Lecture 9: Latent Dirichlet Allocation for Topic Modelling

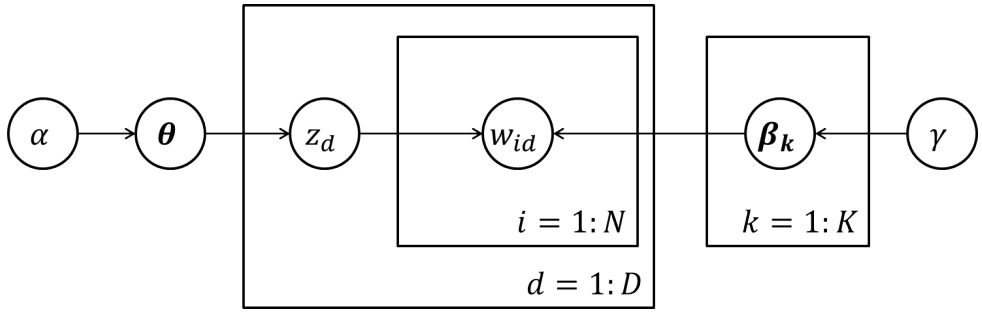
4F13: Machine Learning

Joaquin Quiñonero-Candela and Carl Edward Rasmussen

Department of Engineering
University of Cambridge

http://mlg.eng.cam.ac.uk/teaching/4f13/

Limitations of the mixture of Multinomials model



A generative view of the mixture of Multinomials model

1. Draw a Multinomial θ over topics from the α Dirichlet.
2. Draw K topic Multinomials β_k over words from the γ Dirichlet.
3. Draw a topic z_d for document d from the θ Multinomial.
4. Draw N_d words W_{id} for this document from the β_{z_d} Multinomial.

Limitations:

- All words in each document are drawn from one specific topic Multinomial.
- This works if each document is exlusively about one topics, but if some documents span more than one topic, then "blurred" topics must be learnt.

NIPS dataset: LDA topics 1 to 7 out of 20.

network	network	model	problem	neuron	network	cell
unit	node	data	constraint	cell	neural	model
training	representation	distribution	distance	input	system	visual
weight	input	probability	cluster	model	model	direction
input	unit	parameter	point	synaptic	control	motion
hidden	learning	set	algorithm	firing	output	field
output	activation	gaussian	tangent	response	recurrent	eye
learning	nodes	error	energy	activity	input	unit
layer	pattern	method	clustering	potential	signal	cortex
error	level	likelihood	optimization	current	controller	orientation
set	string	prediction	cost	synapses	forward	map
neural	structure	function	graph	membrane	error	receptive
net	grammar	mean	method	pattern	dynamic	neuron
number	symbol	density	neural	output	problem	input
performance	recurrent	prior	transformation	inhibitory	training	head
pattern	system	estimate	matching	effect	nonlinear	spatial
problem	connectionist	estimation	code	system	prediction	velocity
trained	sequence	neural	objective	neural	adaptive	stimulus
generalization	order	expert	entropy	function	memory	activity
result	context	bayesian	set	network	algorithm	cortical

NIPS dataset: LDA topics 8 to 14 out of 20.

circuit	learning	speech	classifier	network	data	function
chip	algorithm	word	classification	neuron	memory	linear
network	error	recognition	pattern	dynamic	performance	vector
neural	gradient	system	training	system	genetic	input
analog	weight	training	character	neural	system	space
output	function	network	set	pattern	set	matrix
neuron	convergence	hmm	vector	phase	features	component
current	vector	speaker	class	point	model	dimensional
input	rate	context	algorithm	equation	problem	point
system	parameter	model	recognition	model	task	data
vlsi	optimal	set	data	function	patient	basis
weight	problem	mlp	performance	field	human	output
implementation	method	neural	error	attractor	target	set
voltage	order	acoustic	number	connection	similarity	approximation
processor	descent	phoneme	digit	parameter	algorithm	order
bit	equation	output	feature	oscillation	number	method
hardware	term	input	network	fixed	population	gaussian
data	result	letter	neural	oscillator	probability	network
digital	noise	performance	nearest	states	item	algorithm
transistor	solution	segment	problem	activity	result	dimension

NIPS dataset: LDA topics 15 to 20 out of 20.

