Fast sampling and LDA

Announcements – 1/3

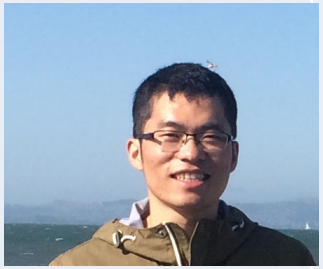
- 12/7 is in-class final exam
- 12/5 is presentation for 10-805 students
  - 10 projects at 5min/project
  - only self-directed 805 projects: reproducibility projects will **not** be presented
- Final review session will be abbreviated
  - maybe just questions
- It's fine if not everything is finished on 12/5
  - I know the writeup is due 132 hours later

Announcements – 2/3

- Do you need AWS $$ for your project?
  - Email William and cc Rose

Announcements – 3/3

I need good TAs for 10-405 this spring and 10-605 next fall!

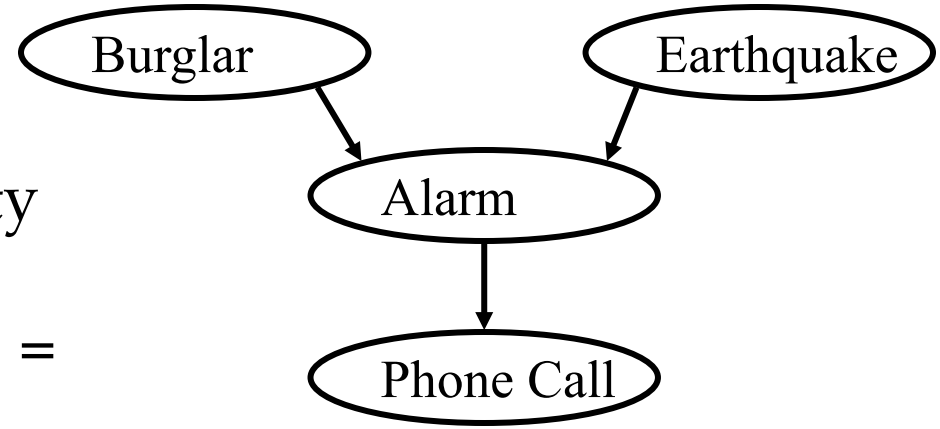
				
Anant	Bo Chen	Chen Hu	Minxing	Ning
				
Janini	Rose	Tao	Yifan	Yuhan

Directed Graphical Models

DGMs: The "Burglar Alarm" example

Node ~ random variable

Arcs define **form** of probability distribution:
Pr(Xi | X1,…,Xi-1,Xi+1,…,Xn) =
Pr(Xi | parents(Xi))

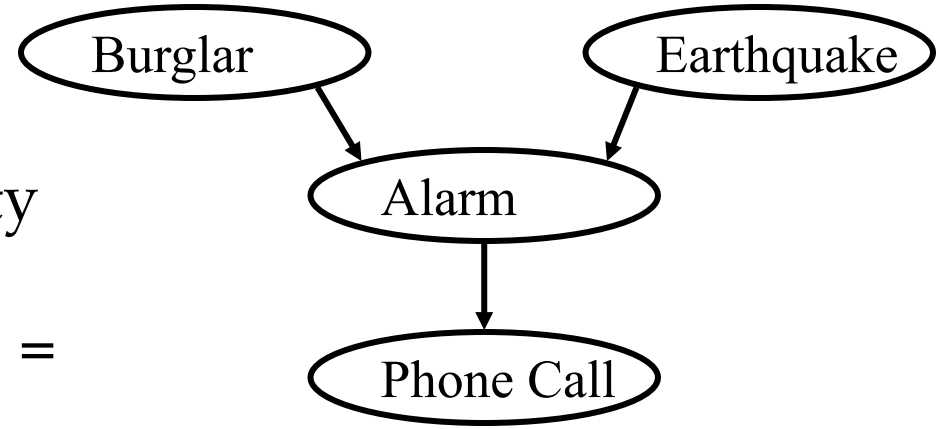


- Your house has a twitchy burglar alarm that is also sometimes triggered by earthquakes.
- While you are on vacation, one of your neighbors calls and tells you your home's burglar alarm is ringing. Uh oh!

DGMs: The "Burglar Alarm" example

Node ~ random variable

Arcs define **form** of probability distribution:
Pr(Xi | X1,…,Xi-1,Xi+1,…,Xn) =
Pr(Xi | parents(Xi))



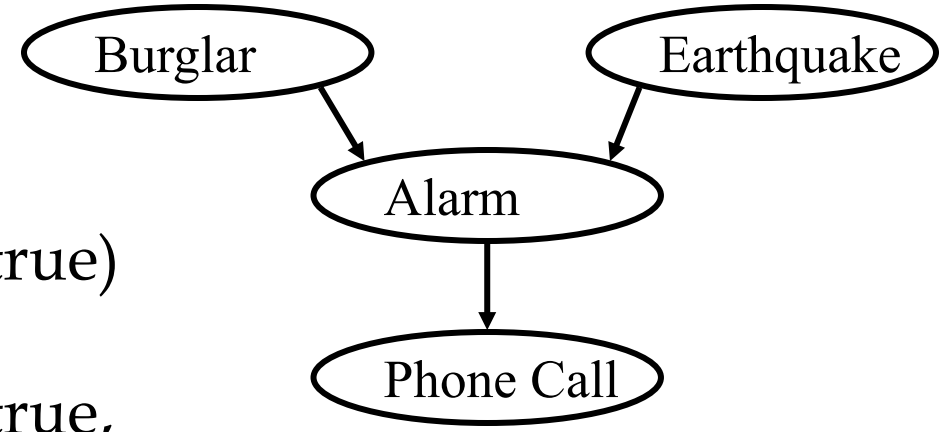
- Generative story (and joint distribution):
  - Pick b ~ Pr(Burglar), a binomial
  - Pick e ~ Pr(Earthquake), a binomial
  - Pick a ~ Pr(Alarm | Burglar=b, Earthquake=e), four binomials
  - Pick c ~ Pr(PhoneCall | Alarm)

DGMs: The "Burglar Alarm" example

You can also compute other quantities: e.g.,

Pr(Burglar=true | PhoneCall=true)

Pr(Burglar=true | PhoneCall=true, and Earthquake=true)



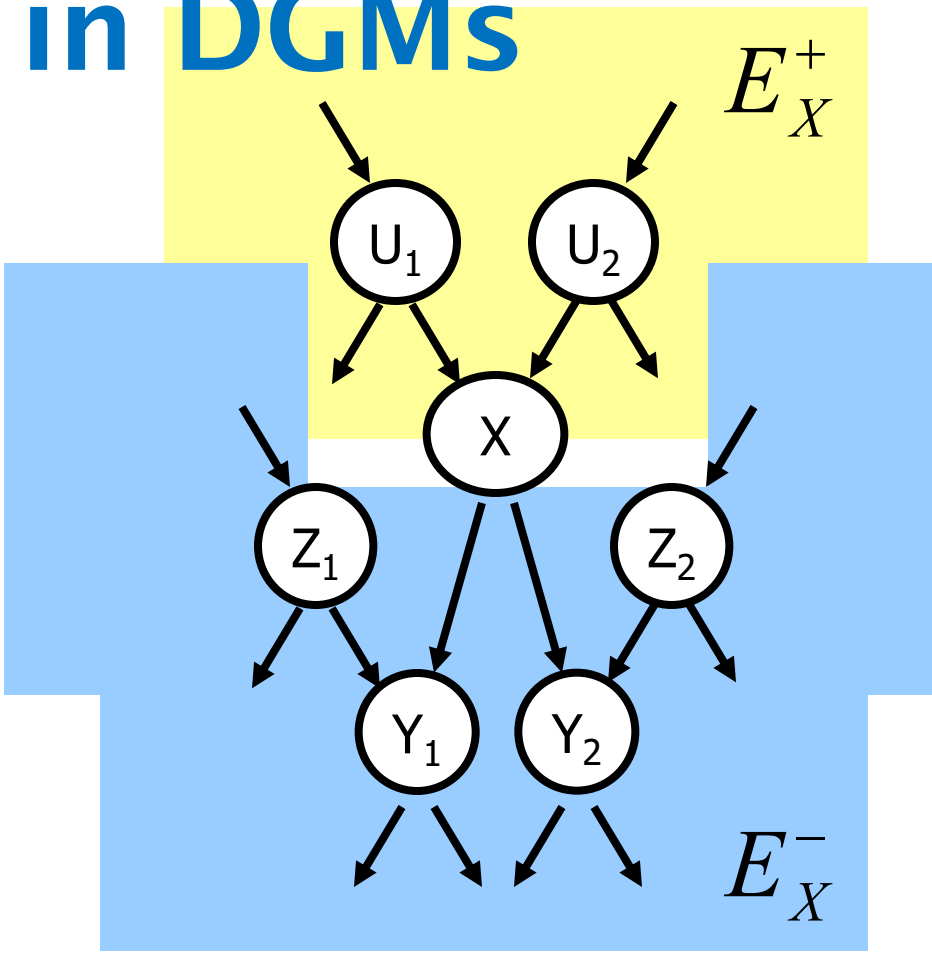
- Generative story:
  - Pick b ~ Pr(Burglar), a binomial
  - Pick e ~ Pr(Earthquake), a binomial
  - Pick a ~ Pr(Alarm | Burglar=b, Earthquake=e), four binomials
  - Pick c ~ Pr(PhoneCall | Alarm)

Inference in DGMs

- General problem: given evidence $E_1,\ldots,E_k$ compute $P(X \mid E_1,..,E_k)$ for any X
- Big assumption: graph is "polytree"
  - <=1 undirected path between any nodes X,Y
- Notation:

$$E_X^+ = \text{"causal support" for X}$$

$$E_X^- = \text{"evidential support" for X}$$

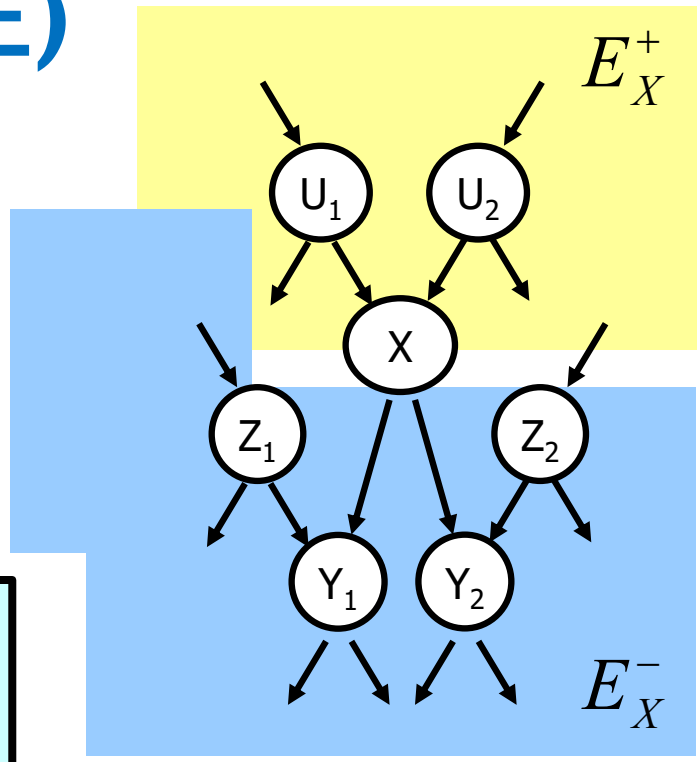


DGM Inference: P(X|E)

$$P(X \mid E) = P(X \mid E_X^+, E_X^-)$$

$$= \frac{P(E_X^- \mid X, E_X^+) P(X \mid E_X^+)}{P(E_X^- \mid E_X^+)}$$

$$P(X \mid E) \propto P(E_X^- \mid X) P(X \mid E_X^+)$$



E+: causal support
E-: evidential support

DGM Inference: P(X|E⁺)

$$P(X \mid E_X^+) =$$

$$\sum_{u_1,u_2} P(X \mid u_1, u_2, E_X^+) P(u_1, u_2 \mid E_X^+)$$

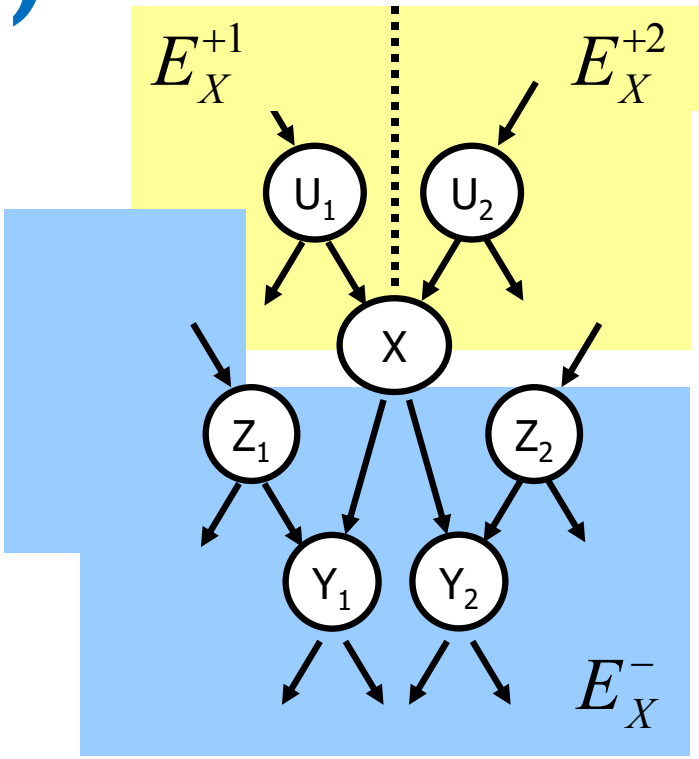
$$= \underbrace{\sum_{u_1,u_2} P(X \mid u_1, u_2,)} \underbrace{\prod_j P(u_j \mid E_X^{+j})}$$

CPT table lookup

Recursive call to $P(\cdot \mid E^+)$

$$= \sum_{\mathbf{u}} P(X \mid \mathbf{u}) \prod_j P(u_j \mid E_{Uj \setminus X})$$

Evidence for Uj that doesn't go thru X



So far: simple way of propagating requests for "belief due to causal evidence" **up** the tree

