COMS30035, Machine learning: Probabilistic Graphical Models 5

James Cussens
james.cussens@bristol.ac.uk

Department of Computer Science, SCEEM
University of Bristol

October 6, 2022

Agenda

- The Metropolis-Hastings algorithm
- MCMC in practice with PyMC

How to get MCMC to work?

- At the end of the last lecture we had a clear goal: **given** a target probability distribution $p(\mathbf{z})$, **construct** a Markov chain $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(i)} \ldots$ such that $\lim_{i \to \infty} p(\mathbf{z}^{(i)}) = p(\mathbf{z})$.
- (For Bayesian machine learning the target distribution will be $P(\theta | D = d)$, the posterior distribution of the model parameters given the observed data.)
- One solution to this is the *Metropolis-Hastings* algorithm.

The Metropolis-Hastings (MH) algorithm

- We define a single transition probability distribution for a homogeneous Markov chain.
- Let the current state be $\mathbf{z}^{(\tau)}$. When using the MH algorithm sampling the next state happens in two stages:
  1. We generate a value $\mathbf{z}^*$ by sampling from a *proposal distribution* $q(\mathbf{z}|\mathbf{z}^{(\tau)})$.
  2. We then accept $\mathbf{z}^*$ as the new state with a certain *acceptance probability*. If we don't accept $\mathbf{z}^*$ then we 'stay where we are', so that $\mathbf{z}^{(\tau)}$ is both the old and new state.

The Metropolis-Hastings acceptance probability

Let $p(\mathbf{z})$ be the *target distribution*. The acceptance probability is: [Bis06, p. 541].

$$A(\mathbf{z}^*, \mathbf{z}^{(\tau)}) = \min\left(1, \frac{p(\mathbf{z}^*)q(\mathbf{z}^{(\tau)}|\mathbf{z}^*)}{p(\mathbf{z}^{(\tau)})q(\mathbf{z}^*|\mathbf{z}^{(\tau)})}\right) \tag{1}$$

- If $p(\mathbf{z}) = \tilde{p}(\mathbf{z})/Z$ then we have $p(\mathbf{z}^*)/p(\mathbf{z}^{(\tau)}) = \tilde{p}(\mathbf{z}^*)/\tilde{p}(\mathbf{z}^{(\tau)})$, so we only need p up to normalisation. This is a big win!
- If the proposal distribution is symmetric then the 'q' terms cancel out: a special case known as the *Metropolis algorithm*.
- Note that for the Metropolis algorithm if $p(\mathbf{z}^*) \geq p(\mathbf{z}^{(\tau)})$ then we always accept and 'move' to $\mathbf{z}^*$.

Does Metropolis-Hastings (always) work?

.

- It can be shown [Bis06, p. 541] that the target distribution is an *invariant distribution of the Markov chain*: if the sequence of distributions $p(\mathbf{z}^{(i)})$ reaches the target distribution then it stays there.
- Also, typically the Markov chain does converge to the target distribution.
- The *rate* at which we converge to the target distribution is greatly influenced by the choice of proposal distribution.

MCMC in practice

- Straightforward Metropolis-Hastings is not the state-of-the-art in MCMC.
- *Probabilistic programming* systems like PyMC by default use more sophisticated MCMC algorithms (to avoid getting stuck).
- From the PyMC intro overview: “Probabilistic programming (PP) allows flexible specification of Bayesian statistical models in code. PyMC is a PP framework with an intuitive and readable, yet powerful, syntax that is close to the natural syntax statisticians use to describe models. It features next-generation Markov chain Monte Carlo (MCMC) sampling algorithms such as the No-U-Turn Sampler”
- When using MCMC we (1) throw away early samples (‘burn-in’) and (2) ‘run independent chains’ to check for convergence.
- PyMC uses $\hat{R}$ (`r_hat`) to check for convergence; this value should be close to 1.

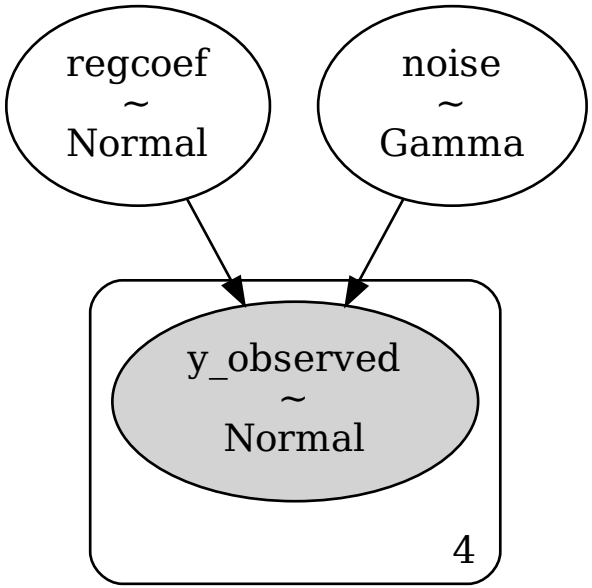
Linear regression with PyMC

```
import pymc as pm
import arviz as az
import numpy as np
import matplotlib.pyplot as plt

X, y = np.array([1,4,5,6]), np.array([1.4,4.9,4,4.6])
with pm.Model() as linear_model:
    weights = pm.Normal('weights', mu=0, sigma=1)
    noise = pm.Gamma('noise', alpha=2, beta=2)
    y_observed = pm.Normal('y_observed',
                           mu=X @ weights,
                           sigma=noise,
                           observed=y)

    posterior = pm.sample()
    print(az.summary(posterior))
    az.plot_trace(posterior)
    plt.show()
```

The model as a Bayesian network

- PyMC has a method `model_to_graphviz` which enables us to see the Bayesian network defined by the Python code.