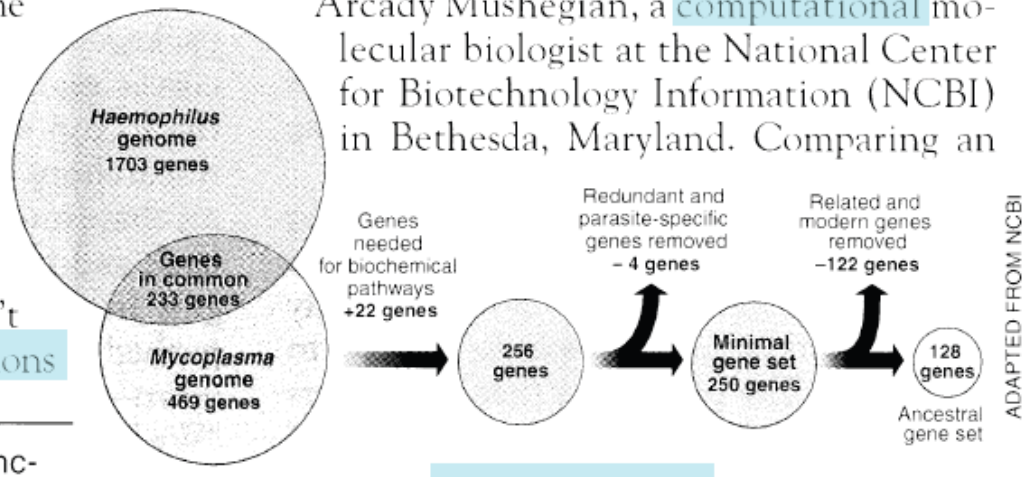
function	learning	model	image	rules	signal
network	action	object	images	algorithm	frequency
bound	task	movement	system	learning	noise
neural	function	motor	features	tree	spike
threshold	reinforcement	point	feature	rule	information
theorem	algorithm	view	recognition	examples	filter
result	control	position	pixel	set	channel
number	system	field	network	neural	auditory
size	path	arm	object	prediction	temporal
weight	robot	trajectory	visual	concept	model
probability	policy	learning	map	knowledge	sound
set	problem	control	neural	trees	rate
proof	step	dynamic	vision	information	train
net	environment	hand	layer	query	system
input	optimal	joint	level	label	processing
class	goal	surface	information	structure	analysis
dimension	method	subject	set	model	peak
case	states	data	segmentation	method	response
complexity	space	human	task	data	correlation
distribution	sutton	inverse	location	system	neuron

Latent Dirichlet Allocation (LDA)

Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK—How many genes does an organism need to survive? Last week at the genome meeting here,* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those predictions "are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a genetic numbers game, particularly as more and more genomes are completely mapped and sequenced. "It may be a way of organizing any newly sequenced genome," explains Arcady Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an

