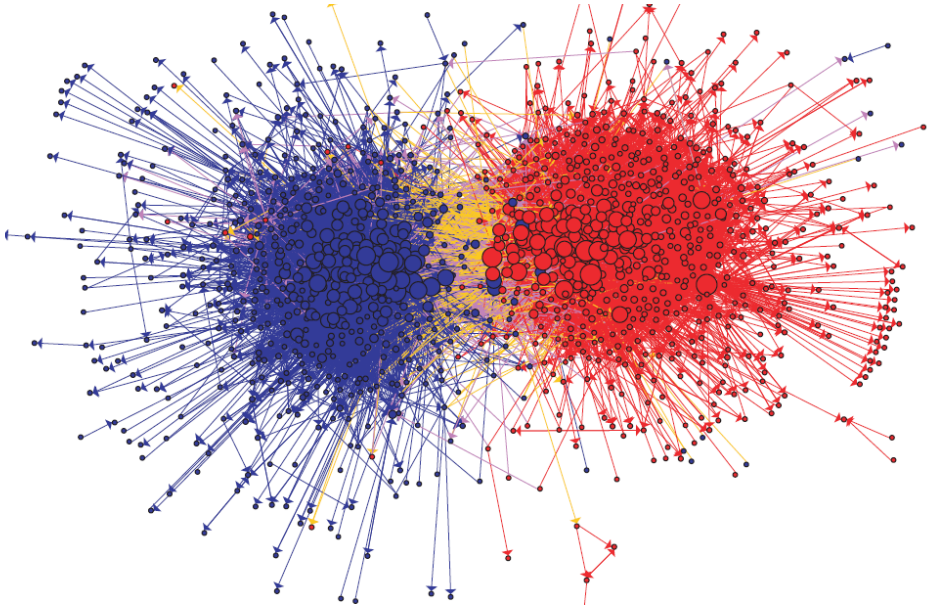
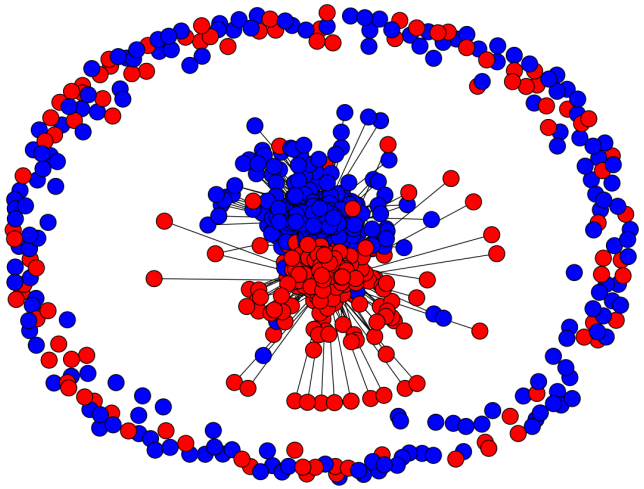
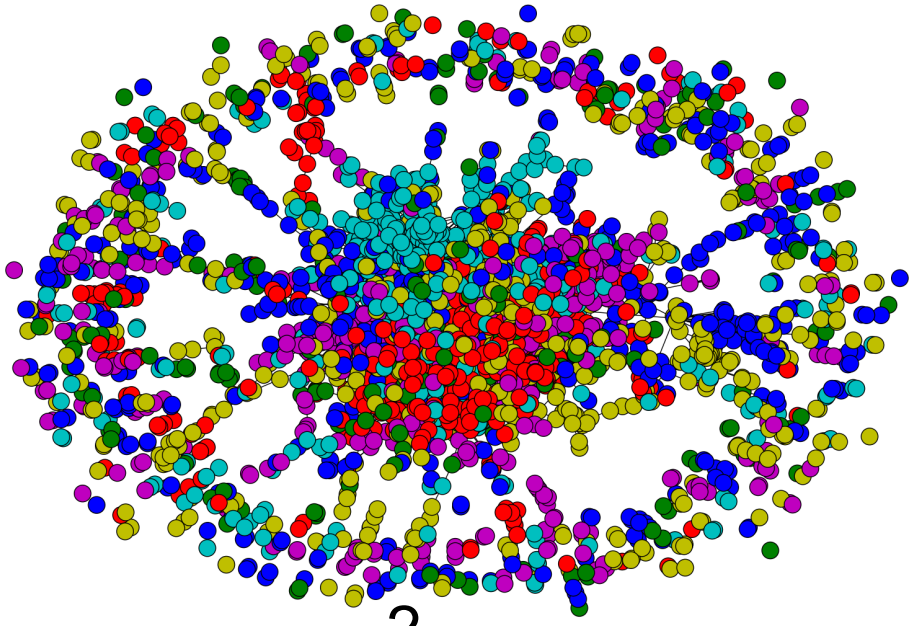
# Topic models for corpora and for graphs

