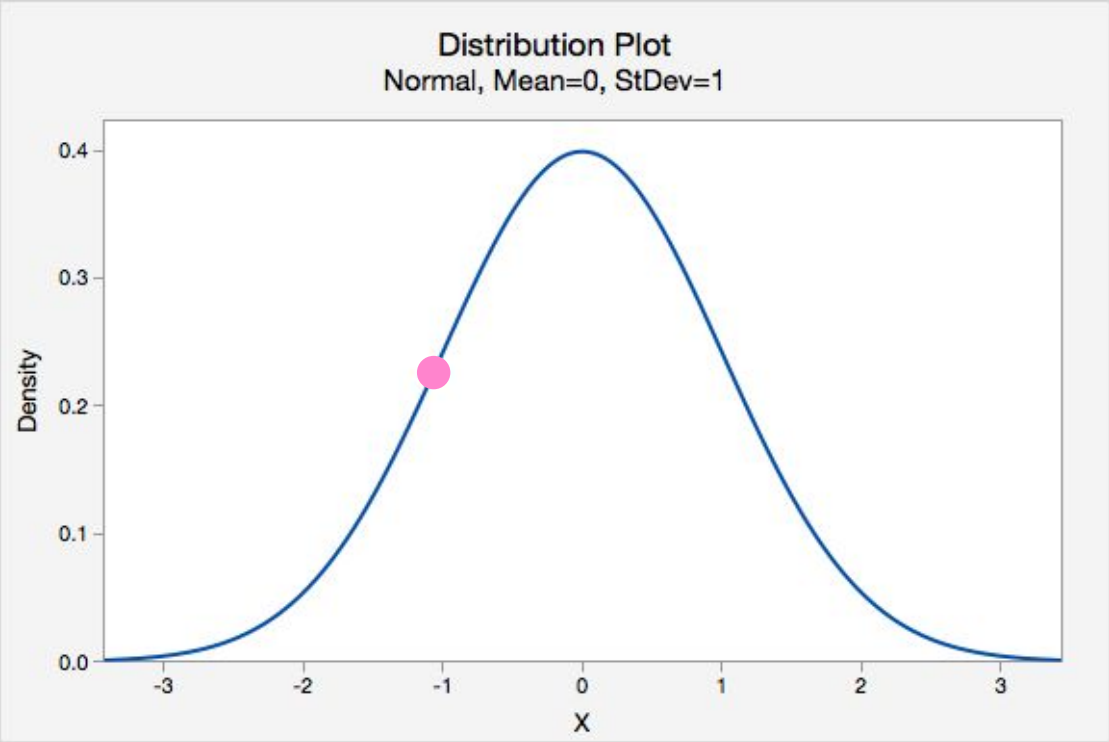


Source:
https://nbviewer.jupyter.org/github/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/blob/master/Chapter3_MCMC/Ch3_IntroMCMC_PyMC3.ipynb