I.e. info on Pr(X | E+) flows down

Inference in Bayes nets: P(E⁻|X) simplified

$$P(X \mid E) \propto P(E_X^- \mid X) P(X \mid E_X^+)$$

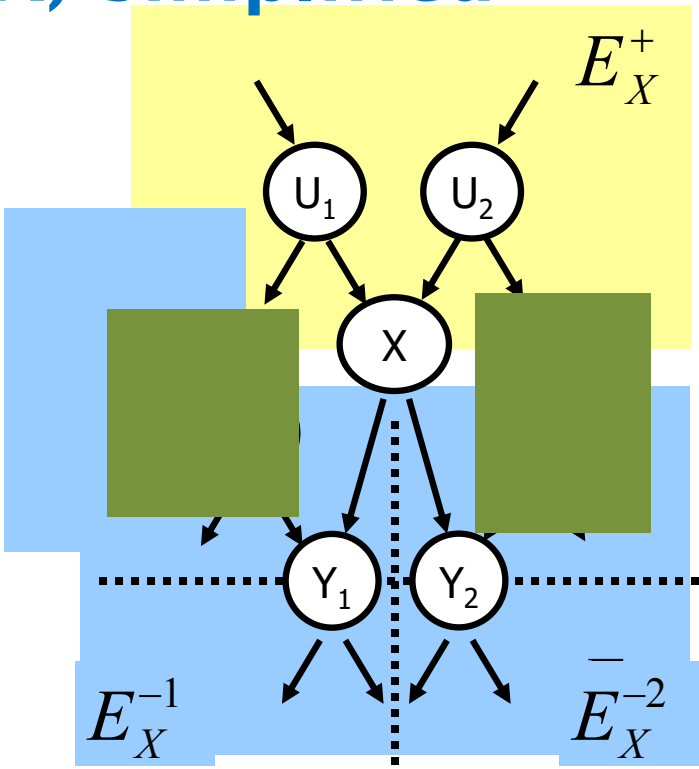
$$P(E_X^- \mid X) = \prod_j P(E_{Yj\backslash X}^- \mid X)$$

d-sep + polytree

$$= \prod_j \sum_{y_{jk}} P(y_{jk} \mid X) P(E_{Yj\backslash X}^- \mid X, y_{jk})$$

$$= \prod_j \sum_{y_{jk}} P(y_{jk} \mid X) \underbrace{P(E_{Yj}^{-j} \mid y_{jk})}$$

Recursive call to P(E⁻|·)



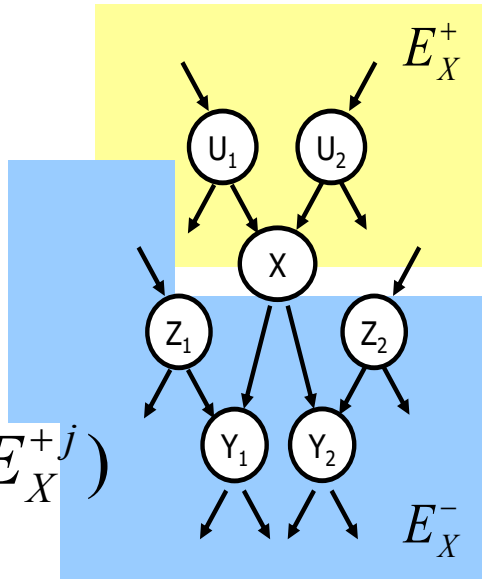
So far: simple way of propagating **requests** for "belief due to evidential support" **down** the tree

I.e. info on Pr(E-|X) flows up

$E_{Y_j}^{-j}$: evidential support for Y_j

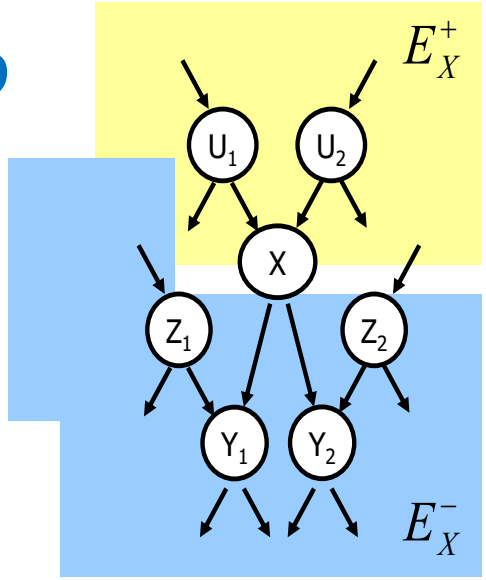
$E_{Y_j\backslash X}$: evidence for Y_j excluding support through X

Message Passing for BP



- We reduced P(X|E) to product of two recursively calculated parts:
  - $P(X{=}x \mid E^+) = \sum_{u_1,u_2} P(X \mid u_1, u_{2,}) \prod_j P(u_j \mid E_X^{+j})$
    - i.e., CPT for X and product of "forward" messages from parents
  - $P(E^- \mid X{=}x) = \beta \prod_j \sum_{y_{jk}} P(E_{Y_j}^{-j} \mid y_{jk}) \sum_{z_{jk}} P(y_{jk} \mid X, z_{jk}) P(z_{jk} \mid E_{Z_j \setminus Y_k})$
    - i.e., combination of "backward" messages from parents, CPTs, and $P(Z \mid E_{Z \setminus Yk})$, a simpler instance of P(X|E)
  - This can also be implemented by message-passing (belief propagation)
    - Messages are distributions – i.e., vectors

Message Passing for BP



- Top-level algorithm
  - Pick one vertex as the "root"
  - Any node with only one edge is a "leaf"
  - Pass messages from the leaves to the root
  - Pass messages from root to the leaves
  - Now every X has received P(X|E+) and P(E-|X) and can compute P(X|E)

NAÏVE BAYES AS A DGM

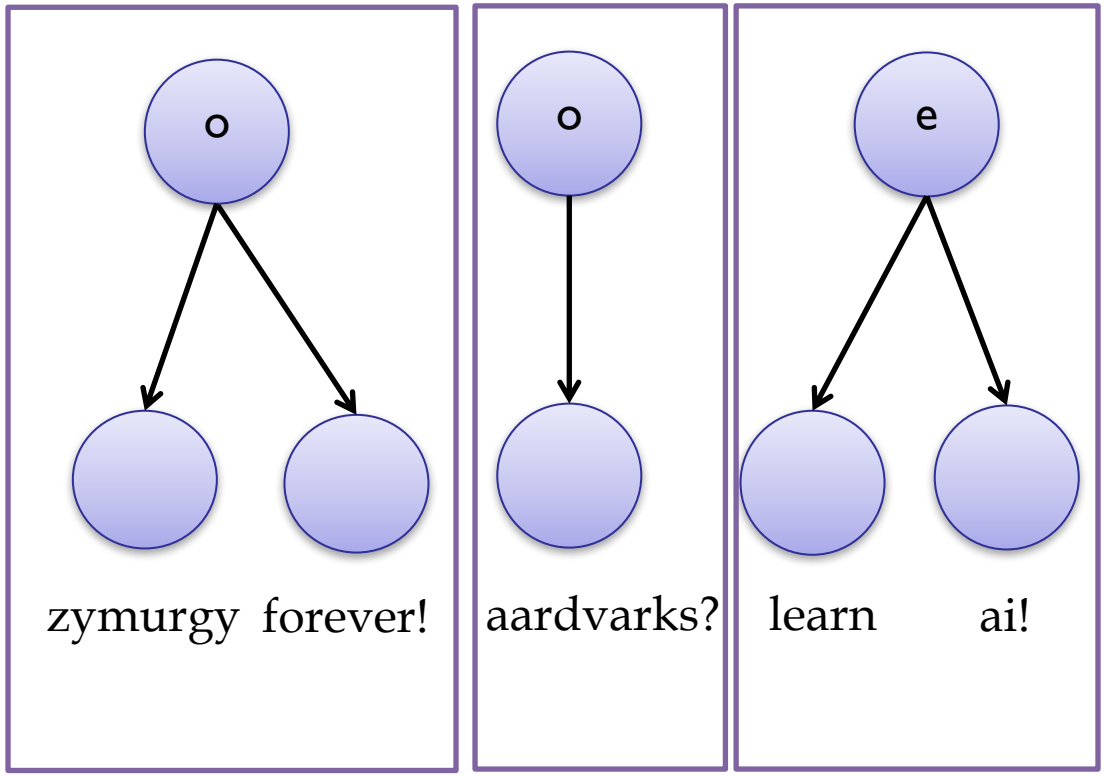
Naïve Bayes as a DGM

- For each document d in the corpus (of size D):
  - Pick a label y_d from $Pr(Y)$
  - For each word in d (of length N_d):
    - Pick a word x_{id} from $Pr(X \mid Y=y_d)$

	Pr(Y=y)
onion	0.3
economist	0.7

for every X

Y	X	Pr(X|y=y)
onion	aardvark	0.034
onion	ai	0.0067
…	…	…
economist	aardvark	0.0000003
….		
economist	zymurgy	0.01000



Naïve Bayes as a DGM

- For each document d in the corpus (of size D):
  - Pick a label y_d from $Pr(Y)$
  - For each word in d (of length N_d):
    - Pick a word x_{id} from $Pr(X \mid Y=y_d)$

	Pr(Y=y)
onion	0.3
economist	0.7

for every X

Y	X	Pr(X|y=y)
onion	aardvark	0.034
onion	ai	0.0067
…	…	…
economist	aardvark	0.0000003
….		
economist	zymurgy	0.01000

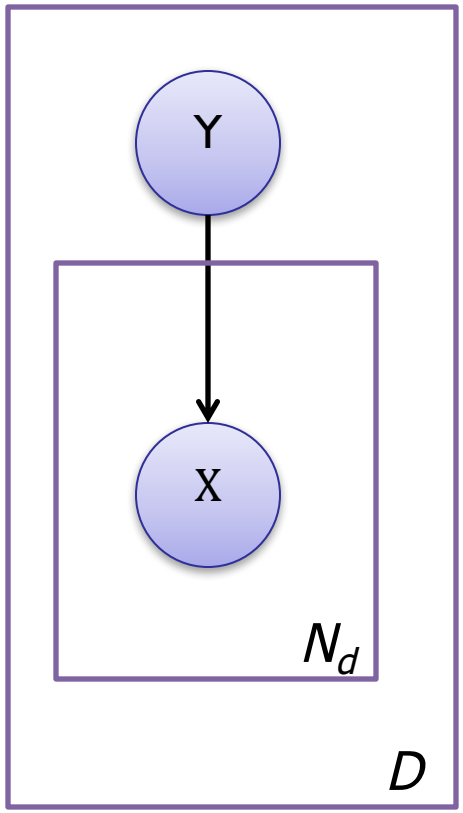


Plate diagram

Naïve Bayes as a DGM

- For each document d in the corpus (of size D):
  - Pick a label y_d from $Pr(Y)$
  - For each word in d (of length N_d):
    - Pick a word x_{id} from $Pr(X \mid Y=y_d)$

Not described: how do we smooth for classes? For multinomials? How many classes are there? ….

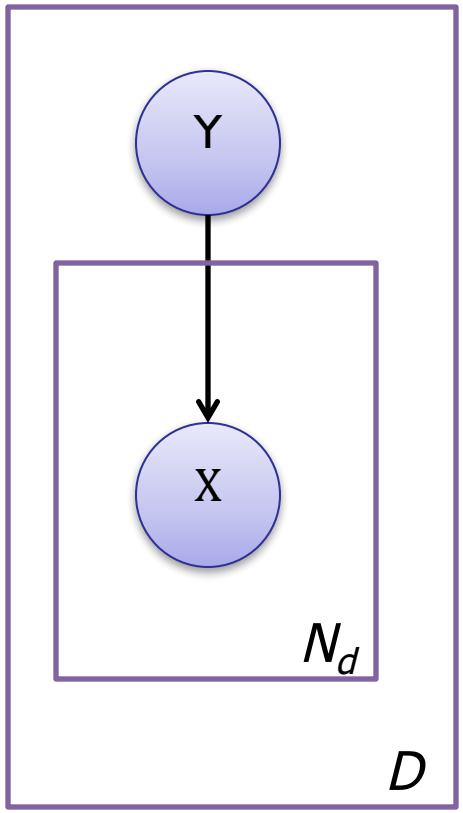
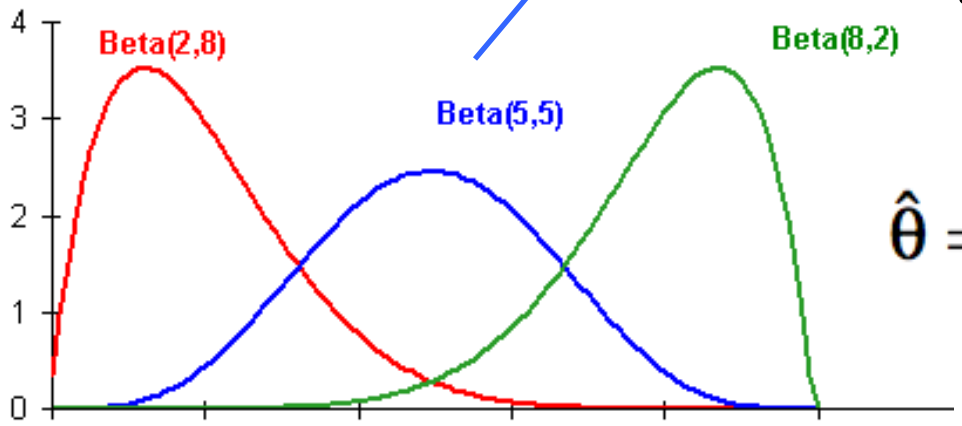


Plate diagram

Recap: smoothing for a binomial

MLE: maximize $\Pr(D|\theta)$

MAP: maximize $\Pr(D|\theta)\Pr(\theta)$



estimate Θ=P(heads) for a binomial with MLE as:

$$\hat{\theta} = \frac{\alpha_1}{\alpha_1 + \alpha_0}$$

#heads

#tails

and with MAP as:

$$\hat{\theta} = \frac{(\alpha_1 + \gamma_1)}{(\alpha_1 + \gamma_1) + (\alpha_0 + \gamma_0)}$$

#imaginary heads

#imaginary tails

Smoothing = prior over the **parameter** θ

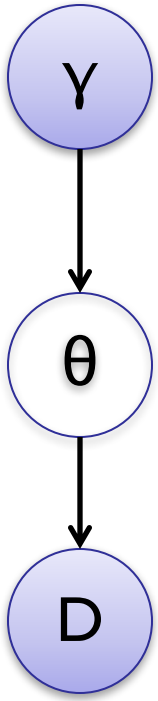
Smoothing for a binomial as a DGM

MAP for dataset D with α_1 heads and α_2 tails:

