PSTAT 160B: Applied Stochastic Processes

Lecture 8. Markov chain Monte Carlo (projects).

Alex Shkolnik
shkolnik@ucsb.edu

April 28, 2025.

Department of Statistics & Applied Probability
University of California, Santa Barbara

Course Project. (Markov Chain Monte Carlo)

- *A technique to solve difficult combinatorial problems often formulated in terms of optimization.*
- *Broad application to problems involving high dimensional integration, Bayesian statistics, machine learning, etc.*
- *An example is the Metropolis-Hastings algorithm (one of the Top 10 greatest algorithms of the 20th century;* `https://www.jstor.org/stable/30037292`*)*
- *Comes out of the Manhattan Project to which Nicholas Metropolis was recruited by Oppenheimer in 1943.* *(*`www.wikipedia.org/wiki/Nicholas_Metropolis`*)*

Course project guidelines.

The default project is on MCMC / Metropolis-Hastings.

- *Groups of 2–4 (due on Monday following finals week).*
- *Submitted as written report (no page limits/requirements, structure guidelines to be posted on Canvas).*
- *Reach out to us via Discord if you would like to present your report for some extra credit.*

By special permission you can change the topic.

- *For those who would like to do something more advanced with the theory of stochastic processes.*
- *Reach out to us via Discord.*

Message encryption

Let Σ denote an alphabet.

- Letters and symbols used to write in a language.

e.g., English:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
. , ! ; # “ ” % & $

Σ (alphabet) is the set of all these characters.

A language is made up of words formed from Σ.

Let Σ denote an alphabet.

- Letters and symbols used to write in a language.

e.g., English:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d **e** f g h i j k l m **n** **o** **p** q **r** s **t** u v w x **y** z
. , ! ; # “ ” % & $

A simple alphabet:

$\Sigma = \{e, n, o, p, r, t, y\}$

What are some words in the language on Σ?

- entropy
- tree
- try
- no
- nope
- not
- yet
- rope
- pretty
- . . .

A simple alphabet:

$\Sigma = \{\text{e, n, o, p, r, t, y}\}$

Permutations can encrypt messages in the language on Σ.

$$\begin{pmatrix} \mathbf{e} & \mathbf{n} & \mathbf{o} & \mathbf{p} & \mathbf{r} & \mathbf{t} & \mathbf{y} \\ \text{n} & \text{y} & \text{t} & \text{r} & \text{o} & \text{p} & \text{e} \end{pmatrix}$$

The permutation here is $\sigma =$ "n y t r o p e".
c.f., cycle notation (`www.wikipedia.org/wiki/Permutation`).

Message. $T =$ no rope try tree

$\Rightarrow \sigma_T =$ yt otrn poe ponn
(click send)

Message decryption

A simple alphabet:

$\Sigma = \{e, n, o, p, r, t, y\}$

Encryption key:

$$\begin{pmatrix} \mathbf{e} & \mathbf{n} & \mathbf{o} & \mathbf{p} & \mathbf{r} & \mathbf{t} & \mathbf{y} \\ \mathrm{n} & \mathrm{y} & \mathrm{t} & \mathrm{r} & \mathrm{o} & \mathrm{p} & \mathrm{e} \end{pmatrix}$$

The permutation here is $\sigma =$ “n y t r o p e”.

Message. $T =$ no rope try tree

$\Rightarrow \sigma_T =$ yt otrn poe ponn
(click send)

Decryption key:

$$\begin{pmatrix} \mathbf{e} & \mathbf{n} & \mathbf{o} & \mathbf{p} & \mathbf{r} & \mathbf{t} & \mathbf{y} \\ ? & ? & ? & ? & ? & ? & ? \end{pmatrix}$$

A simple alphabet: $\Sigma = \{\text{e, n, o, p, r, t, y}\}$

You receive the following email message.

$$T = \text{yt otrn poe ponn}$$

But you happen to have the **decryption key**.

$$\begin{pmatrix} \mathbf{e} & \mathbf{n} & \mathbf{o} & \mathbf{p} & \mathbf{r} & \mathbf{t} & \mathbf{y} \\ \text{y} & \text{e} & \text{r} & \text{t} & \text{p} & \text{o} & \text{n} \end{pmatrix}$$

The permutation $\sigma =$ "y e r t p o n" can decrypt T.