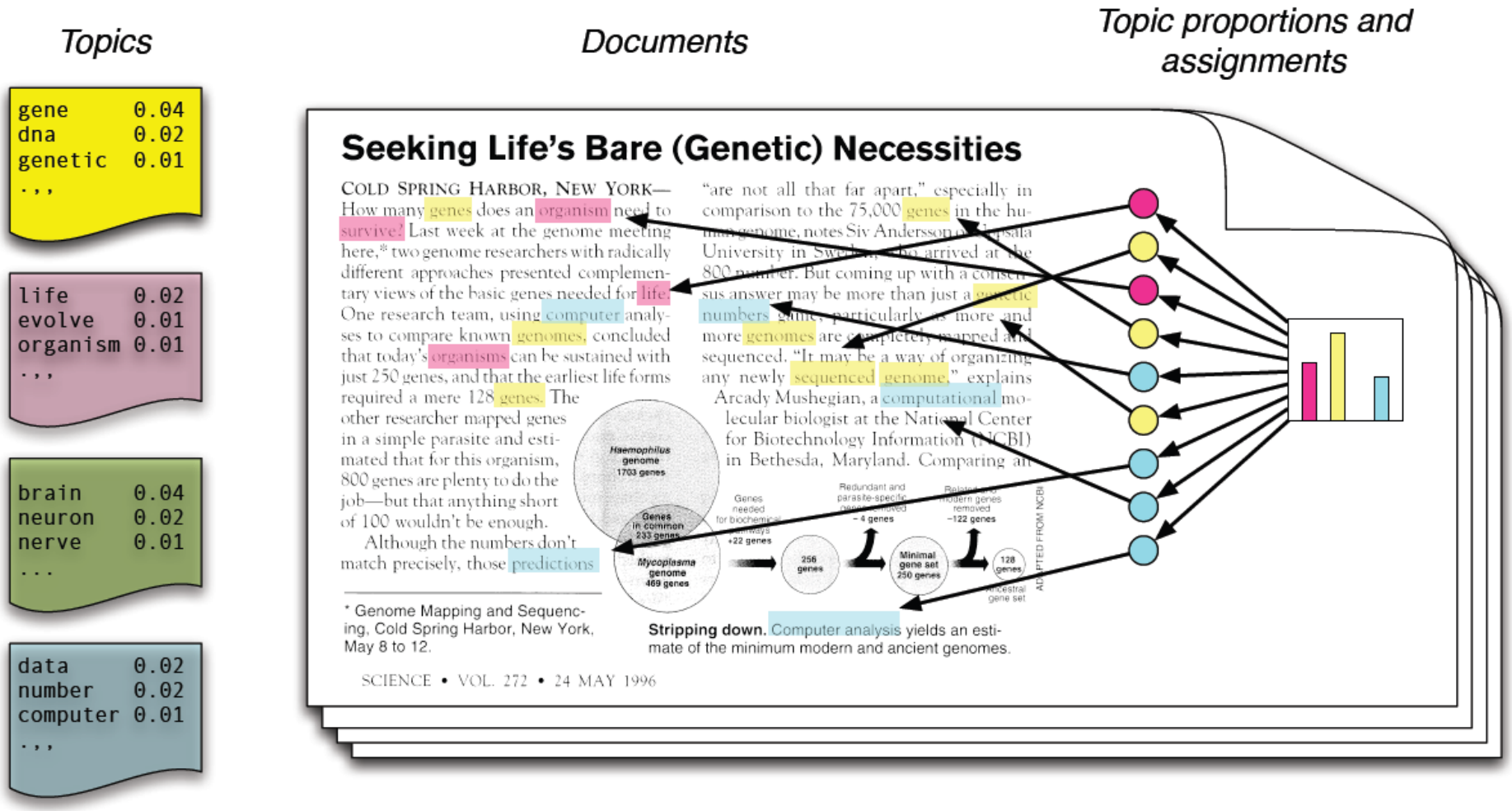


Stripping down. Computer analysis yields an estimate of the minimum modern and ancient genomes.

* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

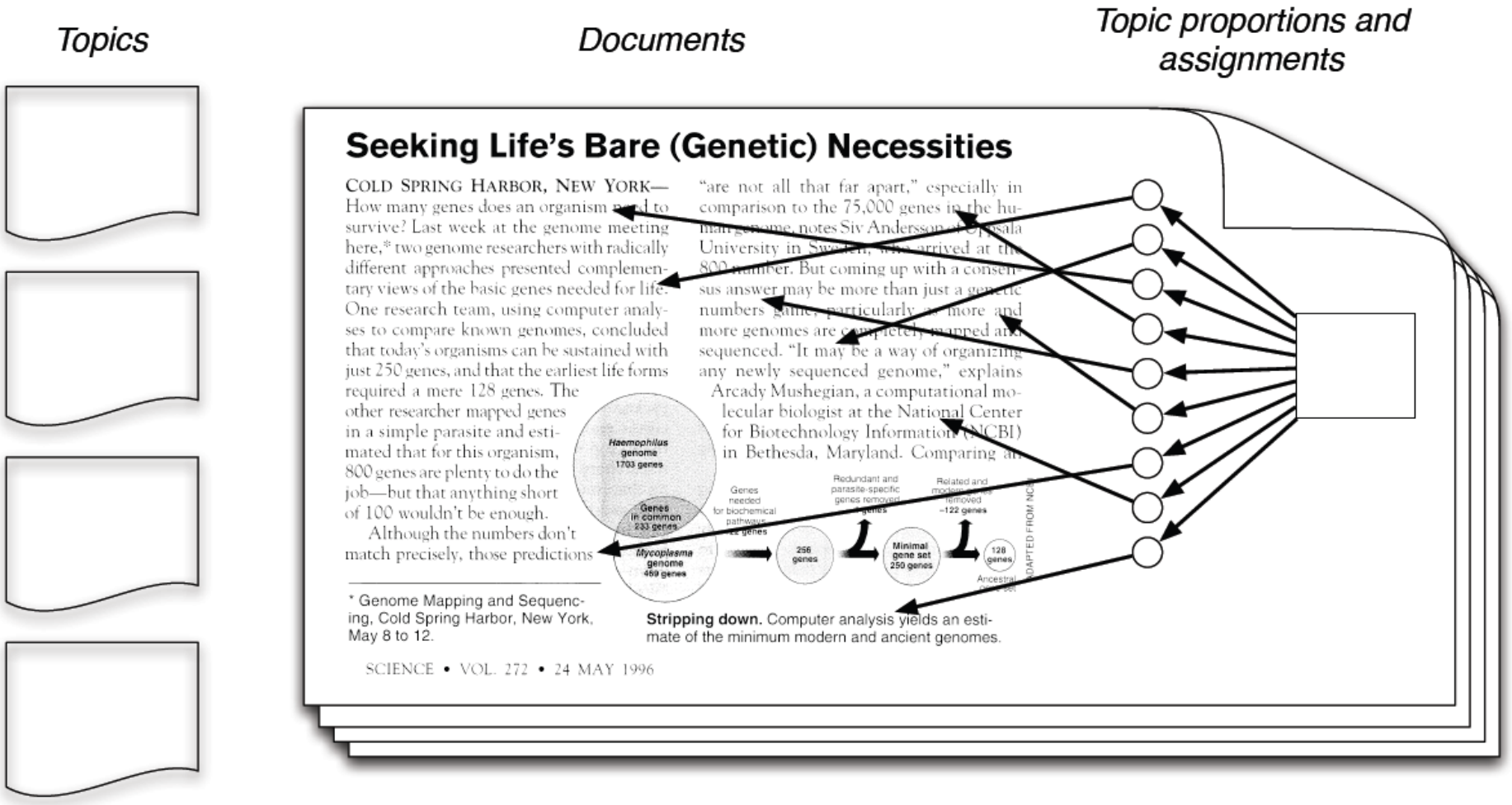
SCIENCE • VOL. 272 • 24 MAY 1996

Generative model for LDA



- Each *topic* is a distribution over words.
- Each *document* is a mixture of corpus-wide topics.
- Each *word* is drawn from one of those topics.