#imaginary heads

$$\hat{\theta} = \frac{(\alpha_1 + \gamma_1)}{(\alpha_1 + \gamma_1) + (\alpha_0 + \gamma_0)}$$

#imaginary tails



MAP is a Viterbi-style inference: want to find max probability parameter θ, to the posterior distribution

Also: inference in a simple graph can be intractable if the conditional distributions are complicated

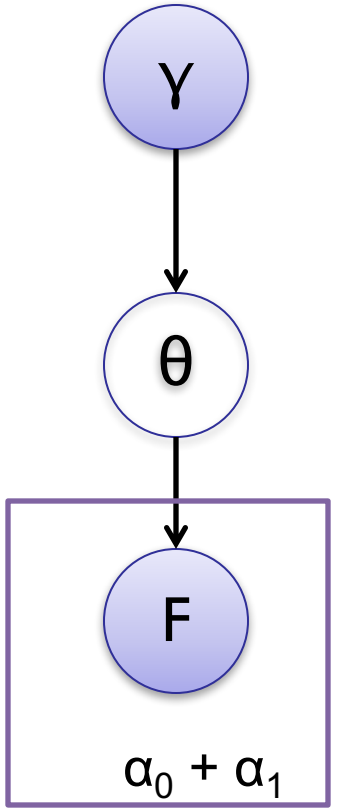
Smoothing for a binomial as a DGM

MAP for dataset D with α_1 heads and α_2 tails:

#imaginary heads

$$\hat{\theta} = \frac{(\alpha_1 + \gamma_1)}{(\alpha_1 + \gamma_1) + (\alpha_0 + \gamma_0)}$$

#imaginary tails



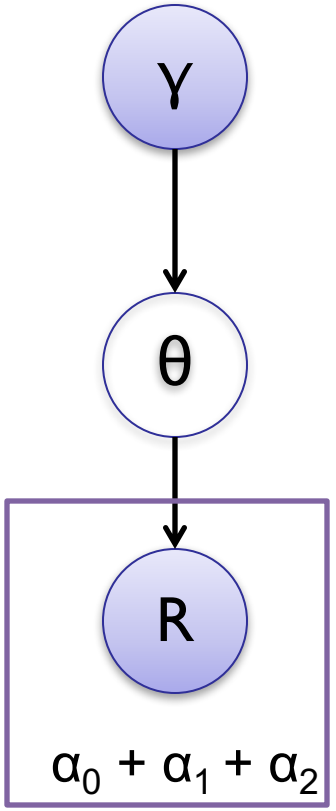
Comment: conjugate for a multinomial is called a **Dirichlet**

Smoothing for a binomial as a DGM

MAP for dataset D with α_1 heads and α_2 tails:

#imaginary rolls of *i*

$$\theta_i = \frac{\alpha_i + \gamma_i}{\sum_j (\alpha_j + \gamma_j)}$$



Comment: conjugate for a multinomial is called a **Dirichlet**

Recap: Naïve Bayes as a DGM

Now: let's turn Bayes up to 11 for naïve Bayes….

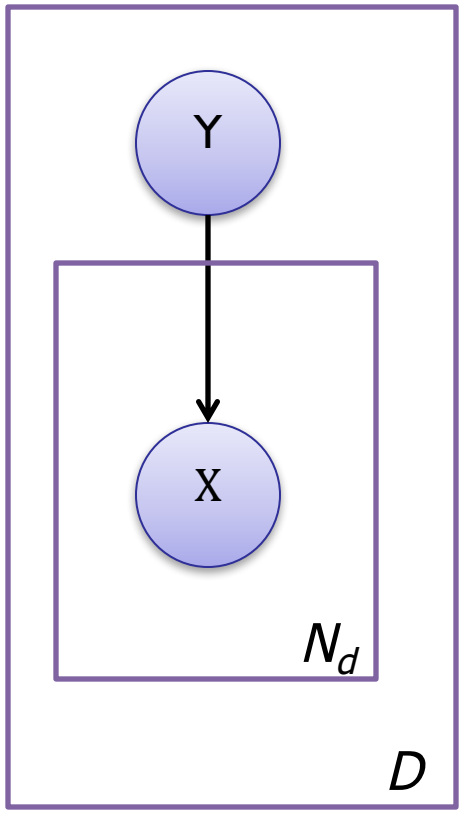
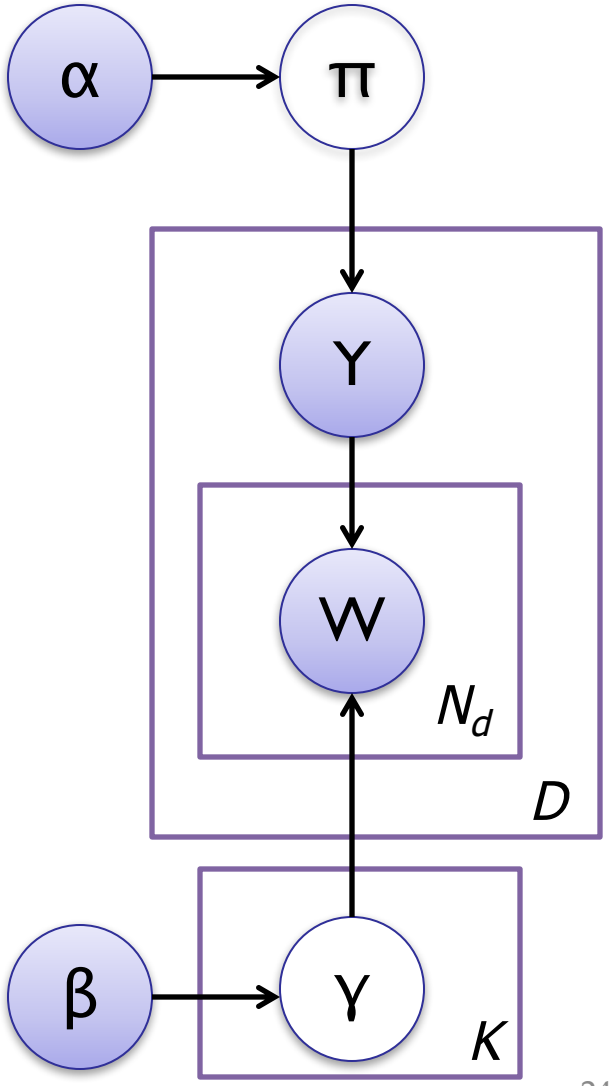


Plate diagram

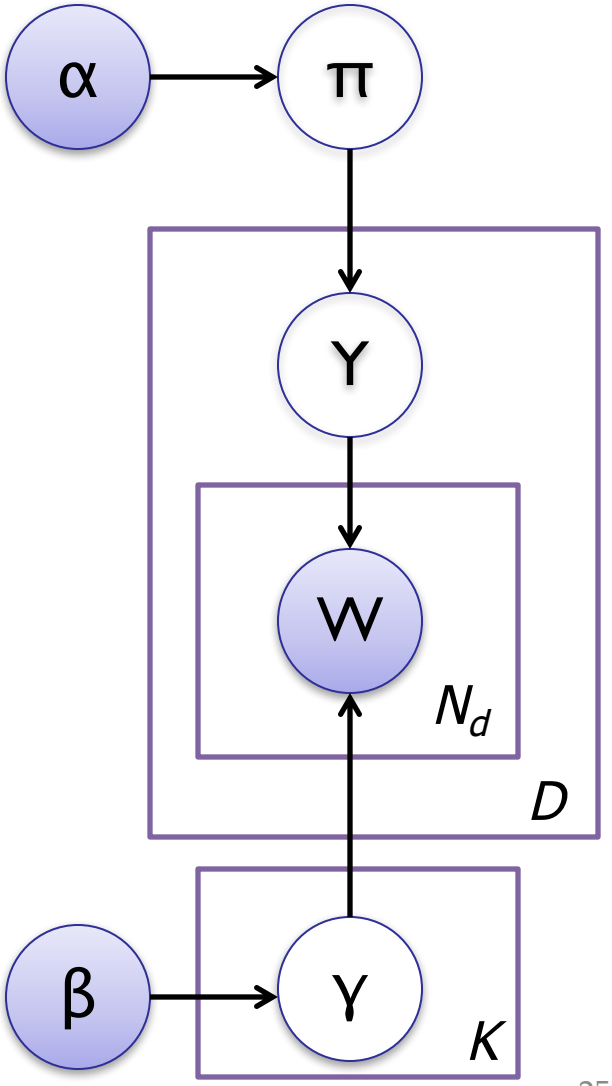
A more Bayesian Naïve Bayes

- From a Dirichlet α:
  - Draw a multinomial π over the K classes
- From a Dirichlet β
  - For each class y=1…K
    - Draw a multinomial γ[y] over the vocabulary
- For each document d=1..D:
  - Pick a label y_d from π
  - For each word in d (of length N_d):
    - Pick a word x_{id} from γ[y_d]



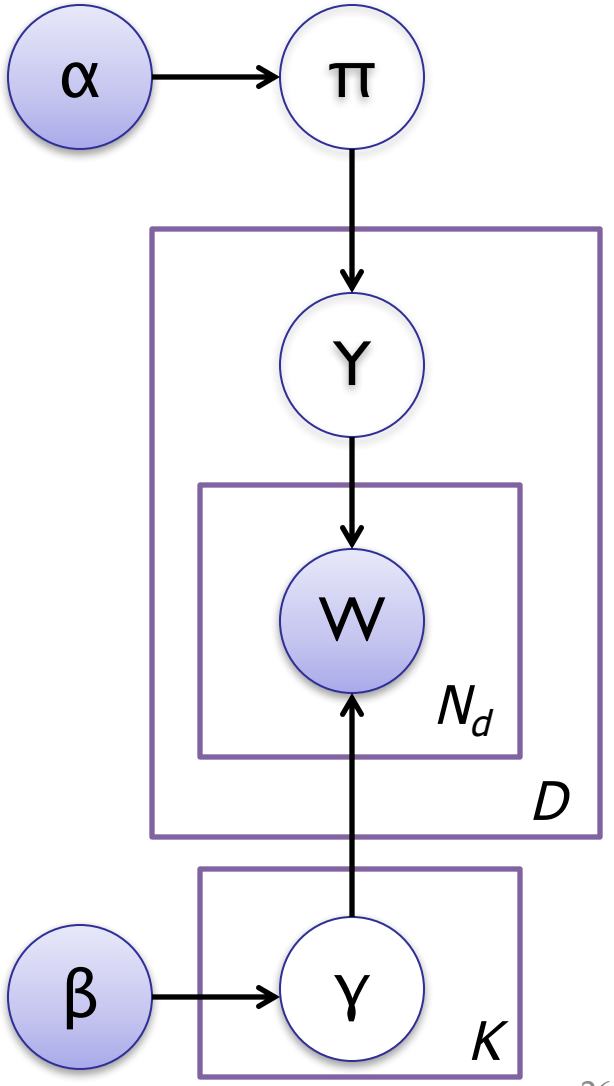
Unsupervised Naïve Bayes

- From a Dirichlet α:
  - Draw a multinomial π over the K classes
- From a Dirichlet β
  - For each class y=1…K
    - Draw a multinomial γ[y] over the vocabulary
- For each document d=*1..D*:
  - Pick a label y_d from π
  - For each word in d (of length N_d):
    - Pick a word x_{id} from γ[y_d]



Unsupervised Naïve Bayes

- The generative story is the same
- What we do is different
  - We want to both find values of γ and π **and** find values for Y for each example.
- EM is a natural algorithm



LDA (AS A DGM)

The LDA Topic Model

Latent Dirichlet Allocation

David M. Blei
Computer Science Division
University of California
Berkeley, CA 94720, USA

Andrew Y. Ng
Computer Science Department
Stanford University
Stanford, CA 94305, USA

Michael I. Jordan
Computer Science Division and Department of Statistics
University of California
Berkeley, CA 94720, USA

Scholar

About 22,800 results (**0.08** sec)

Articles

Case law

My library

Any time

Since 2016

Since 2015

Since 2012

Custom range...

Sort by relevance

Sort by date

include patents

include citations

Latent dirichlet allocation

DM Blei, AY Ng, MI Jordan - the Journal of machine Learning research, 2003 - dl.acm.org

Abstract We describe latent Dirichlet allocation (LDA), a generative probabilistic model for collections of discrete data such as text corpora. LDA is a three-level hierarchical Bayesian model, in which each item of a collection is modeled as a finite mixture over an underlying ...

Cited by 14019 Related articles All 141 versions Web of Science: 3390 Cite Saved

Cited by 20707

[PDF] Latent dirichlet allocation

DM Blei, AY Ng, MI Jordan - Advances in neural …, 2001 - machinelearning.wustl.edu

Я джгдгз в ж и к бг а гж и ми в ги ж гаа и гвз г зЙ ж и и и и в ж а о з гж бджгк з гв з к ж а дж гйз бг аз в ай в в к н зЛйв ж бИ б мийж г йв ж бзℓ И в Рг Й б ввГз зд и бг аИ азг вглв з джг а зи а и ви з б ви в м в ДдФЫСЕ Пℓ К Св и гви ми г и ми бг а в И гйж бг а дгз из и и г йб ...