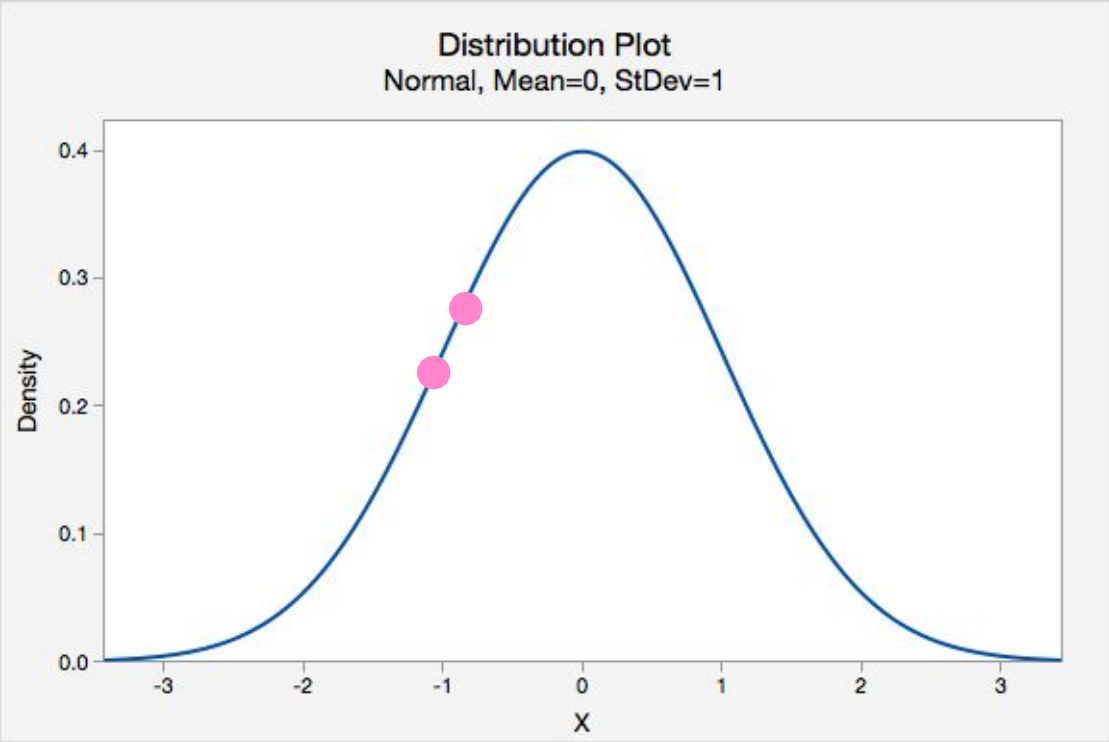
Solutions via Markov Chain Monte Carlo

- MCMC explores this space and returns to us samples from the posterior distribution
  - In PyMC3, samples are “traces”
- MCMC traverses the space, comparing the probability at that point, then moving to areas of higher probability