$\Rightarrow \sigma_T =$ no rope try tree

Likelihood/Entropy

Abstractly, $T = (t_0, t_1, \ldots, t_n)$ is a message of length $n + 1$.

- t_k *is the* k*th character of the message,*
- *all characters belong to some alphabet* Σ.
- *permutation* σ *maps character* t_k *to* σ_{t_k},
- *permutation* σ *maps message* T *to* σ_T.

A simple alphabet: $\Sigma = \{\text{e, n, o, p, r, t, y}\}$

$$T = \text{yt otrn poe ponn}$$

The "right" permutation $\sigma =$"y e r t p o n" can decrypt T.

$\Rightarrow \sigma_T =$ no rope try tree

The "wrong" permutation $\sigma =$"p y t e r n o" will not.

$\Rightarrow \sigma_T =$ on tnry etp etyy

How does a computer decide which one is more "right"?

Let $\ell(a, b)$ denote the prior probability of seeing the letter b given that the previous letter was a.

- *These may be estimated from some large corpus of text.*
- *e.g., Wuthering Heights, War and Peace, etc.*

This results in a $|\Sigma| \times |\Sigma|$ matrix of probabilities.

This is the "training" part of machine learning.

A simple alphabet: $\Sigma = \{\mathrm{e, n, o, p, r, t, y}\}$

The matrix of probabilities is 7×7.

Which is larger, $\ell(\mathrm{n, o})$ or $\ell(\mathrm{n, t})$?

Given any message T and a (possibly wrong) decoding permutation σ, the likelihood of σ_T being in our language is,

$$\mu_T(\sigma) = \prod_{k=1}^{n} \ell(\sigma_{t_{k-1}}, \sigma_{t_k}).$$

Given any message T and a (possibly wrong) decoding permutation σ, the likelihood of σ_T being in our language is,

$$\mu_T(\sigma) = \prod_{k=1}^{n} \ell(\sigma_{t_{k-1}}, \sigma_{t_k}).$$

Goal. Find the permutation σ on Σ for which $\mu(\sigma_T)$ is largest.

For the simple alphabet $\Sigma = \{\text{e, n, o, p, r, t, y}\}$,

- *there are* $7! = 5040$ *permutations* σ*;*
- *and we can check each value* $\mu_T(\sigma)$ *(brute force) to decode the message* $T = \text{yt otrn poe ponn}$.

26 letters in English give a space of permutations $\mathbb{S}$ of size,

$$\approx 4 \times 10^{26},$$

and that's not counting capital letters and punctuation.

The guiding principle

Given any message T and a (possibly wrong) decoding permutation σ, the likelihood of σ_T being in our language is,

$$\mu_T(\sigma) = \prod_{k=1}^{n} \ell(\sigma_{t_{k-1}}, \sigma_{t_k}).$$

Goal. Find the permutation σ on Σ for which $\mu_T(\sigma)$ is largest.

Let $\mathbb{S}$ be the space of all permutations on Σ.

For each $\sigma \in \mathbb{S}$ the value $\mu_T(\sigma)$ is nonnegative. So,

$$\pi(\sigma) = \frac{\mu_T(\sigma)}{\sum_{z \in \mathbb{S}} \mu_T(z)} \qquad \forall \sigma \in \mathbb{S}$$

forms a distribution (nonnegative numbers on $\mathbb{S}$ that sum to 1).

Construct a Markov chain with stationary distribution π.
(equivalently, with stationary measure μ_T)

Important. We cannot compute π directly because the sum $\sum_{y \in \mathbb{S}} \mu(y)$ cannot be evaluated (since $|\mathbb{S}|$ is too large).

Given any message T and a (possibly wrong) decoding permutation σ, the likelihood of σ_T being in our language is,

$$\mu_T(\sigma) = \prod_{k=1}^{n} \ell(\sigma_{t_{k-1}}, \sigma_{t_k}).$$

Construct a Markov chain with stationary measure μ_T.