Cited by 262 Related articles All 8 versions Cite Save More

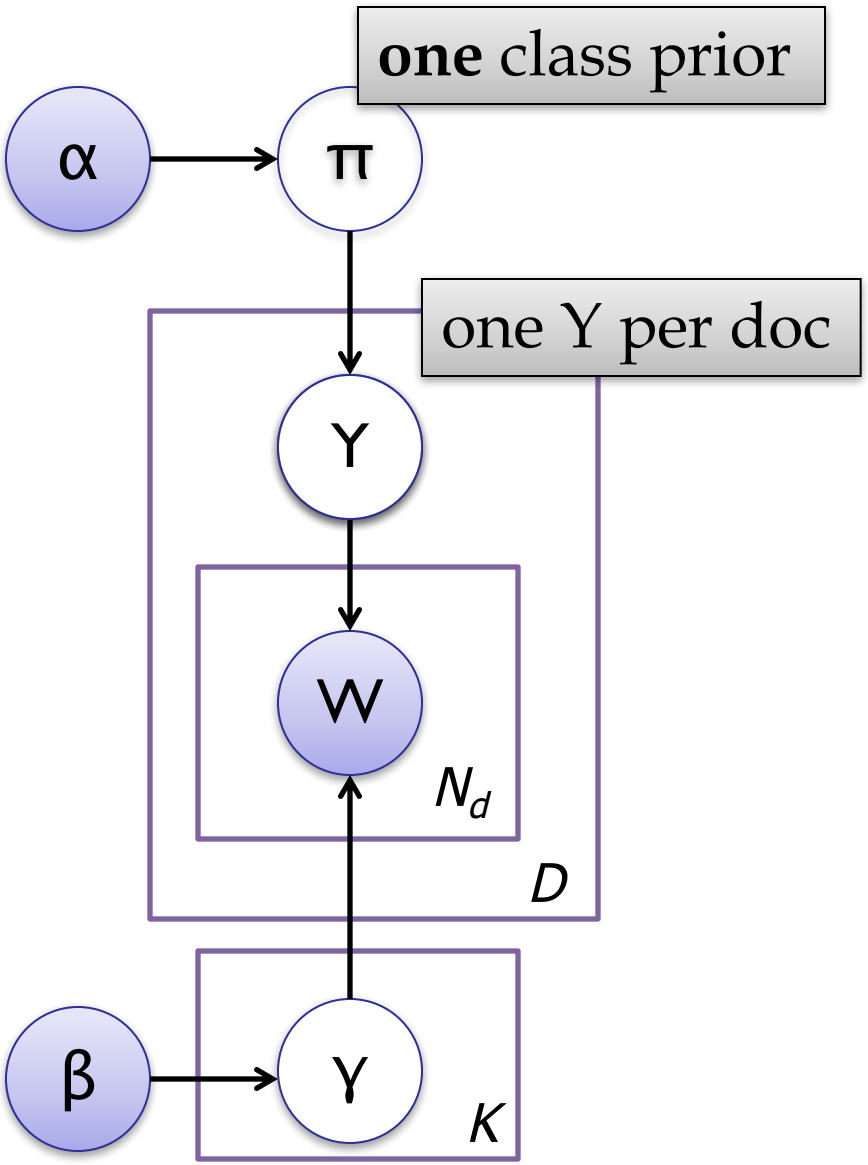
[HTML] Online learning for latent dirichlet allocation

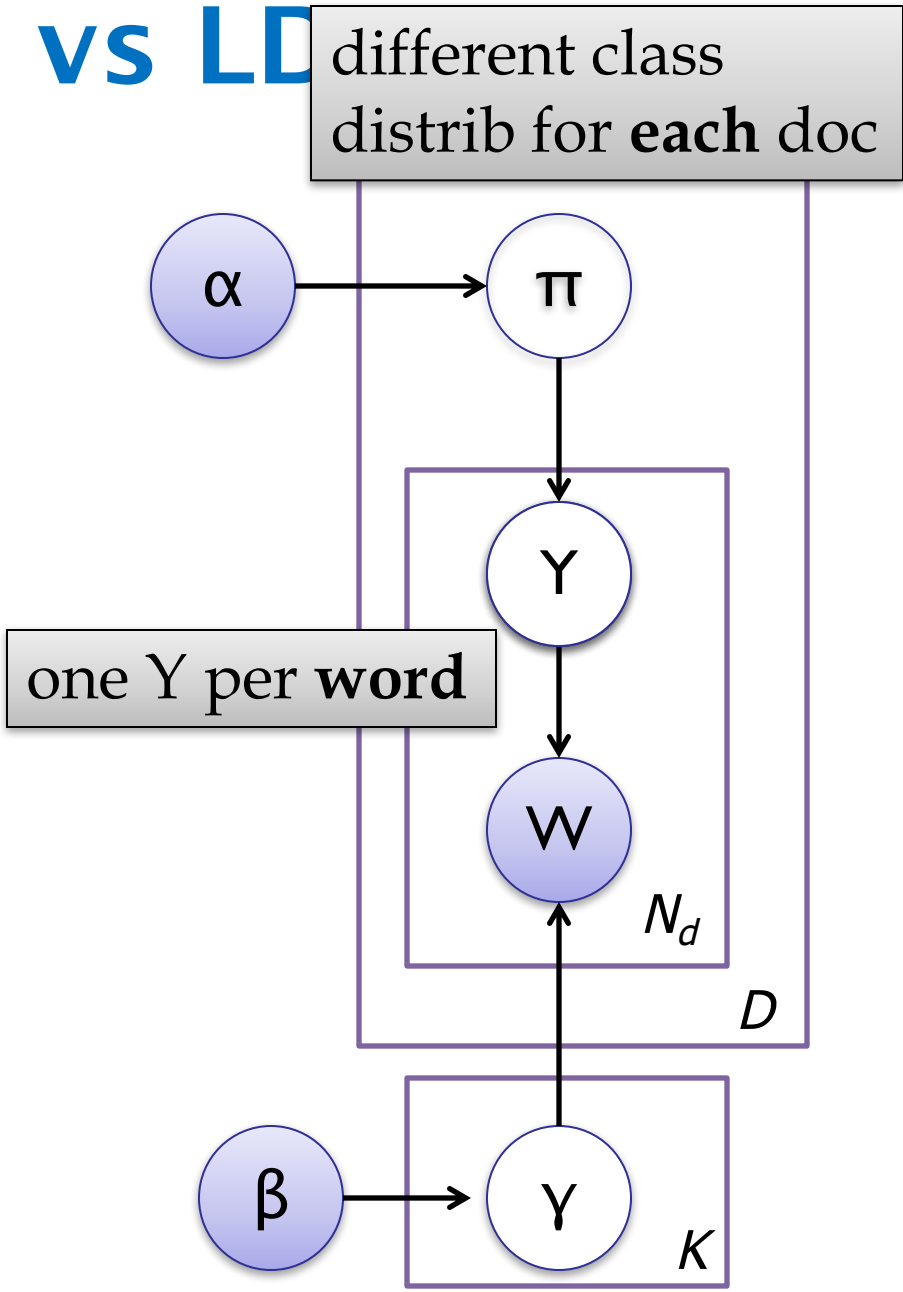
M Hoffman, FR Bach, DM Blei - advances in neural information …, 2010 - papers.nips.cc

Abstract We develop an online variational Bayes (VB) algorithm for **Latent Dirichlet Allocation** (LDA). Online LDA is based on online stochastic optimization with a natural gradient step, which we show converges to a local optimum of the VB objective function. It **...**

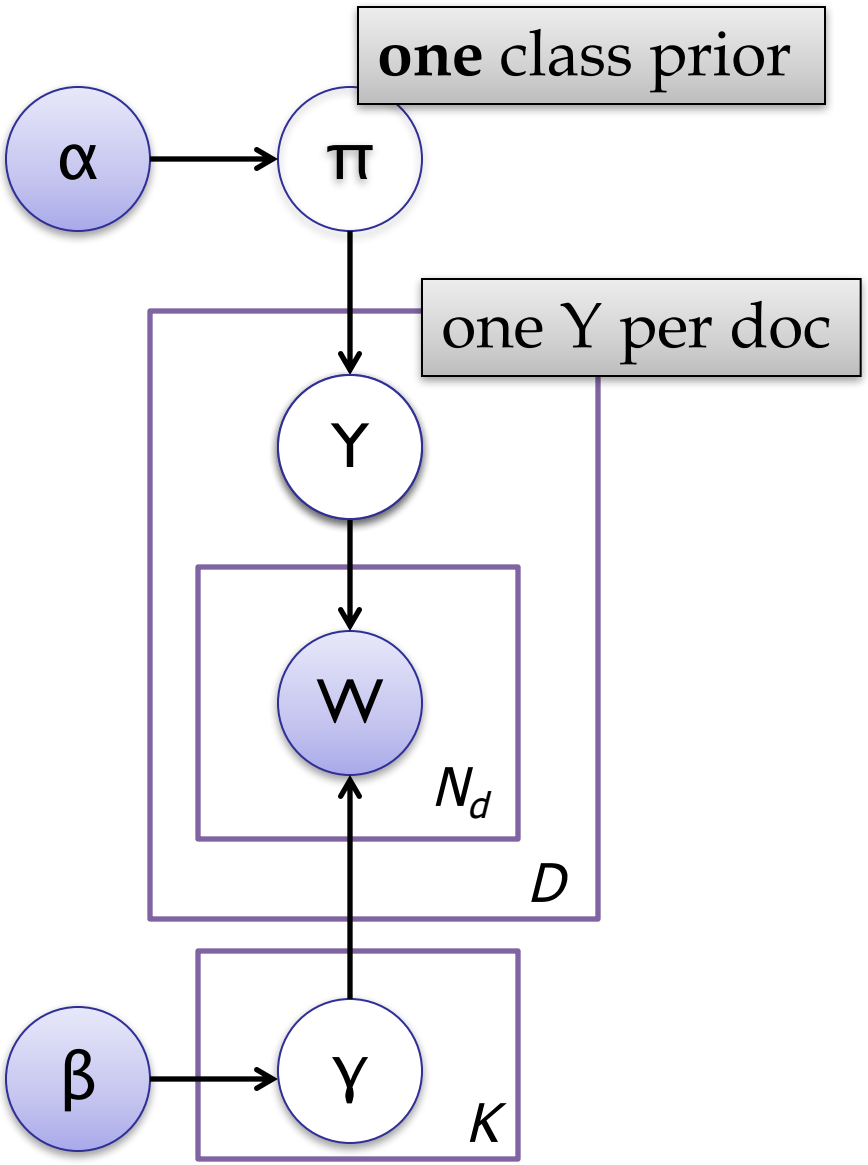
Cited by 559 Related articles All 18 versions Cite Save

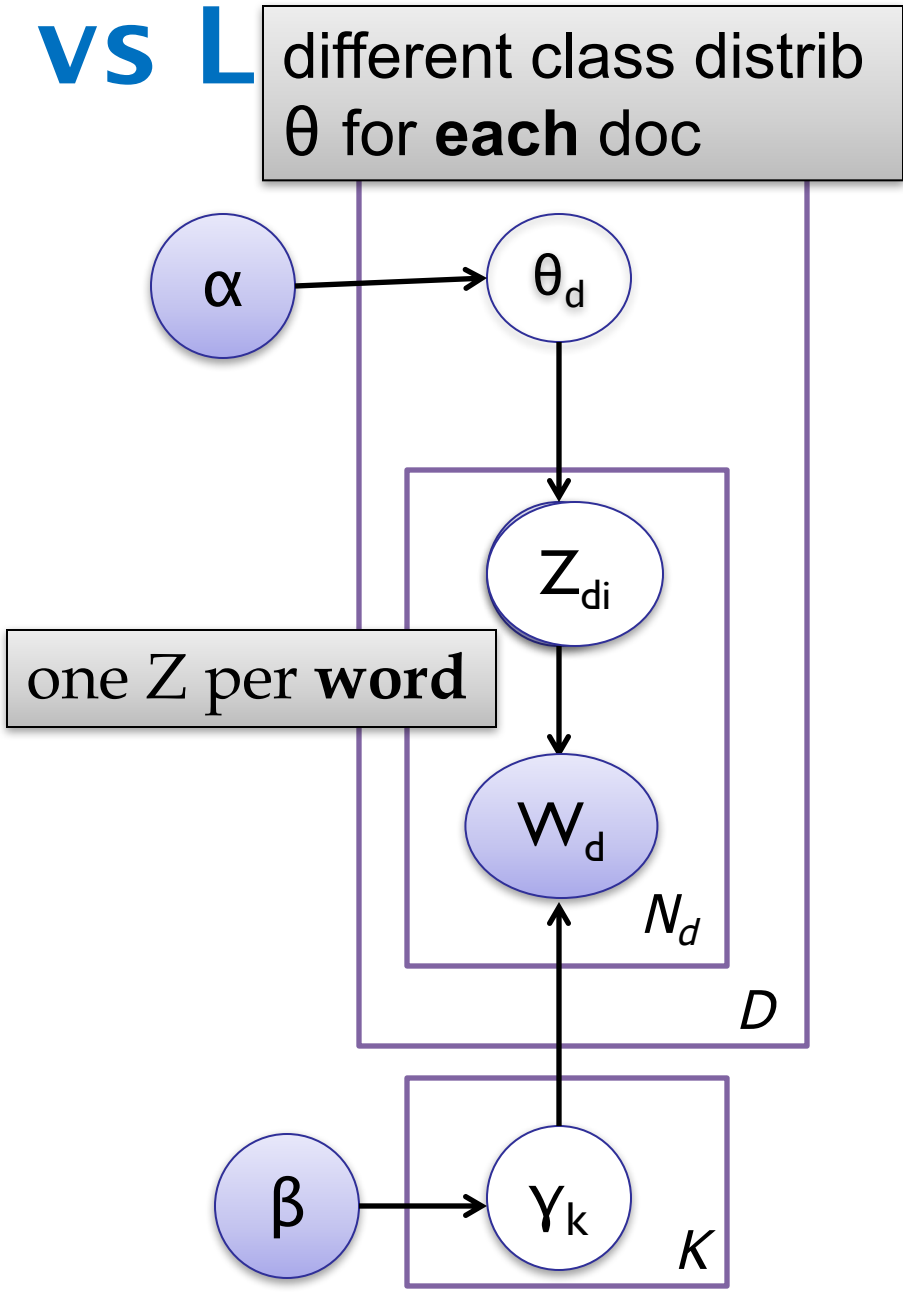
Unsupervised NB vs LDA





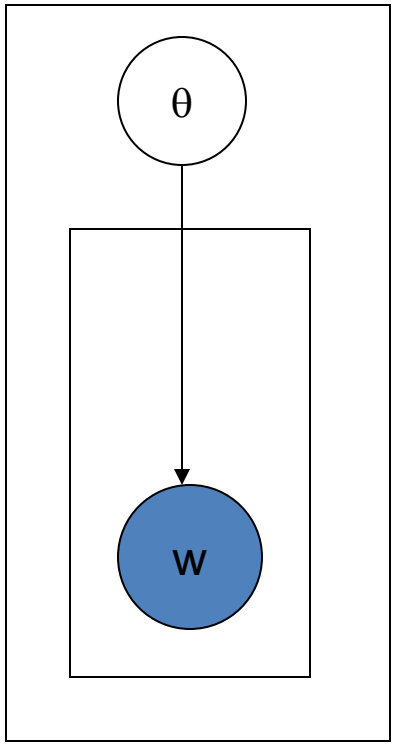
Unsupervised NB vs L





LDA

Blei's motivation: start with BOW assumption

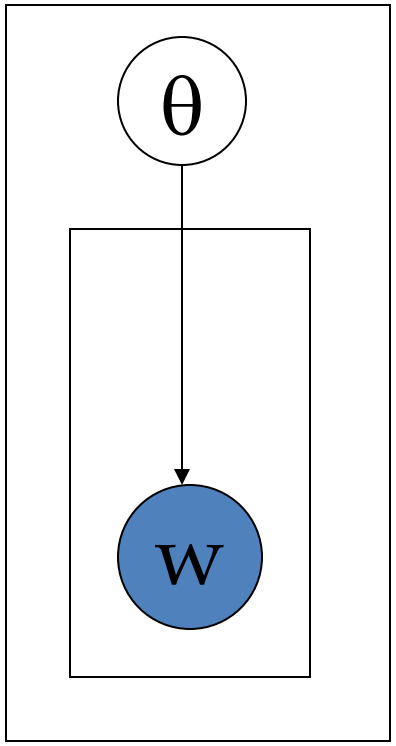


Assumptions: 1) documents are i.i.d 2) *within* a document, words are i.i.d. (bag of words)