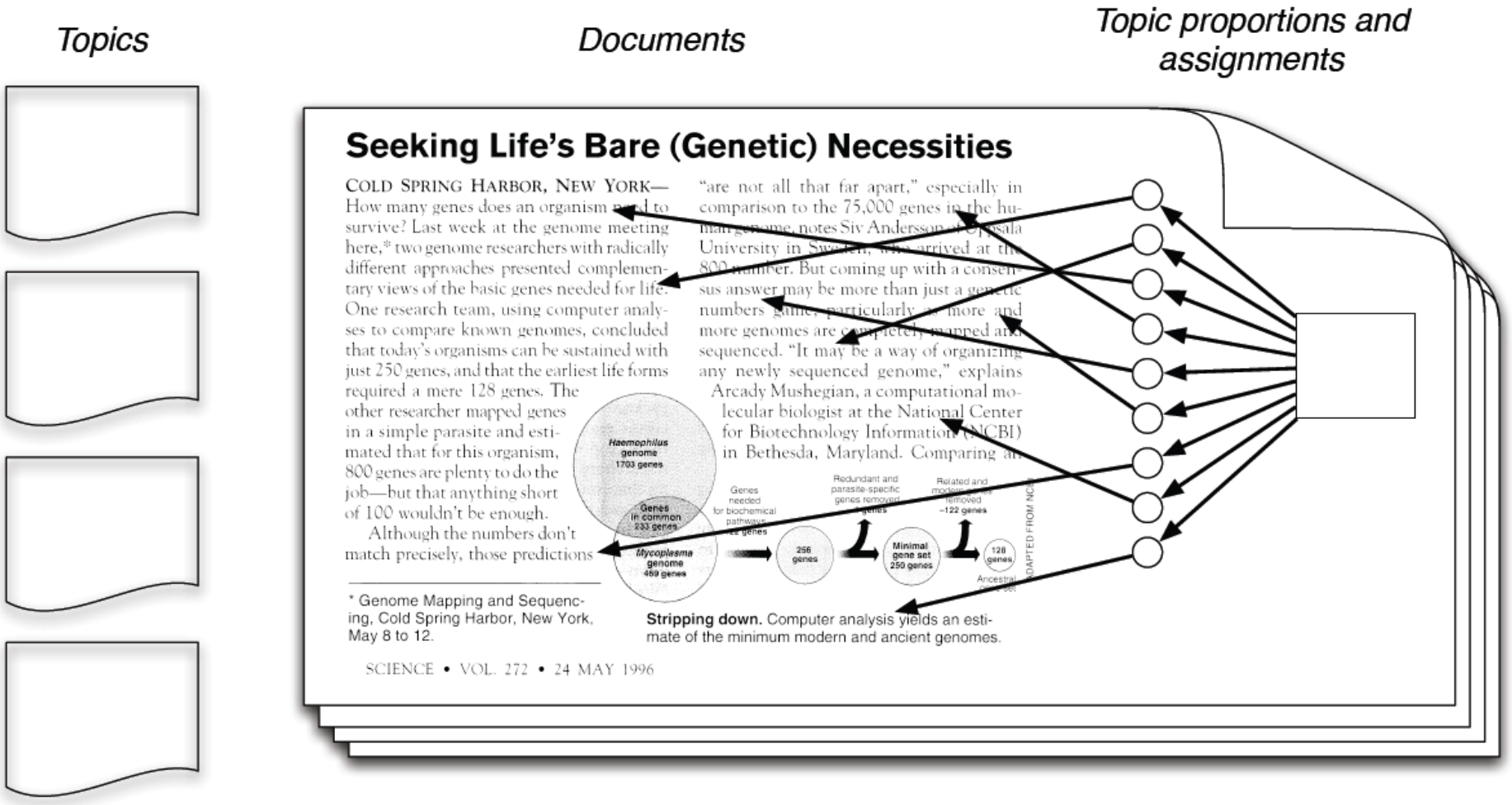
The posterior distribution



- In reality, we only observe the documents.
- The other structure are *hidden* variables.