# Network Datasets



- UBMCBlog
- AGBlog
- MSPBlog
- Cora
- Citeseer





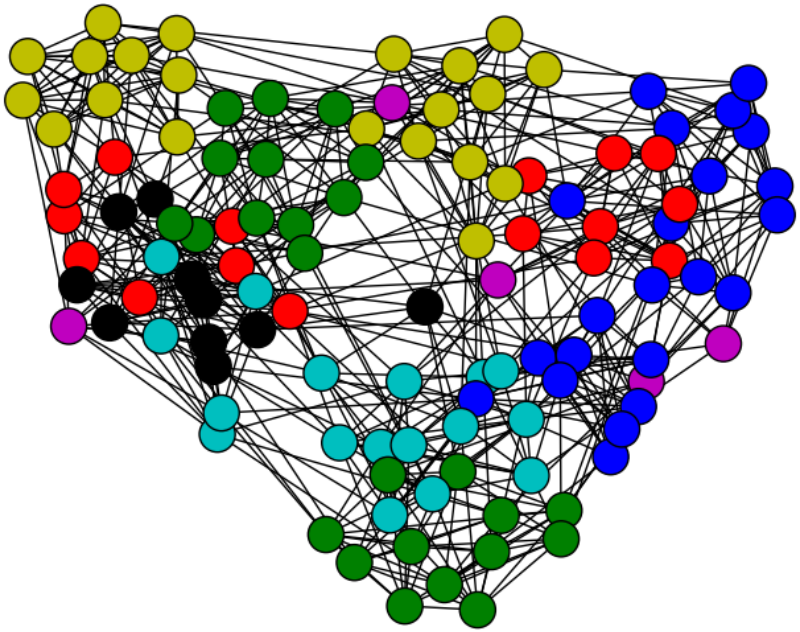
# Motivation

- Social graphs seem to have
  - some aspects of randomness
    - small diameter, giant connected components,..
  - some structure
    - homophily, scale-free degree dist?
- How do you model it?

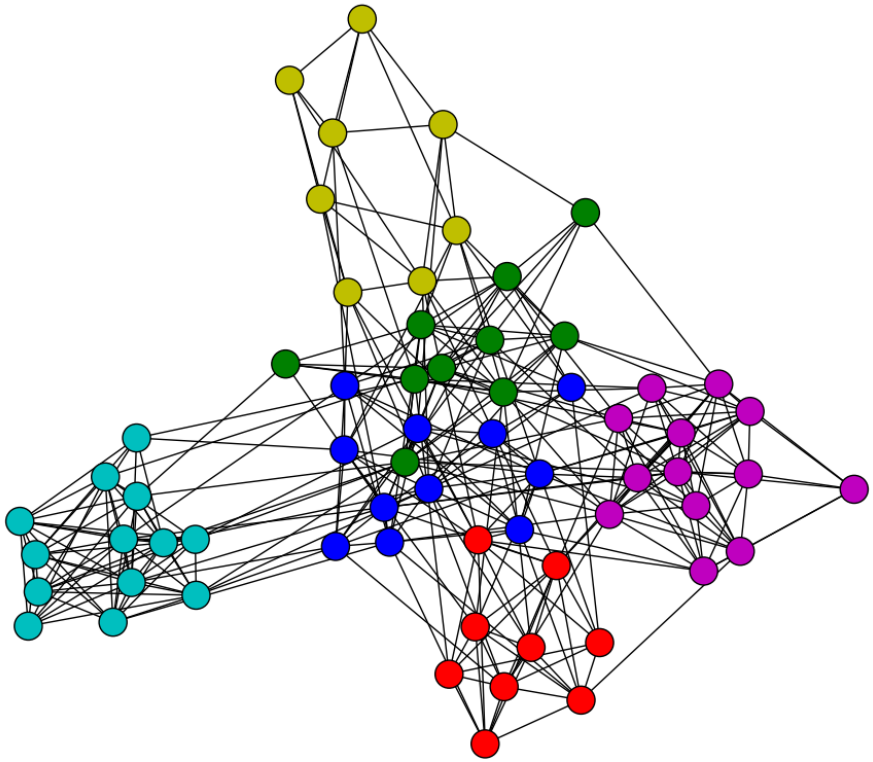
# More terms

- "Stochastic block model", aka "Block-stochastic matrix":
  - Draw $n_i$ nodes in block $i$
  - With probability $p_{ij}$, connect pairs $(u,v)$ where $u$ is in block $i$, $v$ is in block $j$
  - Special, simple case: $p_{ii}=q_i$, and $pij=s$ for all $i\neq j$