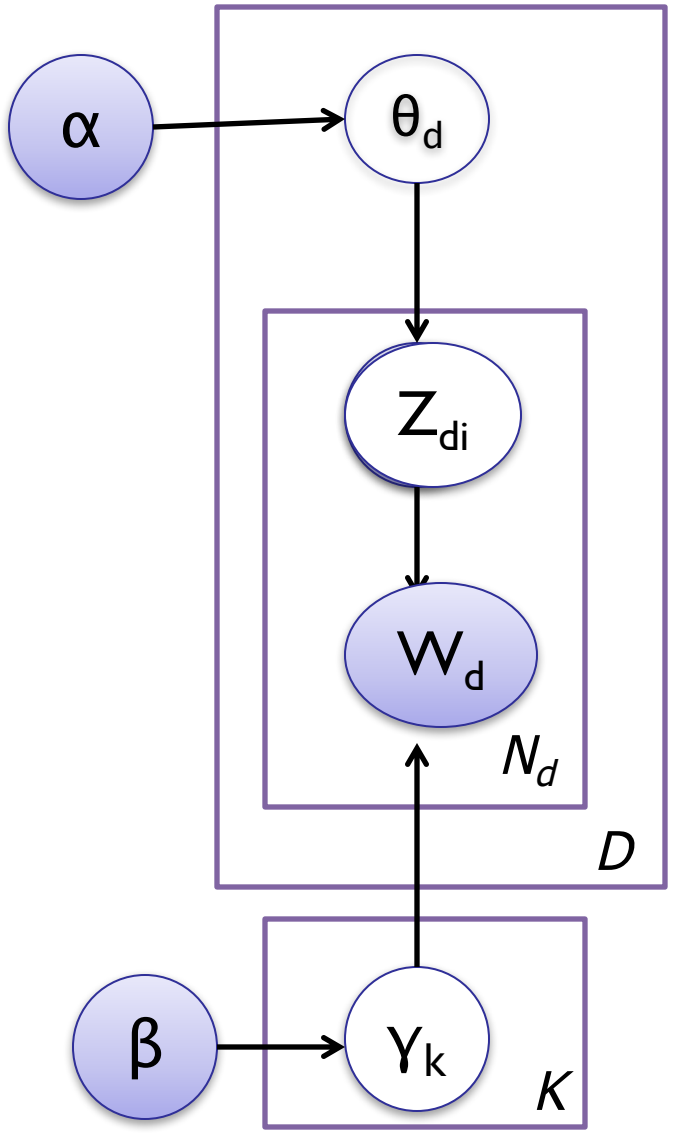
- For each document d = 1,⋯,M
  - Generate $\theta_d \sim D_1(\ldots)$
  - For each word n = 1,⋯, N_d
    - generate $w_n \sim D_2(\ . \mid \vartheta_{d_n})$

Now pick your favorite distributions for D_1, D_2

Unsupervised NB vs LDA



- Unsupervised NB clusters documents into latent classes
- LDA clusters word occurrences into latent classes (topics)
- The smoothness of β_k → same word suggests same topic
- The smoothness of θ_d → same document suggests same topic



- LDA's view of a document

Mixed membership model

The William Randolph Hearst Foundation will give $1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. "Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services," Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center's share will be $200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive $400,000 each. The Juilliard School, where music and the performing arts are taught, will get $250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual $100,000 donation, too.

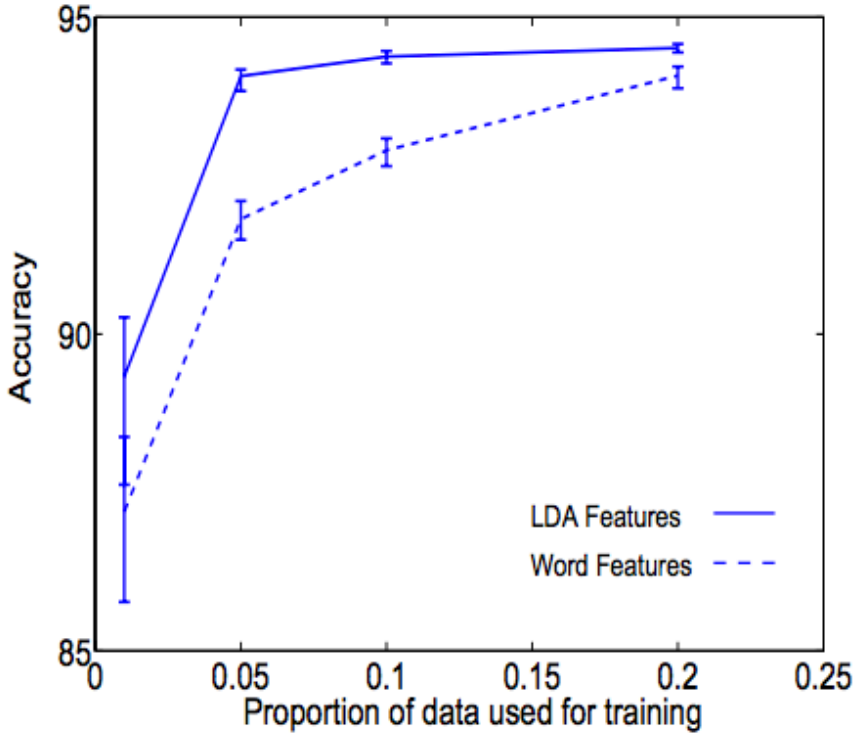
"Arts" "Budgets" "Children" "Education"

- LDA topics: top words w by Pr(w|Z=k)

Z=13	Z=22	Z=27	Z=19
"Arts"	"Budgets"	"Children"	"Education"
NEW	MILLION	CHILDREN	SCHOOL
FILM	TAX	WOMEN	STUDENTS
SHOW	PROGRAM	PEOPLE	SCHOOLS
MUSIC	BUDGET	CHILD	EDUCATION
MOVIE	BILLION	YEARS	TEACHERS
PLAY	FEDERAL	FAMILIES	HIGH
MUSICAL	YEAR	WORK	PUBLIC
BEST	SPENDING	PARENTS	TEACHER
ACTOR	NEW	SAYS	BENNETT
FIRST	STATE	FAMILY	MANIGAT
YORK	PLAN	WELFARE	NAMPHY
OPERA	MONEY	MEN	STATE
THEATER	PROGRAMS	PERCENT	PRESIDENT
ACTRESS	GOVERNMENT	CARE	ELEMENTARY
LOVE	CONGRESS	LIFE	HAITI

SVM using 50 features: $Pr(Z=k|\theta_d)$

50 topics vs all words, SVM



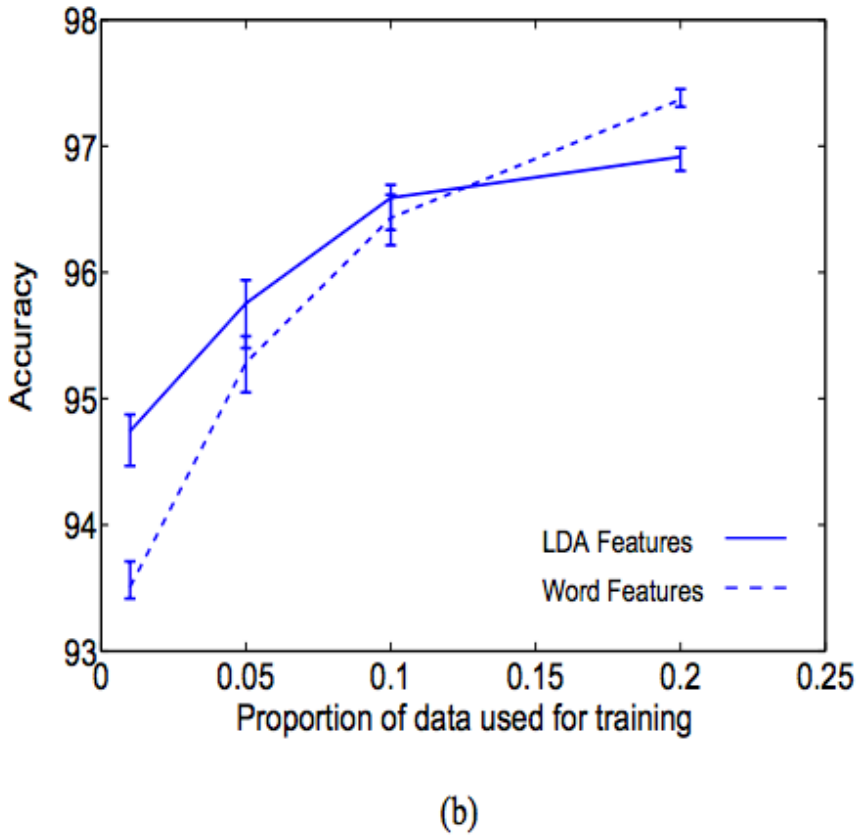


Figure 10: Classification results on two binary classification problems from the Reuters-21578 dataset for different proportions of training data. Graph (a) is EARN vs. NOT EARN. Graph (b) is GRAIN vs. NOT GRAIN.

Gibbs Sampling for LDA

LDA

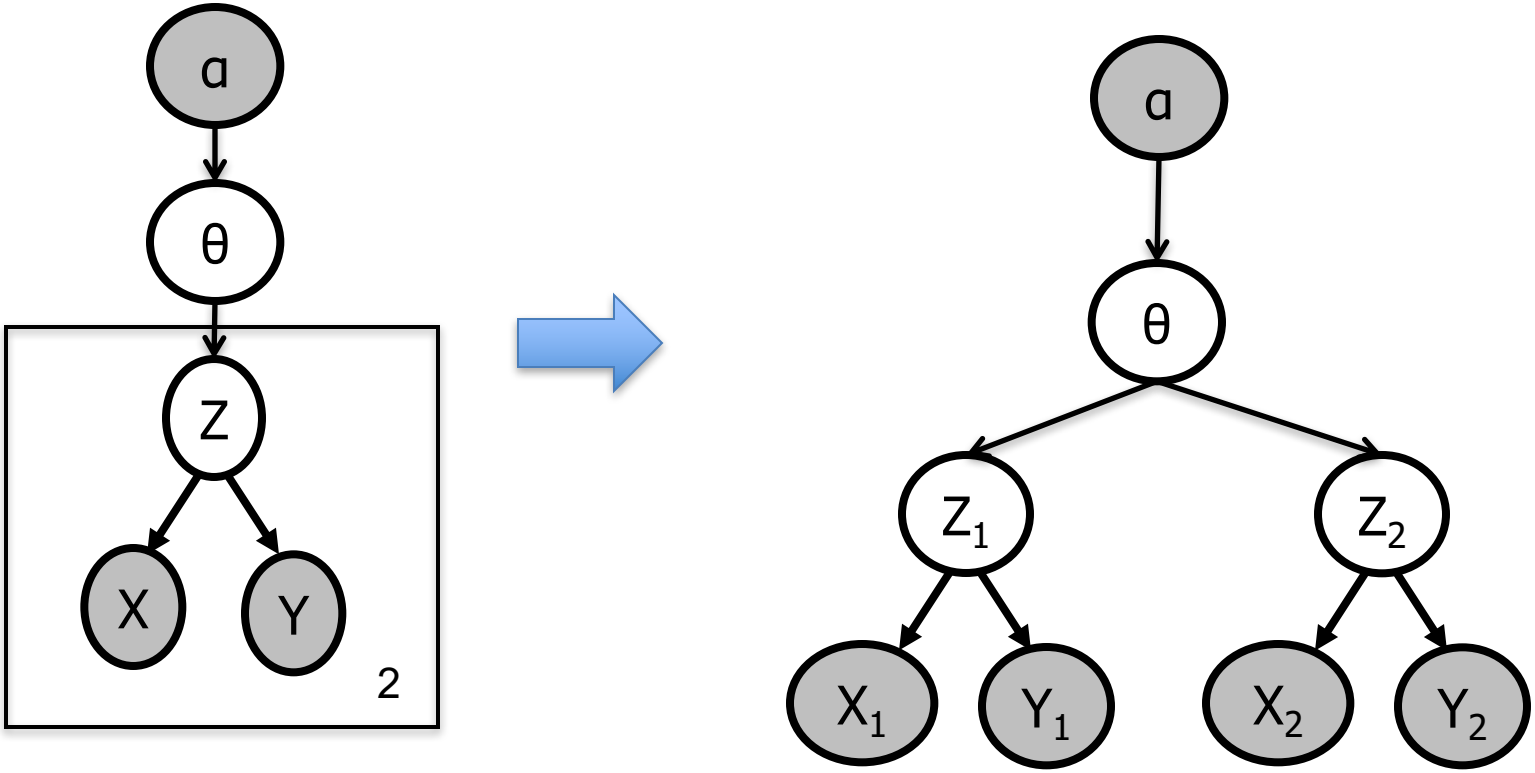
- Latent Dirichlet Allocation
  - Parameter learning:
    - Variational EM
      - Not covered in 601-B
    - Collapsed Gibbs Sampling
      - Wait, why is sampling called “learning” here?
      - Here’s the idea….

LDA

- Gibbs sampling – works for *any* directed model!
  - Applicable when joint distribution is hard to evaluate but conditional distribution is known
  - Sequence of samples comprises a Markov Chain
  - Stationary distribution of the chain is the joint distribution

1. Initialise $x_{0,1:n}$.
2. For $i = 0$ to $N - 1$
   - Sample $x_1^{(i+1)} \sim p(x_1 | x_2^{(i)}, x_3^{(i)}, \ldots, x_n^{(i)})$.
   - Sample $x_2^{(i+1)} \sim p(x_2 | x_1^{(i+1)}, x_3^{(i)}, \ldots, x_n^{(i)})$.
   - $\vdots$
   - Sample $x_j^{(i+1)} \sim p(x_j | x_1^{(i+1)}, \ldots, x_{j-1}^{(i+1)}, x_{j+1}^{(i)}, \ldots, x_n^{(i)})$.
   - $\vdots$
   - Sample $x_n^{(i+1)} \sim p(x_n | x_1^{(i+1)}, x_2^{(i+1)}, \ldots x_{n-1}^{(i+1)})$.