- *Statisticians think of* μ_T *as the likelihood.*
- *A physicist would call* μ_T *the entropy.*

As the chain runs, its entropy increases.

- *it gets closer closer to the stationary distribution,*
- *i.e., closer and closer to the more likely states,*
- *i.e., closer to the permutations that decrypt* T*.*

The algorithm

Our state space is $\mathbb{S}$ (all permutations on Σ).

- *Lets walk randomly on this state space.*
- *At each state $\sigma \in \mathbb{S}$ we apply σ to the message T.*
- $\mu_T(\sigma)$ *tells us how likely it is that σ decrypts T.*
- *Eventually, we will find $\sigma \in \mathbb{S}$ that decrypts T.*

What are the problems with this?

(1) Too many transitions from each state.
(2) Its worse than even brute force search.

To fix issue (1) we can remove most transitions.

- *Allow transition $\sigma \to \sigma'$ only if the two differ by a switch of two characters (e.g., "y e r t p o n" and "e y r t p o n")*

To fix issue (2) we will bias our transition probabilities toward the more "likely" states by using the ratio, $\mu_T(\sigma')/\mu_T(\sigma)$.

Our state space is $\mathbb{S}$ (all permutations on Σ).

- *Consider the transition* $\sigma \to \sigma'$ *with probability*

$$q(\sigma, \sigma') = \binom{|\Sigma|}{2}^{-1}$$

if $\sigma, \sigma' \in \mathbb{S}$ *differ by a switch of two characters.*

- *If* $\mu_T(\sigma') > \mu_T(\sigma)$ *then move to* σ'.
- *Else, flip a coin with Heads probability*

$$\mu_T(\sigma')/\mu_T(\sigma).$$

- *Move to* σ' *on Heads and stay at* σ *otherwise.*
- *Repeat for a while (keeping track of* σ_T*).*

What we just described is a Markov chain X.

The state space $\mathbb{S}$ is all permutations on Σ.

We can evaluate $\mu_T(\sigma)$ at any $\sigma \in \mathbb{S}$, the likelihood of σ_T.

The chain X has the transition probability, zero or

$$\mathrm{p}(\sigma, \sigma') = \binom{|\Sigma|}{2}^{-1} \min\left(1, \frac{\mu_T(\sigma')}{\mu_T(\sigma)}\right)$$

for $\sigma, \sigma' \in \mathbb{S}$ that differ by a switch of two characters.

Show that X is reversible with stationary distribution

$$\pi(\sigma) = \frac{\mu_T(\sigma)}{\sum_{y \in \mathbb{S}} \mu_T(y)} \qquad \forall \sigma \in \mathbb{S}.$$

Important. We cannot compute π directly because the sum $\sum_{y \in \mathbb{S}} \mu(y)$ cannot be evaluated (since $|\mathbb{S}|$ is too large).

Metropolis-Hastings

Problem. Given a huge $\mathbb{S}$ with a function $\mu : \mathbb{S} \to [0, \infty)$ we want to find the largest $\mu(x)$ for $x \in \mathbb{S}$.

- *This x is the most likely/probable state (highest PageRank).*

MCMC constructs a Markov chain X with stationary measure μ and lets it run untill it reaches the most likely states.

This also constructs π by implicitly computing $\sum_{y \in \mathbb{S}} \mu(y)$.

Metropolis-Hastings is a specific MCMC algorithm.

Suppose $\mathbb{S}$ with a function $\mu : \mathbb{S} \to [0, \infty)$

Let $\mathrm{q}(x, y)$ be a set of transition probabilities for $x, y \in \mathbb{S}$.

Constuct X with transition probability for $x \to y$ given by,

$$\mathrm{p}(x, y) = \mathrm{q}(x, y)a(x, y)$$

with acceptance probability of the form

$$a(x, y) = \min\left(1, \frac{\mu(y)\mathrm{q}(y, x)}{\mu(x)\mathrm{q}(x, y)}\right)$$

- *Self-transitions ensure each* $\sum_y \mathrm{p}(x, y) = 1$
- *What was our choice for* $\mathrm{q}(x, y)$ *above?*

Try to show that X is reversible with stationary distribution

$$\pi(x) = \frac{\mu(x)}{\sum_{z \in \mathbb{S}} \mu(z)} \qquad \forall x \in \mathbb{S}.$$

We will post code to play with on Canvas/Discord.