from David Blei

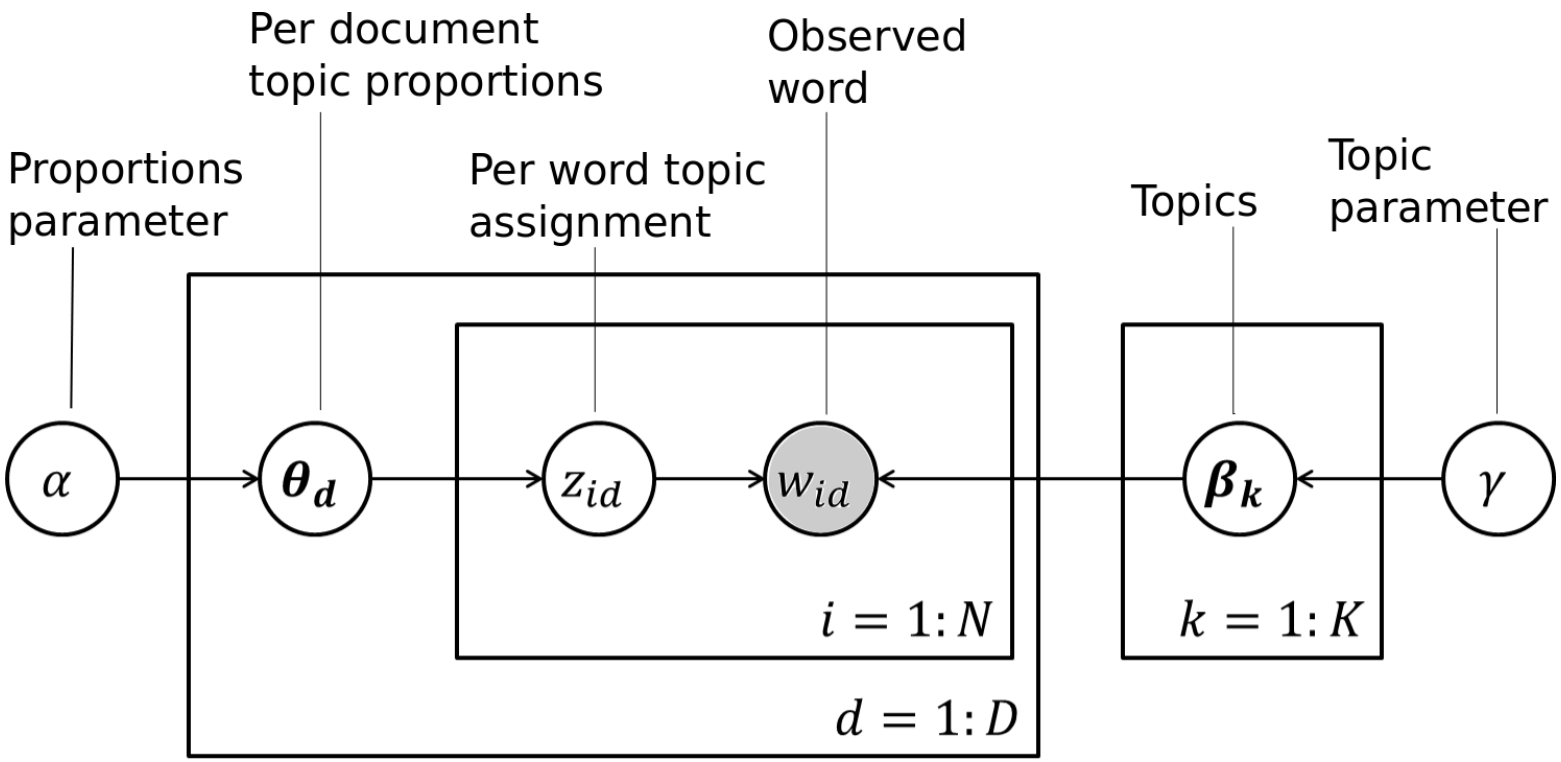
The posterior distribution



- Our goal is to *infer* the hidden variables.
- This means computing their distribution conditioned on the documents

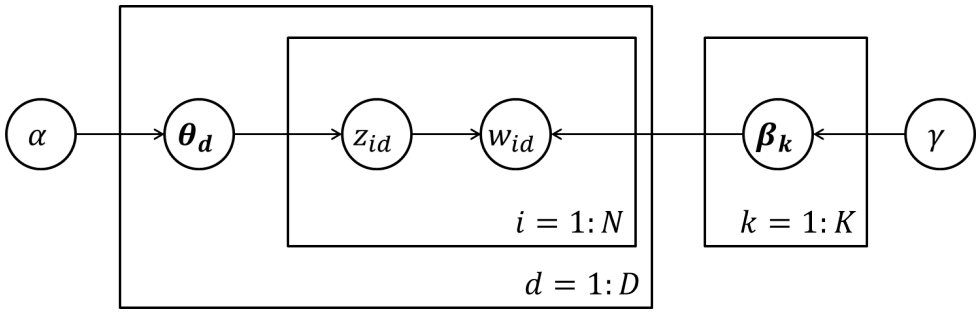
$$p(\text{topics, proportions, assignments}|\text{documents})$$

The LDA graphical model



- Nodes are random variables; edges indicate dependence.
- Shaded nodes indicate *observed* variables.

The difference between LDA and mixture of Multinomials



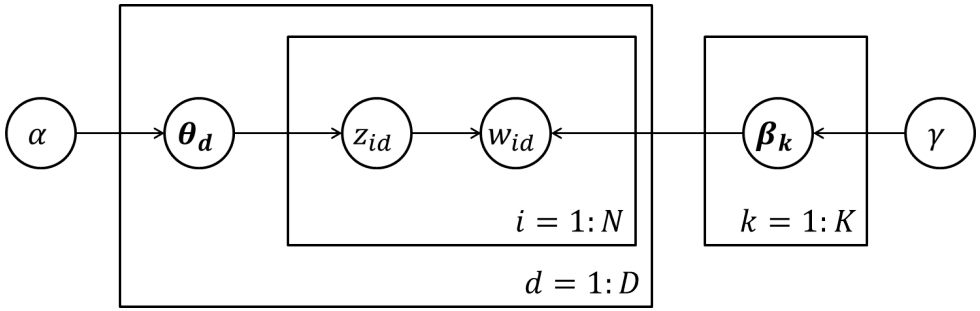
A generative view of LDA

1. For each document draw a Multinomial θ_d over topics from the α Dirichlet.
2. Draw K topic Multinomials β_k over words from the γ Dirichlet.
3. Draw a topic z_{id} for the i-th word in document d from the θ Multinomial.
4. Draw word w_{id} from the β_{z_d} Multinomial.

Differences with the mixture of Multinomials model:

- Every word in a document can be drawn from a different topic.
- Every document has its own topic assigment Multinomial θ_d.

The impossible LDA math



"Always write down the probability of everything." (Steve Gull)

$$p(\beta_{1:K}, \theta_{1:D}, \{z_{id}\}, \{w_{id}\}|\gamma, \alpha)$$
$$= \prod_{k=1}^{K} p(\beta_k|\gamma) \prod_{d=1}^{D} p(\theta_d|\alpha) \Big(\prod_{i=1}^{N_d} p(z_{id}|\theta_d) p(w_{id}|\beta_{1:K}, z_{id}) \Big)$$

For example, the posterior over the parameters, $\beta_{1:K}$ and $\theta_{1:D}$ requires the we marginalize out the latent $\{z_{id}\}$. But how many configurations are there?

This computation is *intractable*.

Monte Carlo and Markov Chain Monte Carlo

Instead of attempting to evaluate all possible configurations of the latent variables, in Monte Carlo we use *random samples*, drawn from the distribution in question:

$$\int f(x)p(x)dx \simeq \frac{1}{T}\sum_{t=1}^{T} f(x^{(t)}),$$

where $x^{(t)}$ are samples drawn from $p(x)$.