Key capability: estimate distribution of **one** latent variables given **the other latent variables** and observed variables.



I'll assume we know parameters for Pr(X|Z) and Pr(Y|Z)

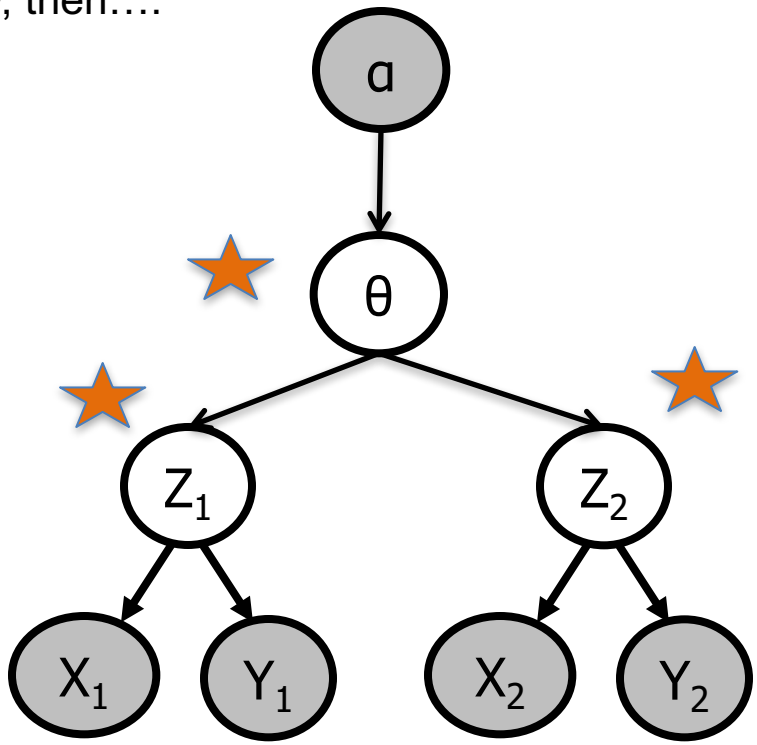
Initialize all the hidden variables randomly, then….

Pick $Z_1 \sim Pr(Z|x_1,y_1,\theta)$

Pick $\theta \sim Pr(z_1, z_2,\alpha)$
pick from posterior

Pick $Z_2 \sim Pr(Z|x_2,y_2,\theta)$

.
.
.



Pick $Z_1 \sim Pr(Z|x_1,y_1,\theta)$

Pick $\theta \sim Pr(z_1, z_2,\alpha)$
pick from posterior

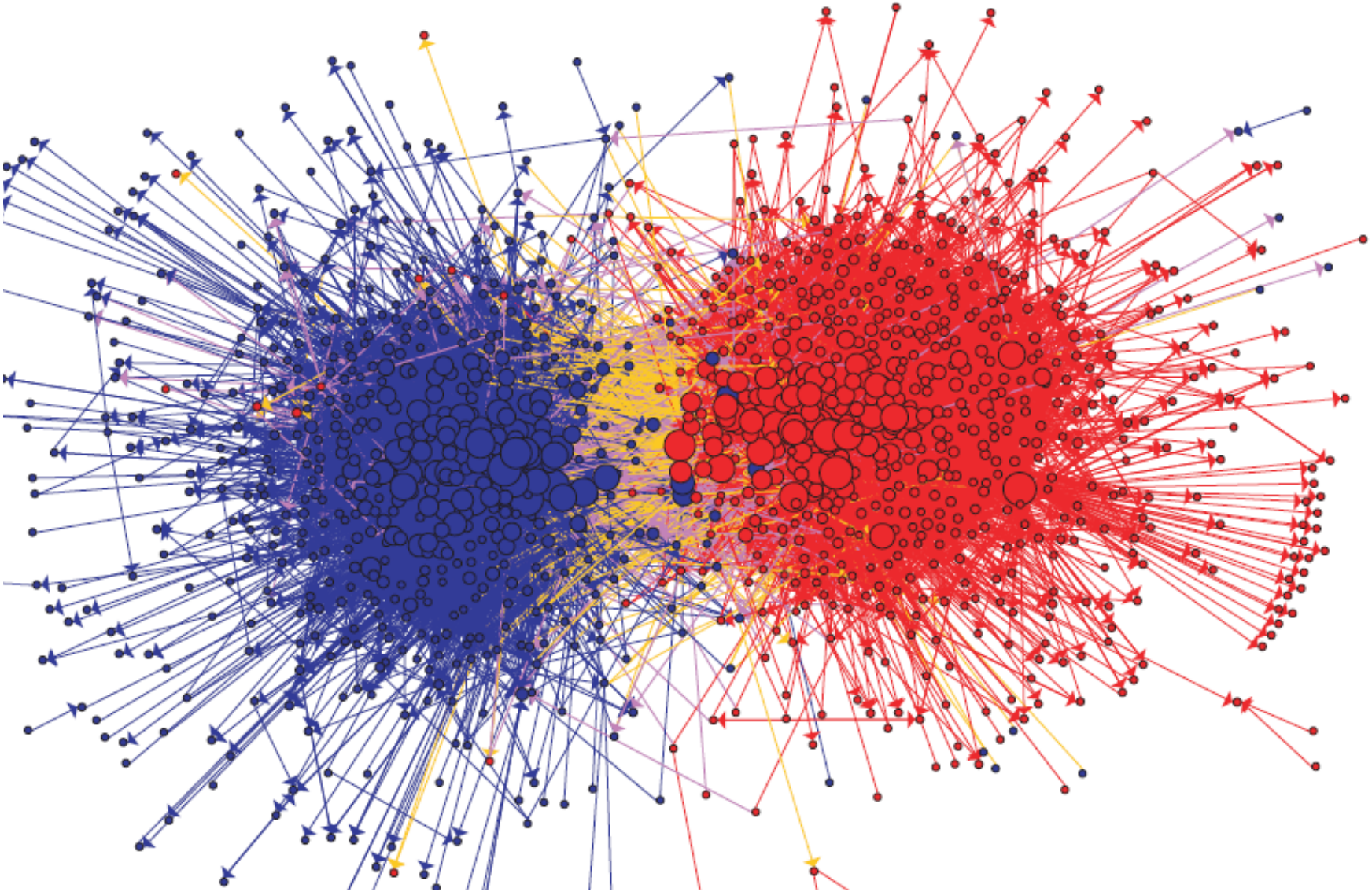
Pick $Z_2 \sim Pr(Z|x_2,y_2,\theta)$

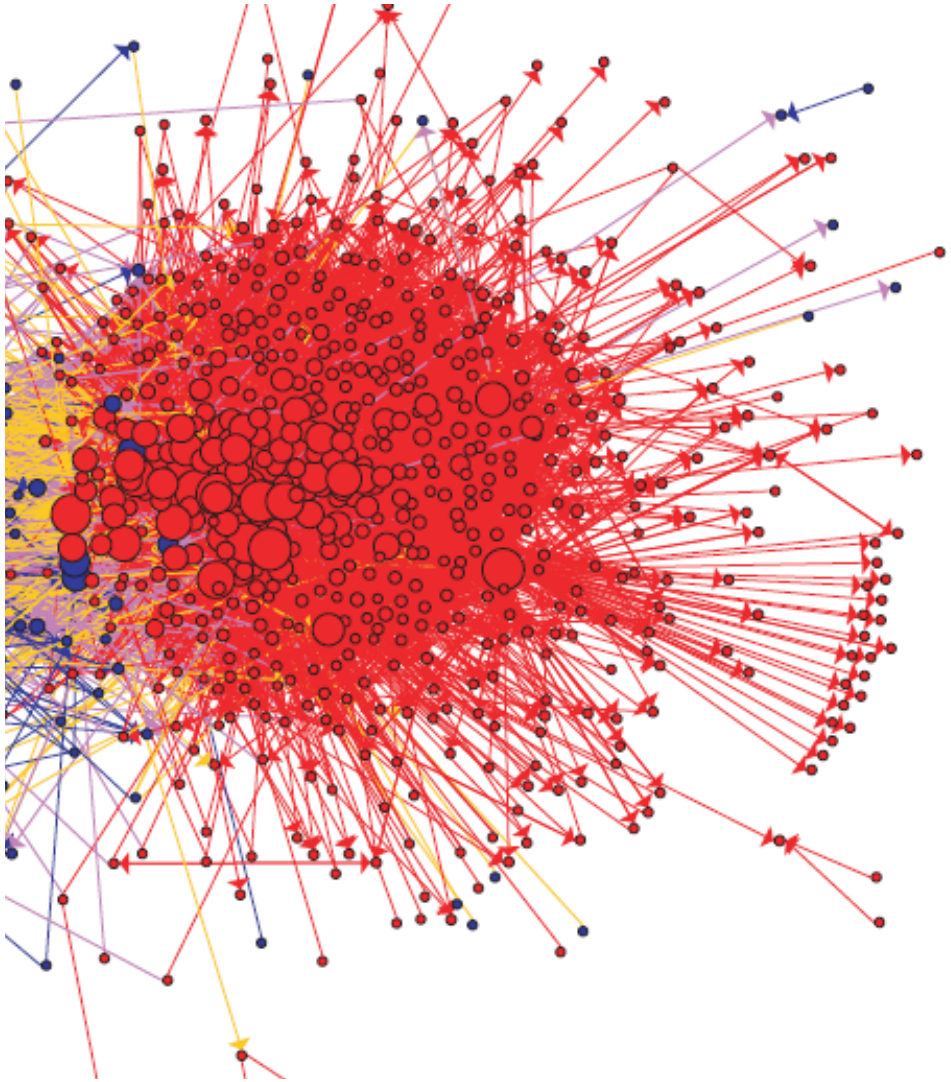
So we will have (a sample of) the true θ

in a broad range of cases eventually these will converge to samples from the **true joint distribution**

Why does Gibbs sampling work?

- Basic claim: when you sample $x \sim P(X|y_1,\ldots,y_k)$ then ***if*** $y_1,\ldots,y_k$ were sampled from the true joint ***then*** x will be sampled from the true joint
- So the true joint is a "fixed point"
  - you tend to stay there if you ever get there
- How long does it take to get there?
  - depends on the structure of the space of samples: how well-connected are they by the sampling steps?





LDA

- Latent Dirichlet Allocation
  - Parameter learning:
    - Variational EM
      - Not covered in 601-B
    - Collapsed Gibbs Sampling
      - What is collapsed Gibbs sampling?

Initialize all the Z's randomly, then….

Pick $Z_1 \sim Pr(Z_1|z_2,z_3,\alpha)$

Pick $Z_2 \sim Pr(Z_2|z_1,z_3,\alpha)$

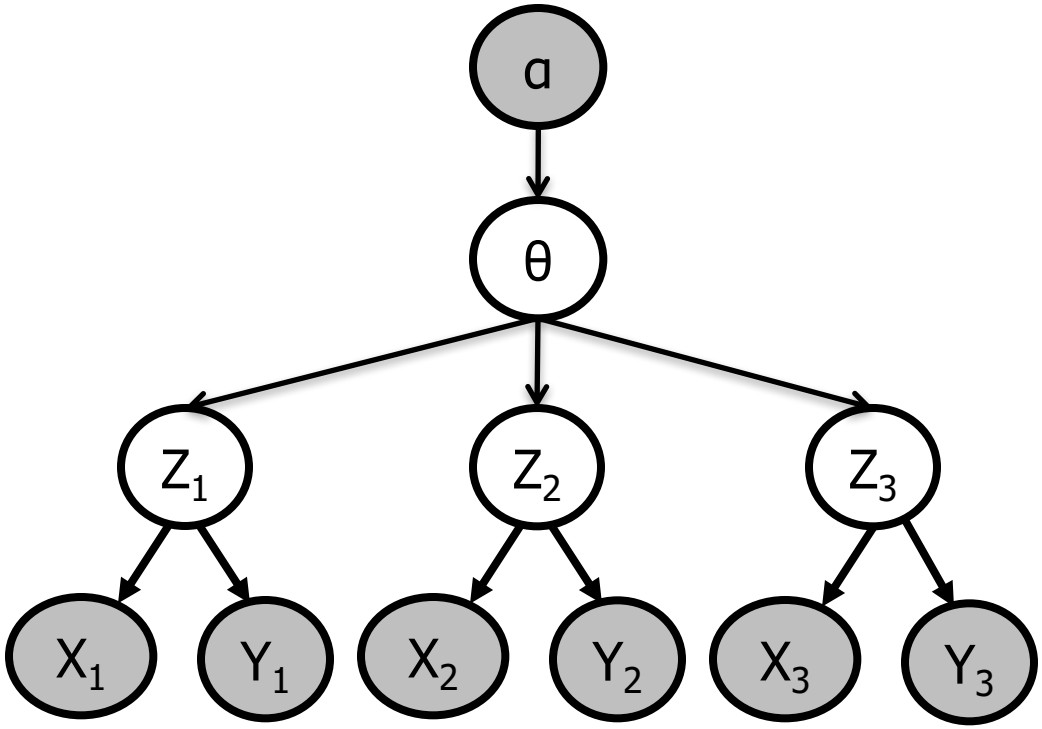
Pick $Z_3 \sim Pr(Z_3|z_1,z_2,\alpha)$

Pick $Z_1 \sim Pr(Z_1|z_2,z_3,\alpha)$

Pick $Z_2 \sim Pr(Z_2|z_1,z_3,\alpha)$

Pick $Z_3 \sim Pr(Z_3|z_1,z_2,\alpha)$

.
.
.