Solutions via Markov Chain Monte Carlo

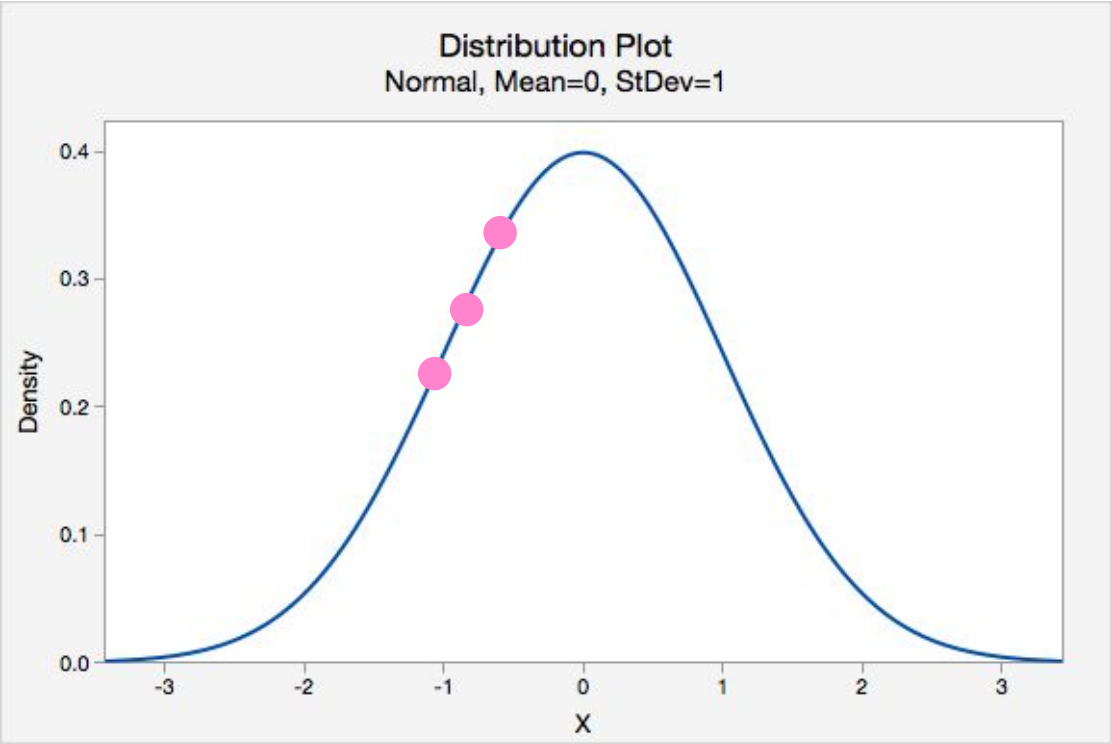


Solutions via Markov Chain Monte Carlo

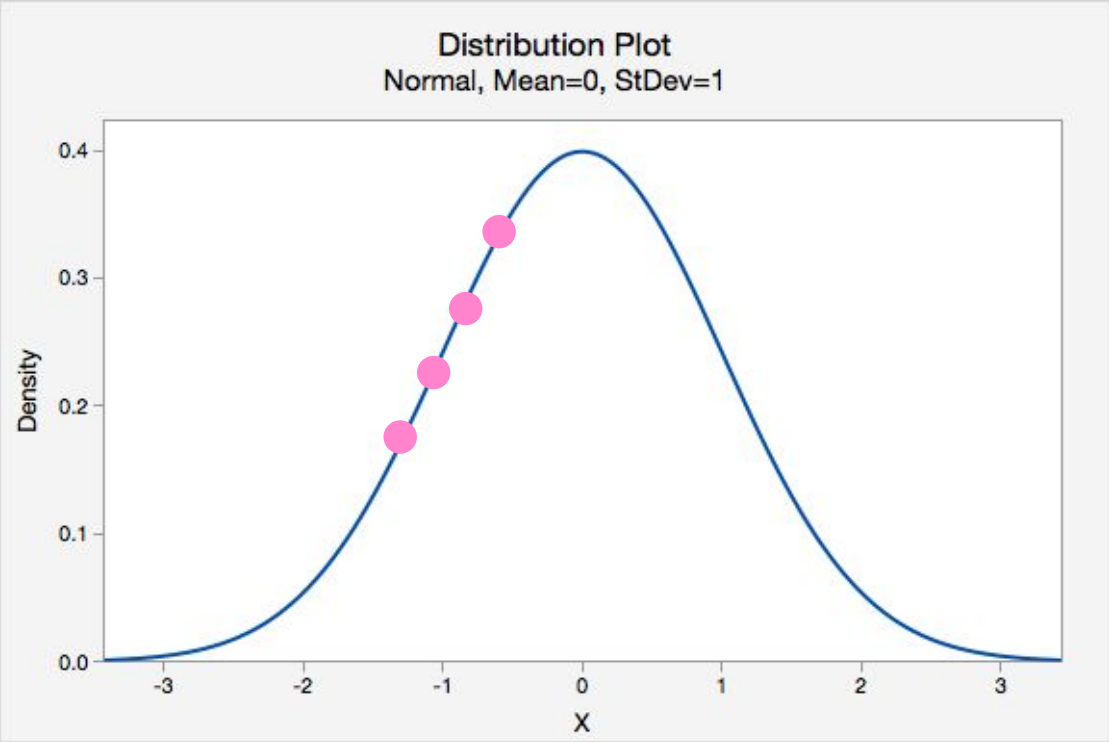


As you move to new point, either reject or accept based on prior and data; markov chain comes from the fact that we don't care about history: only prior position matters (memoryless)