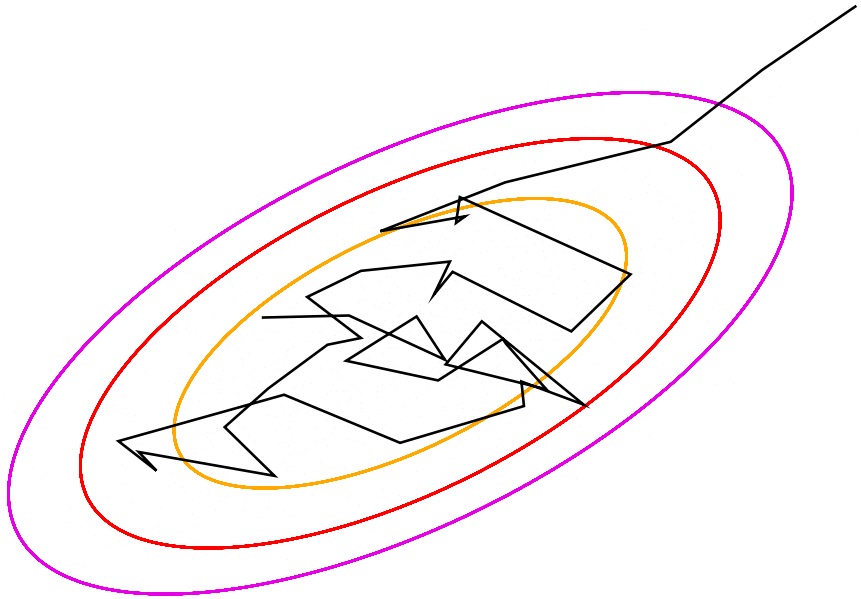
This is a powerful technique, which may work well, even if x is very high dimensional.

Usually, it is difficult to draw samples *independently* from $p(x)$. In Markov Chain Monte Carlo, one designs a Markov Chain to generate (dependent) samples from the target distribution $p(x)$.

Markov Chain Monte Carlo

We want to construct a Markov Chain that explores $p(\mathbf{x})$.

Markov Chain: $\mathbf{x}^{(t)} \sim q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)})$.



MCMC gives approximate, correlated samples from $p(x)$.

Challenge: how do we find transition probabilities $q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)})$, which give rise to the correct stationary distribution $p(\mathbf{x})$?

Discrete Markov Chains

Consider

$$\mathbf{p} = \begin{bmatrix} 3/5 \\ 1/5 \\ 1/5 \end{bmatrix}, \qquad Q = \begin{bmatrix} 2/3 & 1/2 & 1/2 \\ 1/6 & 0 & 1/2 \\ 1/6 & 1/2 & 0 \end{bmatrix}, \qquad Q_{ij} = Q(x_i \leftarrow x_j)$$

where Q is a stochastic (or transition) matrix.

To machine precision: $Q^{100} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \mathbf{p}$.

$\mathbf{p}$ is called a stationary distribution of Q, since $Q\mathbf{p} = \mathbf{p}$.

Ergodicity is also a requirement.

Continuous Spaces and Detailed Balance

In continuous spaces transitions are governed by $q(\mathbf{x}'|\mathbf{x})$.

Now, $p(\mathbf{x})$ is a stationary distribution for $q(\mathbf{x}'|\mathbf{x})$ if

$$\int q(\mathbf{x}'|\mathbf{x})p(\mathbf{x})d\mathbf{x} = p(\mathbf{x}').$$

Detailed balance means

$$q(\mathbf{x}'|\mathbf{x})p(\mathbf{x}) = q(\mathbf{x}|\mathbf{x}')p(\mathbf{x}').$$

Now, integrating both sides wrt $\mathbf{x}$, we get

$$\int q(\mathbf{x}'|\mathbf{x})p(\mathbf{x})d\mathbf{x} = \int q(\mathbf{x}|\mathbf{x}')p(\mathbf{x}')d\mathbf{x} = p(\mathbf{x}').$$

Thus, detailed balance implies the existence of a stationary distribution

The Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm:

- propose a new state $\mathbf{x}^*$ from $q(\mathbf{x}^*|\mathbf{x}^{(\tau)})$
- compute the acceptance probability a

$$a = \frac{p(\mathbf{x}^*)}{p(\mathbf{x}^{(\tau)})} \frac{q(\mathbf{x}^{(\tau)}|\mathbf{x}^*)}{q(\mathbf{x}^*|\mathbf{x}^{(\tau)})}$$

- if $a > 1$ then the proposed state is accepted,
  otherwise the proposed state is accepted with probability a.
  If the proposed state is accepted, then $\mathbf{x}^{(\tau+1)} = \mathbf{x}^*$ otherwise $\mathbf{x}^{(\tau+1)} = \mathbf{x}^{(\tau)}$.