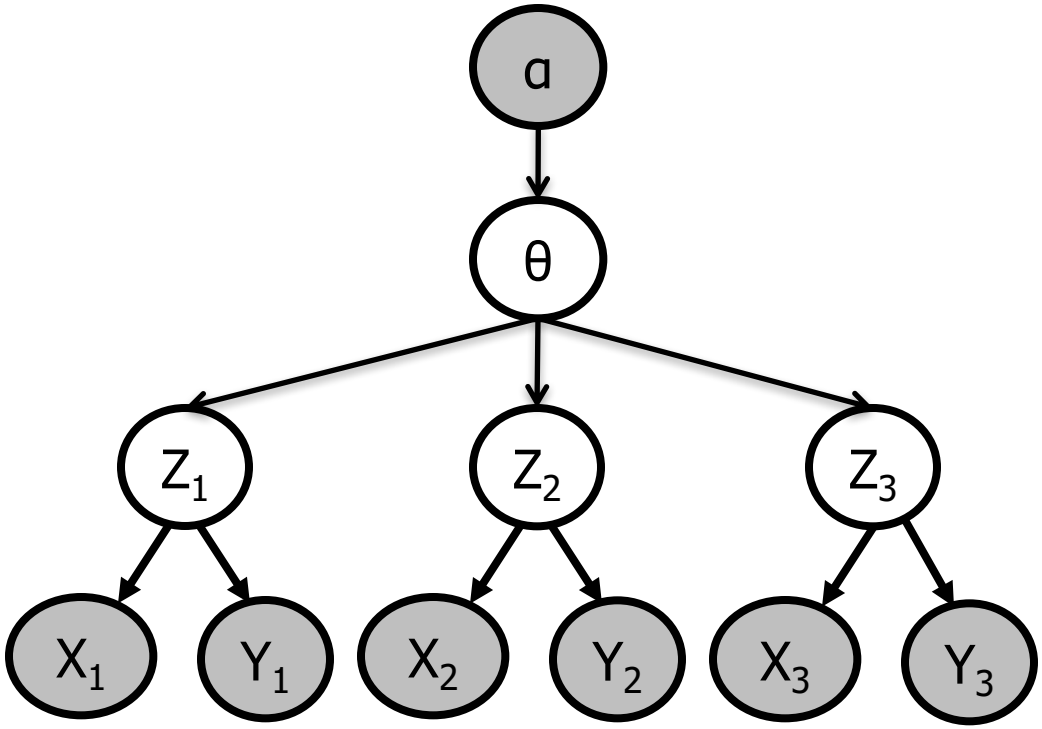


Converges to samples from the true joint … and then we can estimate Pr(θ| α, sample of Z's)

Initialize all the Z's randomly, then….

Pick $Z_1 \sim Pr(Z_1|z_2,z_3,\alpha)$

What's this distribution?

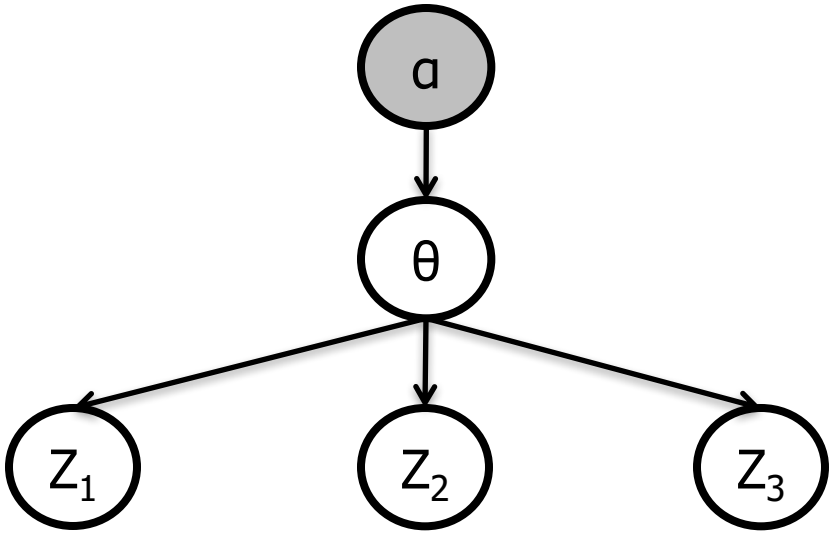


Initialize all the Z's randomly, then….

Pick $Z_1 \sim \Pr(Z_1 | z_2, z_3, \alpha)$

Simpler case: What's **this** distribution?

called a Dirichlet-multinomial and it looks like this:



$$\Pr(Z_1 = k_1, Z_2 = k_2, Z_3 = k_3 \mid \alpha) = \int_\theta \Pr(Z_1 = k_1, Z_2 = k_2, Z_3 = k_3 \mid \theta) \Pr(\theta \mid \alpha) d\theta$$

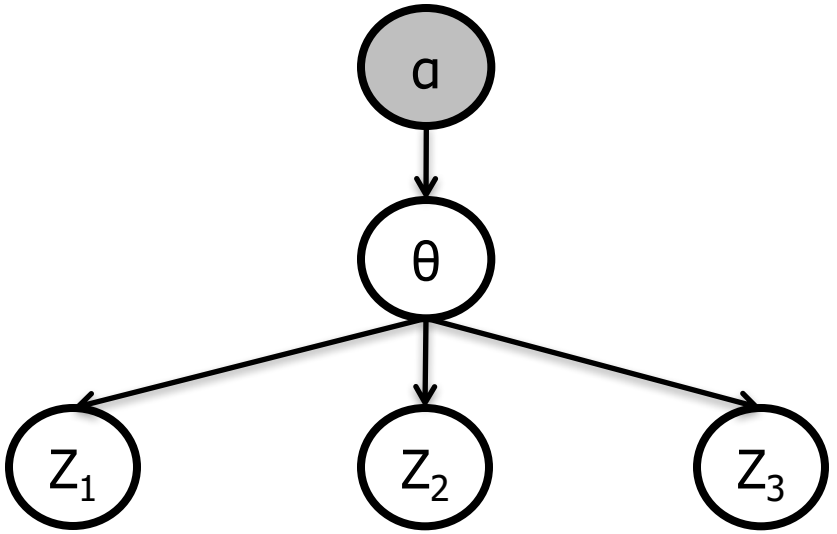
If there are k values for the Z's and n_k = # Z's with value k, then it turns out:

$$\Pr(\mathbf{Z} \mid \alpha) = \int_\theta \Pr(\mathbf{Z} \mid \theta) \Pr(\theta \mid \alpha) d\theta = \frac{\Gamma\left(\sum_k \alpha_k\right)}{\Gamma\left(\sum_k \alpha_k + \sum_k n_k\right)} \prod_k \frac{\Gamma\left(n_k + \alpha_k\right)}{\Gamma(\alpha_k)}$$

Initialize all the Z's randomly, then….

Pick $Z_1 \sim Pr(Z_1 | z_2, z_3, \alpha)$

It turns out that sampling from a Dirichlet-multinomial is very easy!



Notation:
- k values for the Z's
- n_k = # Z's with value k
- $\mathbf{Z} = (Z_1, \ldots, Z_m)$
- $\mathbf{Z}^{(-i)} = (Z_1, \ldots, Z_{i-1}, Z_{i+1}, \ldots, Z_m)$
- $n_k^{(-i)}$ = # Z's with value k excluding Z_i

$$\Pr(Z_i = k \mid \mathbf{Z}^{(-i)}, \alpha) \propto n_k^{(-i)} + \alpha_k$$

$$\Pr(\mathbf{Z} \mid \alpha) = \int_\theta \Pr(\mathbf{Z} \mid \theta) \Pr(\theta \mid \alpha) d\theta = \frac{\Gamma\left(\sum_k \alpha_k\right)}{\Gamma\left(\sum_k \alpha_k + \sum_k n_k\right)} \prod_k \frac{\Gamma\left(n_k + \alpha_k\right)}{\Gamma(\alpha_k)}$$

What about with downstream evidence?

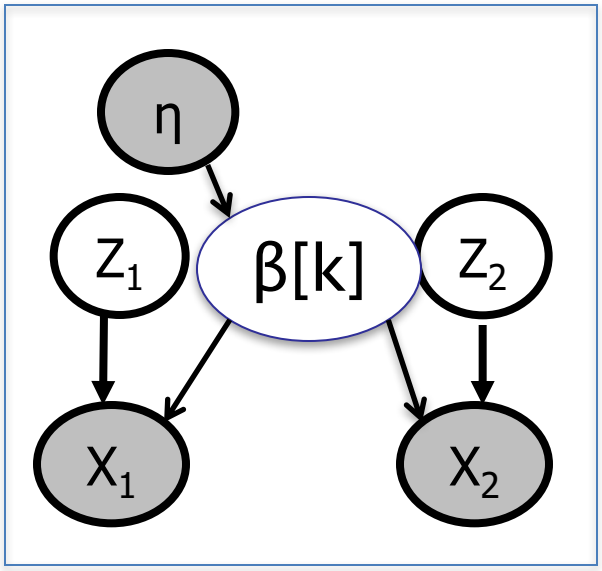
$$\Pr(Z_i = k \mid \mathbf{Z}^{(-i)}, \alpha) \quad \propto \quad n_k^{(-i)} + \alpha_k$$

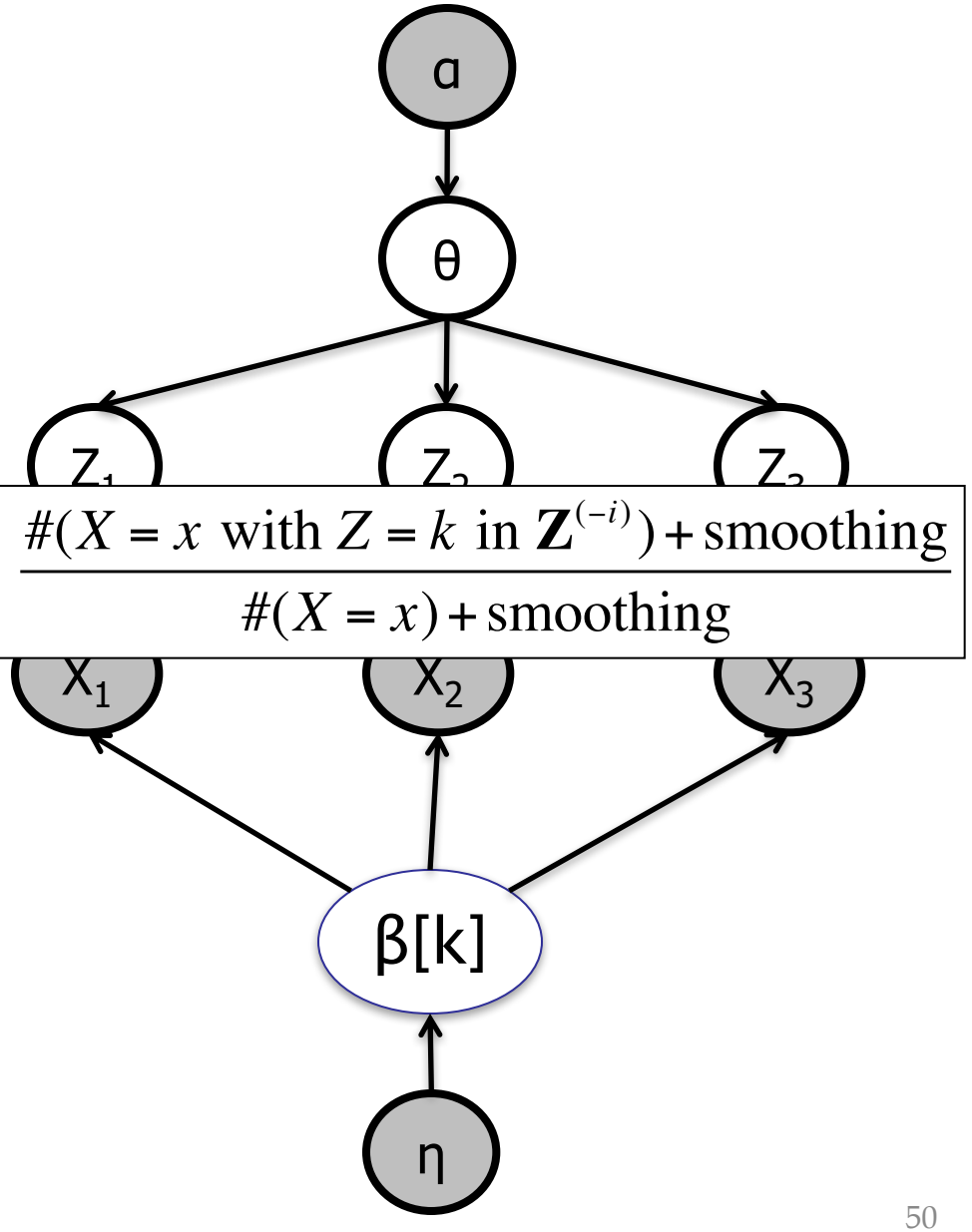
captures the constraints on Z_i via θ
(from "above", "causal" direction)

what about via β?

Pr(Z) = (1/c) * Pr(Z|E+) Pr(E−|Z)

$$\Pr(Z_i = k \mid \mathbf{Z}^{(-i)}, X_i = x, \eta) \quad \propto \quad \frac{\#(X = x \text{ with } Z = k \text{ in } \mathbf{Z}^{(-i)}) + \text{smoothing}}{\#(X = x) + \text{smoothing}}$$





Sampling for LDA

Notation:
- k values for the $Z_{d,i}$'s
- $\mathbf{Z}^{(-d,i)}$ = all the Z's but $Z_{d,i}$
- $n_{w,k}$ = # $Z_{d,i}$'s with value k paired with $W_{d,i}=w$
- $n_{*,k}$ = # $Z_{d,i}$'s with value k
- $n_{w,k}^{(-d,i)}$ = $n_{w,k}$ excluding $Z_{i,d}$
- $n_{*,k}^{(-d,i)}$ = n_{*k} excluding $Z_{i,d}$
- $n_{*,k}^{d,(-i)}$ = $n_{*k\ from}$ doc d excluding $Z_{i,d}$

$$\Pr(Z_{d,i} = k \mid \mathbf{Z}^{(-d,i)}, W_{d,i} = w, \alpha) \propto$$

Pr(Z|E+)