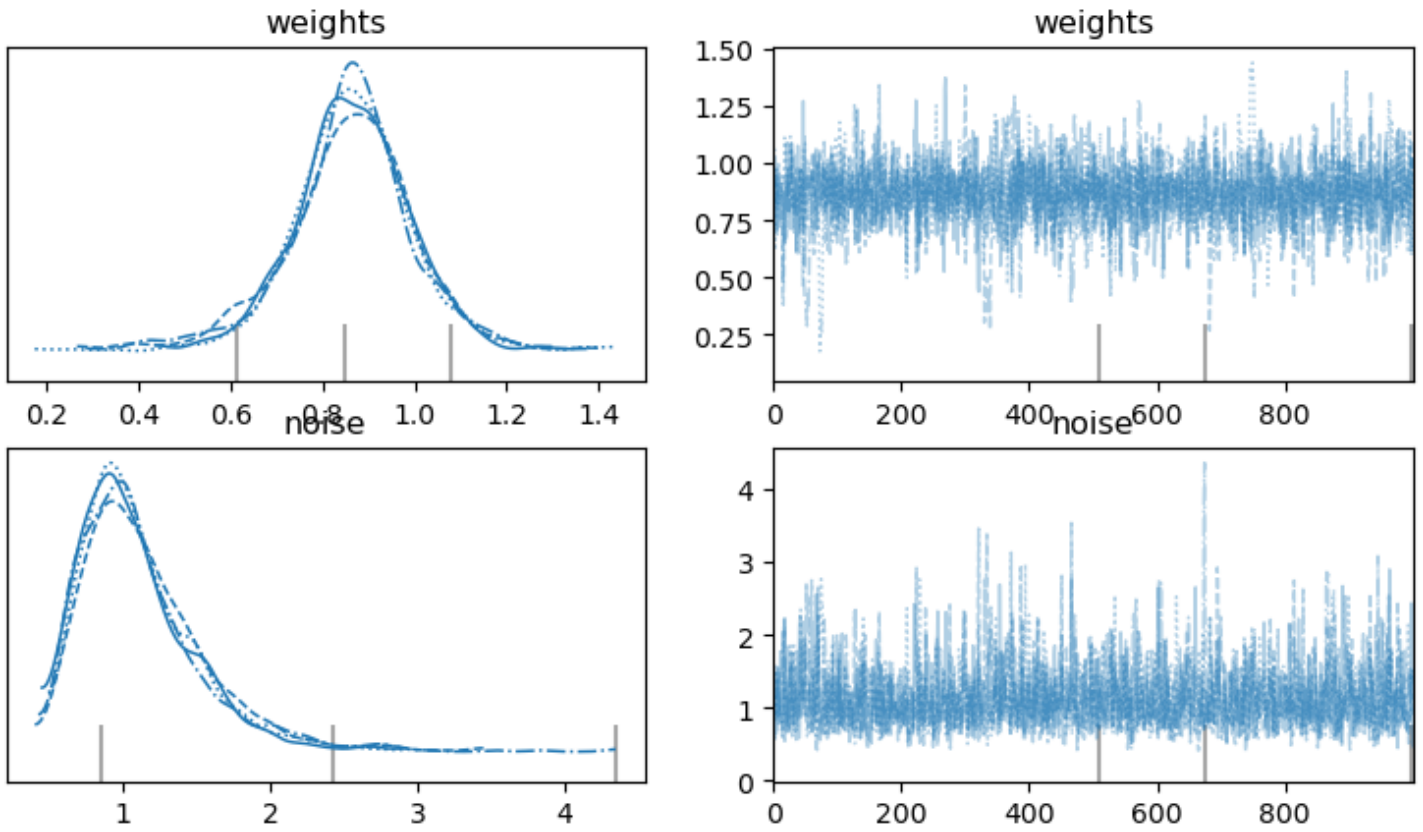
```
gv=pm.model_to_graphviz(linear_model)
gv.format = "pdf"
gv.render(filename="pgm5modelbn")
```



Textual summary of posterior distributions

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [weights, noise]
Sampling 4 chains for 1_000 tune and 1_000 draw iterations
 (4_000 + 4_000 draws total) took 6 seconds.
There were 2 divergences after tuning.
  Increase `target_accept` or reparameterize.
There was 1 divergence after tuning.
 Increase `target_accept` or reparameterize.
          mean     sd  hdi_3%  hdi_97%  ...  r_hat
weights  0.863  0.136   0.600    1.115  ...    1.0
noise    1.113  0.413   0.485    1.848  ...    1.0
```

Four estimates of the two posterior distributions



Now do the quiz!

Yes, please do the quiz for this lecture on Blackboard!


Christopher M. Bishop.
Pattern Recognition and Machine Learning.
Springer, 2006.