Start reading,

- *www.wikipedia.org/wiki/Metropolis-Hastings_algorithm*
- *Section 4.9 of Ross (2019).*
- *Section 2.1 of Conner (2003).*

and comparing with the above.

In the course projects, your group should try to focus on either

- *theory*
- *applications,*
- *or coding.*

Further, directions upcoming on Canvas/Discord.

Markov Chain Monte Carlo

One (but not the only) way to think about MCMC is in terms of solving very challenging combinatorial problems.

The number of combinations is too large.

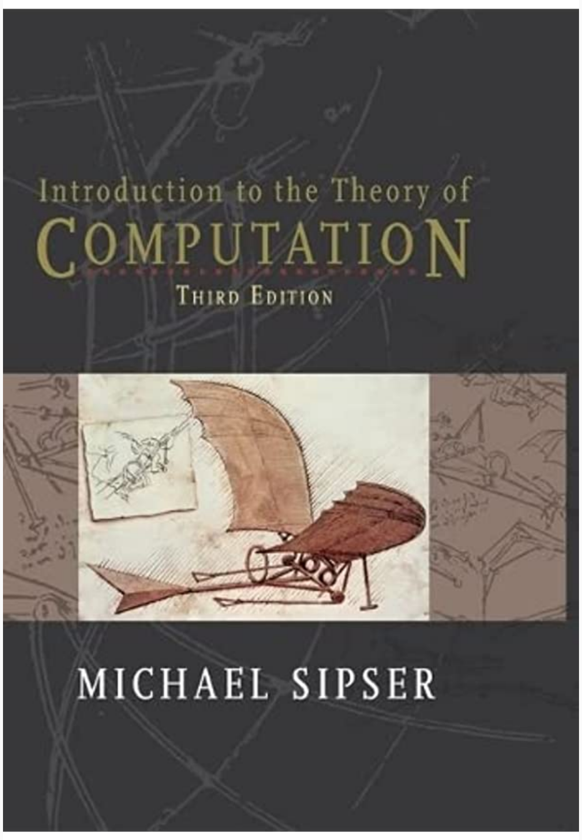
- *N stocks with (just) two possible future prices leads to a total of 2^N possible economic outcomes (e.g., $2^{40} \sim 1\text{e}12$).*
- *The total number of ways to arrange N (labeled) particles in some particular order is $N!$ (e.g., $40! \sim 1\text{e}48$).*

Let $\mathbb{S}$ be a set of many things and $\mu : \mathbb{S} \to [0, \infty)$.

- $\mu(y)$ *is how happy I am with economic outcome y.*
- $\mu(x)$ *is the energy of the particles with order x.*

MCMC solves (approximately) the problem of finding the "states" in $\mathbb{S}$ for which μ is maximized.

- *Helpful when checking each $\mu(x)$ is infeasible ($\mathbb{S}$ too big).*
- *Works remarkably well even on problems that are theoretically shown to be intractable. (Huh?)*


Introduction to the Theory of
COMPUTATION
THIRD EDITION
MICHAEL SIPSER

How does MCMC find the maximum of a function μ on $\mathbb{S}$?

It constructs a reversible Markov chain X on state space $\mathbb{S}$ with

$$\pi(x) = \frac{\mu(x)}{\sum_{z \in \mathbb{S}} \mu(z)} \qquad \forall x \in \mathbb{S},$$

the stationary distribution.

If you simulate X for along time you are most likely to see it in states x with largest π (since π is the stationary distribution).

- *After a while, X finds $x \in \mathbb{S}$ where $\mu(x)$ tends to be large.*

How to construct Markov chains that reach their stationary distributions quickly is the "art/science" of MCMC.

References

Conner, S. (2003), 'Simulation and solving substitution codes', *Master's thesis, Department of Statistics, University of Warwick* .

Ross, S. M. (2019), *Introduction to Probability Models*, Elsevier. 12 ed.