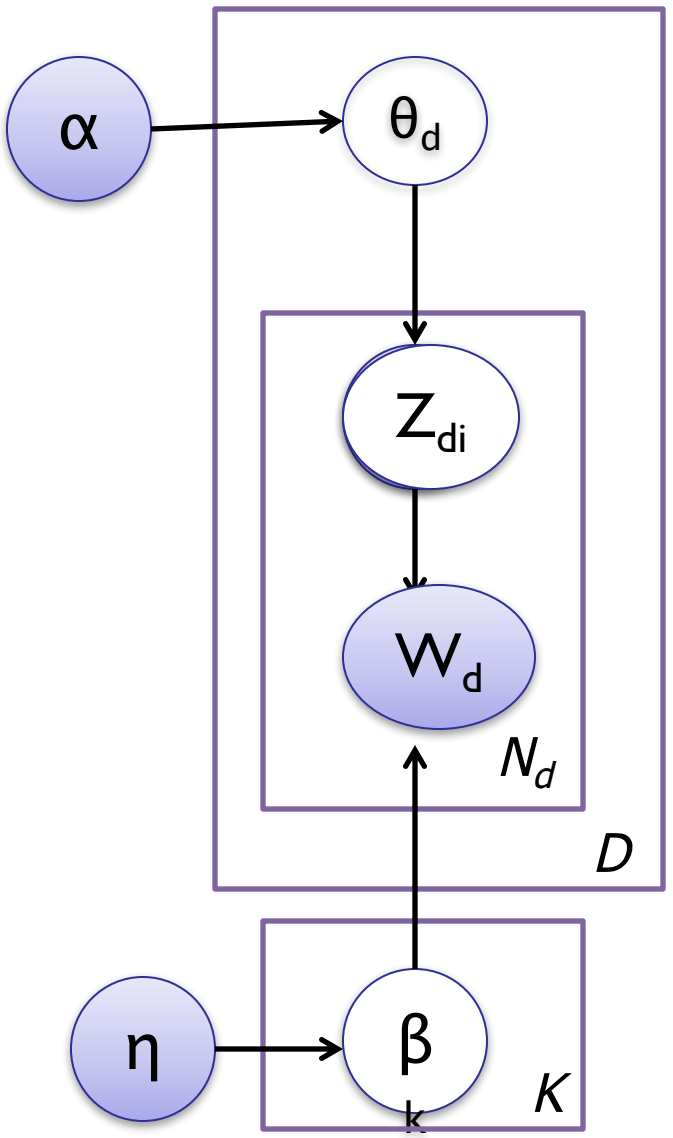
$$\left(n_{*,k}^{d,(-i)} + \alpha_k\right)$$

fraction of time $Z=k$ in doc d

Pr(E−|Z)

$$\left(\frac{n_{w,k}^{(-d,i)} + \eta_w}{\sum_{w',k} n_{w',k}^{(-d,i)} + \eta_{w'}}\right)$$

fraction of time W=w in topic k



LDA (IN TOO MUCH DETAIL)

Way way more detail

```
# topic k, docId d, and wordId w are integer indices
#
# x[d][j] = w, index of j-th word in doc d
# z[d][j] = k, index of latent topic of j-th word in doc d
# vocab[w] = string for the word with index w
#
# totalTopicCount[k] = number of words in topic k
# docTopicCount[d][k] = number of words in topic k for document d
# wordTopicCount[w][k] = number of occurrences of word w in topic k
# totalWords = number of words in the corpus
```

```
    # topic k, docId d, and wordId w are integer indices
    #
    # x[d][j] = w, index of j-th word in doc d
    # z[d][j] = k, index of latent topic of j-th word in doc d
    # vocab[w] = string for the word with index w
    #
    # totalTopicCount[k] = number of words in topic k
    # docTopicCount[d][k] = number of words in topic k for document d
    # wordTopicCount[w][k] = number of occurrences of word w in topic k
    # totalWords = number of words in the corpus


def initGibbs(self):
    print '.initializing latent vars'
    self.totalTopicCount = self.topicCounter()
    self.docTopicCount = [self.topicCounter() for d in xrange(len(self.x))]
    self.wordTopicCount = [self.topicCounter() for w in xrange(len(self.vocab))]
    self.z = [[-1 for j in xrange(len(self.x[d]))] for d in xrange(len(self.x))]
    for d in xrange(len(self.x)):
        if (d+1)%self.dstep==0: print '..doc',d+1,'of',len(self.x)
        for j in xrange(len(self.x[d])):
            w = self.x[d][j]
            k = random.randint(0,self.numTopics-1)
            self.z[d][j] = k
            self.docTopicCount[d].add(k, 1)
            self.wordTopicCount[w].add(k, 1)
            self.totalTopicCount.add(k, 1)
    #reasonable parameters
    self.alpha = 1.0/self.numTopics
    self.beta = 1.0/len(self.vocab)
    print "alpha:",self.alpha,"beta:",self.beta
```

```
def runGibbs(self,maxT):
    for t in xrange(maxT):
        print '.iteration',t+1,'of',maxT
        for d in xrange(len(self.x)):
            if (d+1)%self.dstep==0: print '..doc',d+1,'of',len(self.x)
            for j in xrange(len(self.x[d])):
                k = self.resample(d,j)
                self.flip(d, j, self.z[d][j], k)

def flip(self, d, j, k_old, k_new):
    """update counts to reflect a changed value of z[d][j]"""
    if k_old != k_new:
        w = self.x[d][j]
        self.docTopicCount[d].add(k_old, -1)
        self.docTopicCount[d].add(k_new, +1)
        self.wordTopicCount[w].add(k_old, -1)
        self.wordTopicCount[w].add(k_new, +1)
        self.totalTopicCount.add(k_old, -1)
        self.totalTopicCount.add(k_new, +1)
        self.z[d][j] = k_new
```

```
def runGibbs(self,maxT):
    for t in xrange(maxT):
        print '.iteration',t+1,'of',maxT
        for d in xrange(len(self.x)):
            if (d+1)%self.dstep==0: print '..doc',d+1,'of',len(self.x)
            for j in xrange(len(self.x[d])):
                k = self.resample(d,j)
                self.flip(d, j, self.z[d][j], k)


def resample(self,d,j):
    """sample a new value of z[d][j]"""
    p = []
    norm = 0.0
    #compute pk = Pr(z_dj=k | everything else)
    for k in xrange(self.numTopics):
        w = self.x[d][j]
        z_dj_equals_k = 1 if self.z[d][j]==k else 0
        #unnormalized chance of picking topic k in doc d
        ak = (self.docTopicCount[d][k] - z_dj_equals_k + self.alpha)
        #unnormalized chance of picking topic k for word w
        bk = ((self.wordTopicCount[w][k] - z_dj_equals_k + self.beta)
              /(self.totalTopicCount[k] - z_dj_equals_k + self.totalWords * self.beta))
        pk = ak*bk
        p.append(pk)
        norm += pk
    #pick randomly from the normalized pk
    sum_p_up_to_k = 0.0
    r = random.random()
    for k in xrange(self.numTopics):
        sum_p_up_to_k += p[k]/norm
        if r < sum_p_up_to_k:
            return k
```

What gets learned…..

```
def phi(self,w,k):
    """weight of word w under topic k"""
    num = (self.wordTopicCount[w][k] + self.beta)
    denom = (self.totalTopicCount[k] + self.totalWords * self.beta)
    return num/denom

def theta(self,d,k):
    """weight of doc unde
    num = (self.docTopicC
    denom = (sum(self.doc
    return num/denom
```

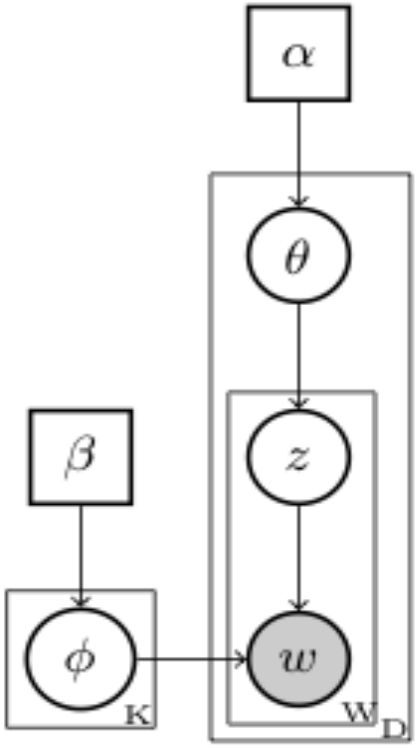


Figure 1: Graphical model for LDA.

In A Math-ier Notation

```
# topic k, docId d, and wordId w are integer indices
#
# x[d][j] = w, index of j-th word in doc d
# z[d][j] = k, index of latent topic of j-th word in doc d
# vocab[w] = string for the word with index w
#
# totalTopicCount[k] = number of words in topic k
# docTopicCount[d][k] = number of words in topic k for document d
# wordTopicCount[w][k] = number of occurrences of word w in topic k
# totalWords = number of words in the corpus
```

N[,k]*

N[d,k]

N[,*]=V*

M[w,k]

for each document *d* and word position *j* in *d*

- *z[d,j] = k,* a random topic
- *N[d,k]++*
- *W[w,k]++* where *w* = id of *j*-th word in *d*

```
def initGibbs(self):
    print '.initializing latent vars'
    self.totalTopicCount = self.topicCounter()
    self.docTopicCount = [self.topicCounter() for d in xrange(len(self.x))]
    self.wordTopicCount = [self.topicCounter() for w in xrange(len(self.vocab))]
    self.z = [[-1 for j in xrange(len(self.x[d]))] for d in xrange(len(self.x))]
    for d in xrange(len(self.x)):
        if (d+1)%self.dstep==0: print '..doc',d+1,'of',len(self.x)
        for j in xrange(len(self.x[d])):
            w = self.x[d][j]
            k = random.randint(0,self.numTopics-1)
            self.z[d][j] = k
            self.docTopicCount[d].add(k, 1)
            self.wordTopicCount[w].add(k, 1)
            self.totalTopicCount.add(k, 1)
    #reasonable parameters
    self.alpha = 1.0/self.numTopics
    self.beta = 1.0/len(self.vocab)
    print "alpha:",self.alpha,"beta:",self.beta
```

```
def runGibbs(self,maxT):
    for t in xrange(maxT):
        print '.iteration',t+1,'of',maxT
        for d in xrange(len(self.x)):
            if (d+1)%self.dstep==0: print '..doc',d+1,'of',len(self.x)
            for j in xrange(len(self.x[d])):
                k = self.resample(d,j)
                self.flip(d, j, self.z[d][j], k)
```

for each pass t=1,2,….

for each document *d* and word position *j* in *d*

- *z[d,j] = k*, a new random topic
- update *N,W* to reflect the new assignment of z:
  - *N[d,k]++; N[d,k']* - - where *k'* is old *z[d,j]*
  - *W[w,k]++;W[w,k']* - - where *w* is *w[d,j]*

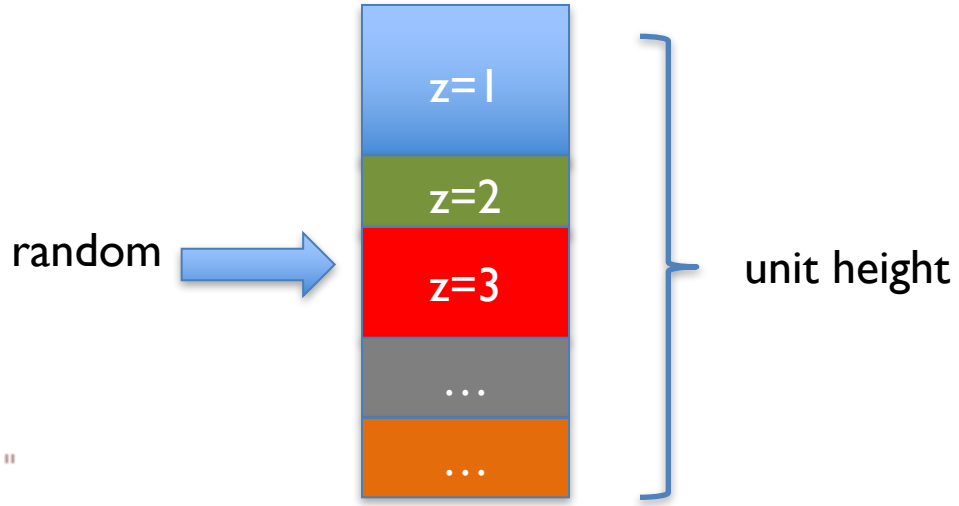
```
def flip(self, d, j, k_old, k_new):
    """update counts to reflect a changed value of z[d][j]"""
    if k_old != k_new:
        w = self.x[d][j]
        self.docTopicCount[d].add(k_old, -1)
        self.wordTopicCount[w].add(k_old, -1)
        self.wordTopicCount[w].add(k_new, +1)
        self.totalTopicCount.add(k_old, -1)
        self.totalTopicCount.add(k_new, +1)
        self.z[d][j] = k_new
```

$$p(Z_{d,j}=k\,|\,..) \propto \Pr(Z_{d,j}=k\,|\,"d") * \Pr(W_{d,k}=w\,|\,Z_{d,j}=k,...)$$

$$= \frac{N[k,d]-C_{d,j,k}+\alpha}{Z} \cdot \frac{W[w,k]-C_{d,j,k}+\beta}{(W[*,k]-C_{d,j,k})+\beta N[*,*]}$$

$$C_{d,j,k} = \begin{cases} 1 & Z_{d,j}=k \\ 0 & \text{else} \end{cases}$$

```
def resample(self,d,j):
    """sample a new value of z[d][j]"""
    p = []
    norm = 0.0
    #compute pk = Pr(z_dj=k | everything else)
    for k in xrange(self.numTopics):
        w = self.x[d][j]
        z_dj_equals_k = 1 if self.z[d][j]==k else 0
        #unnormalized chance of picking topic k in doc d
        ak = (self.docTopicCount[d][k] - z_dj_equals_k + self.alpha)
        #unnormalized chance of picking topic k for word w
        bk = ((self.wordTopicCount[w][k] - z_dj_equals_k + self.beta)
              /(self.totalTopicCount[k] - z_dj_equals_k + self.totalWords * self.beta))
        pk = ak*bk
        p.append(pk)
        norm += pk
    #pick randomly from the normalized pk
    sum_p_up_to_k = 0.0
    r = random.random()
    for k in xrange(self.numTopics):
        sum_p_up_to_k += p[k]/norm
        if r < sum_p_up_to_k:
            return k
```




```
def resample(self,d,j):
    """sample a new value of z[d][j]"""
    p = []
    norm = 0.0
    #compute pk = Pr(z_dj=k | everything else)
    for k in xrange(self.numTopics):
        w = self.x[d][j]
        z_dj_equals_k = 1 if self.z[d][j]==k else 0
        #unnormalized chance of picking topic k in doc d
        ak = (self.docTopicCount[d][k] - z_dj_equals_k + self.alpha)
        #unnormalized chance of picking topic k for word w
        bk = ((self.wordTopicCount[w][k] - z_dj_equals_k + self.beta)
              /(self.totalTopicCount[k] - z_dj_equals_k + self.totalWords * self.beta))
        pk = ak*bk
        p.append(pk)
        norm += pk
    #pick randomly from the normalized pk
    sum_p_up_to_k = 0.0
    r = random.random()
    for k in xrange(self.numTopics):
        sum_p_up_to_k += p[k]/norm
        if r < sum_p_up_to_k:
            return k
```

1. You spend a *lot* of time sampling
2. There's a loop over all topics here in the sampler