This Markov chain has $p(\mathbf{x})$ as a stationary distribution. This holds trivially if $\mathbf{x}^{(\tau+1)} = \mathbf{x}^{(\tau)}$, otherwise

$$\begin{aligned} p(\mathbf{x})Q(\mathbf{x}' \leftarrow \mathbf{x}) &= p(\mathbf{x})q(\mathbf{x}'|\mathbf{x}) \min\left(1, \frac{p(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{p(\mathbf{x})q(\mathbf{x}'|\mathbf{x})}\right) \\ &= \min\left(p(\mathbf{x})q(\mathbf{x}'|\mathbf{x}), p(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')\right) \\ &= p(\mathbf{x}')q(\mathbf{x}|\mathbf{x}') \min\left(1, \frac{p(\mathbf{x})q(\mathbf{x}'|\mathbf{x})}{p(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}\right) = p(\mathbf{x}')Q(\mathbf{x} \leftarrow \mathbf{x}'). \end{aligned}$$

Some properties of Metropolis Hastings

- The Metropolis algorithm has $p(\mathbf{x})$ as its stationary distribution
- If $q(\mathbf{x}^*|\mathbf{x}^{(\tau)})$ is symmetric, then
  - the expression for a simplifies to $a = p(\mathbf{x}^*)/p(\mathbf{x}^{(\tau)})$
  - the algorithm then always accepts if the proposed state has higher probability than the current state and sometimes accepts a state with lower probability.
- we only need the ratio of $p(\mathbf{x})$'s, so we don't need the normalization constant. This is important, e.g. when sampling from a posterior distribution.

The Metropolis algorithm can be widely applied, you just need to specify a proposal distribution.

The proposal distribution must satisfy some (mild) constraints (related to ergodicity).

The Proposal Distribution

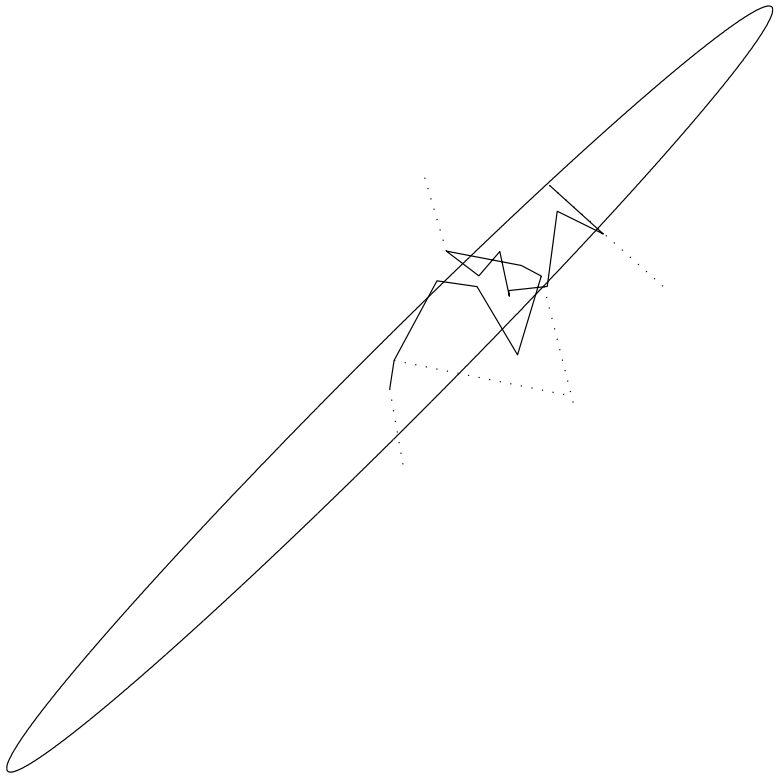
Often, Gaussian proposal distributions are used, centered on the current state. You need to specify the width of the proposal distribution.

What happens if the proposal distribution is

- too wide?
- too narrow?

Metropolis Hastings Example

20 iterations of the Metropolis Hastings algorithm for a bivariate Gaussian



The proposal distribution was Gaussian centered on the current state.

Rejected states are indicated by dotted lines.