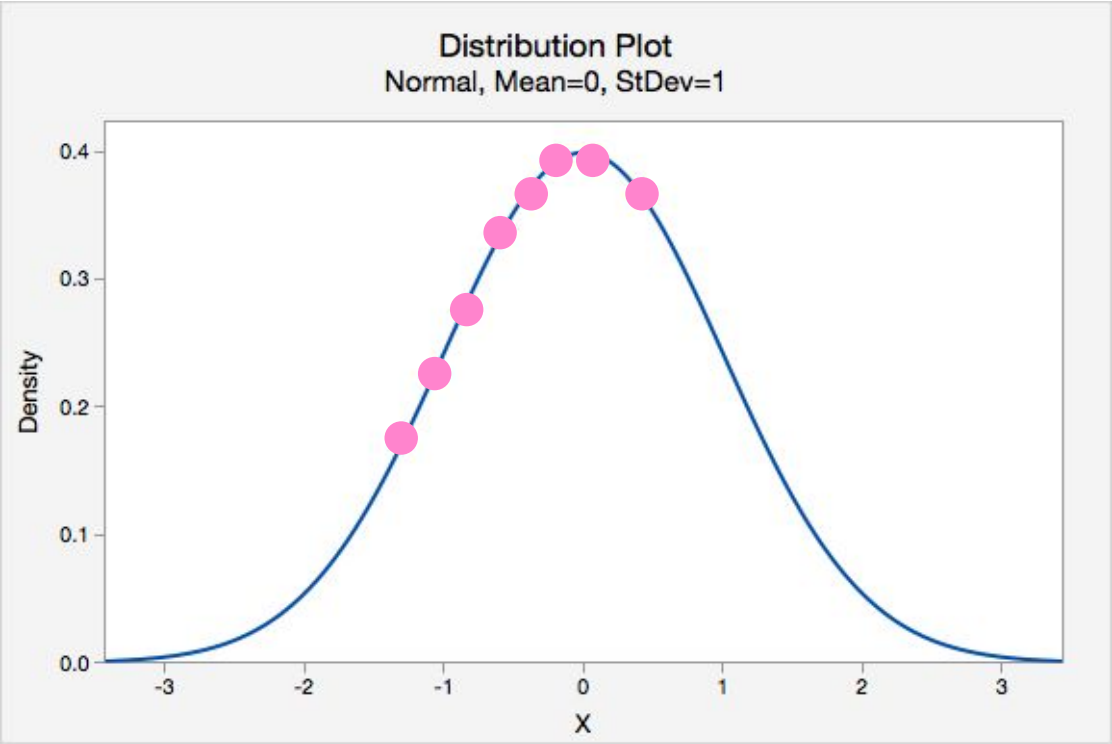
Solutions via Markov Chain Monte Carlo



Solutions via Markov Chain Monte Carlo



Solutions via Markov Chain Monte Carlo



Solutions via Markov Chain Monte Carlo

- Better than trying to analytically solve the N-dimensional distributions
- Can be used to reconstruct the shape of the distribution, not just the peak
- Do we need “burn-in” (to discard the first several hundred or so draws)?
  - Opinions differ! http://users.stat.umn.edu/~geyer/mcmc/burn.html
  - Instead of burn-in, use “better” starting values
- Many different types:
  - Gibbs sampling: given a multivariate distribution it is simpler to sample from a conditional distribution than to marginalize by integrating over a joint distribution
  - Hamiltonian -- uses information from gradient to take steps
  - NUTS (no u-turn sampler) -- eliminates need to set number of steps
- Downside: slow
  - Other approaches: variational inference, which is optimization, not simulation

PyMC3

- From the package authors: “PyMC3 is a new open source probabilistic programming framework written in Python that uses Theano to compute gradients via automatic differentiation as well as compile probabilistic programs on-the-fly to C for increased speed”
  - Theano: Python library for expressing math in terms of tensors (multi-dimensional arrays); fast

PyMC3 Examples

1. Fitting a distribution to data
2. Linear regression
3. Hierarchical linear regression
    a. Helpful for small data, especially when the number of observations per group is low (e.g. students per school)
    b. Not efficient to estimate each group individually... but if you pool, you lose idiosyncratic group-level information

Hierarchical Linear Regression

- Related to **fixed effects** and **random effects** models in traditional metrics
- Fixed effects are constant across groups, random effects vary (pretty much)
    - E.g. If you build a model with a fixed slope but random intercept for each group, you'd have a bunch of parallel lines with different intercepts

Mixed Effects Modeling

- Canonical example is often looking at student test scores:
  - Students have different achievement test scores
  - ... but, students within a school (group) have more similar test scores than across the population
  - You could estimate a student-level model, with a school effect (μ) and student-level epsilon
  - $$Y_{ij} = \mu_j + \epsilon_{ij} \qquad \epsilon_{ij} \sim N(0, \sigma^2)$$

    sigma^2 is how much each individual student deviates from the school mean
- We also know about the distribution of each school's mean:
  $$\mu_j \sim N(\mu, \sigma_\mu^2)$$ where mu is the overall population mean
- μ is the population param (fixed effect); σ^2 is the variance within a group (school); σ^2_μ is the between-group variance

Mixed Effects Modeling

- Notice that μ_j was the group-level (school) fixed effect in the prior model
- Now we can add a random effect for each student where g is the group-level effect (each school is a group) $\mu_j = \mu + g_j$
- Rewrite the student-level model as

$$Y_{ij} = \mu + g_j + \epsilon_{ij}$$

- This is our mixed effects model where μ is the fixed effect, g is the group-level random effect, and epsilon is individual random effects
- We estimate the following parameters: $\mu, \sigma^2, \sigma^2_\mu, \mu_j$
- Shrinkage: each group's parameters are pulled to the population mean

When do we use Bayesian inference?

- You want to incorporate prior beliefs into your model
- Little data, lots of parameters
  - If you had lots of data, you might use deep learning instead
- You'd like to report your results as uncertainty (vs. the point estimate of Frequentist methods)
- Often used in, e.g., marketing and pharma
- Things standing in the way of wider adoption:
  - Still computationally intensive
  - Hard to verify the accuracy of the model

Topics we won’t cover but are important

- How to choose a prior
- How to choose a solution method (e.g. MCMC vs variational inference)
- Bayesian point estimates
- ... lots of other things! Keep exploring!

Resources to continue learning

- *Bayesian Methods for Hackers* by Cam Davidson
- PyMC3 documentation and tutorials
- Probabilistic programming primer (Adrian Sampson)
- Overview of No U-Turn Sampler
- Overview of prior predictive checks
- Overview of variational inference

https://github.com/larakattan/bayesian_inference_pymc3