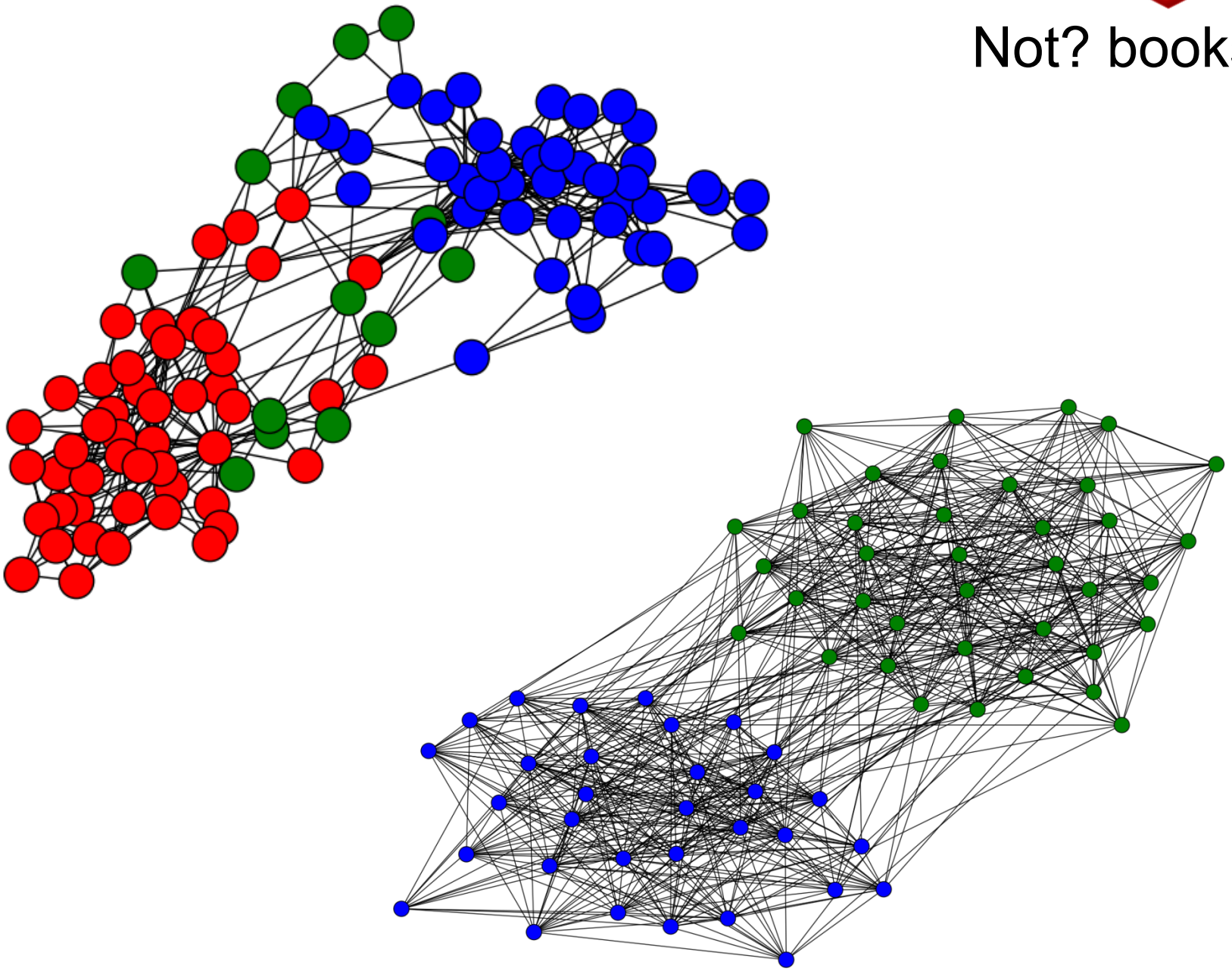
- Question: can you fit this model to a graph?
  - find each $p_{ij}$ and latent node→block mapping



Not? football



Not? books

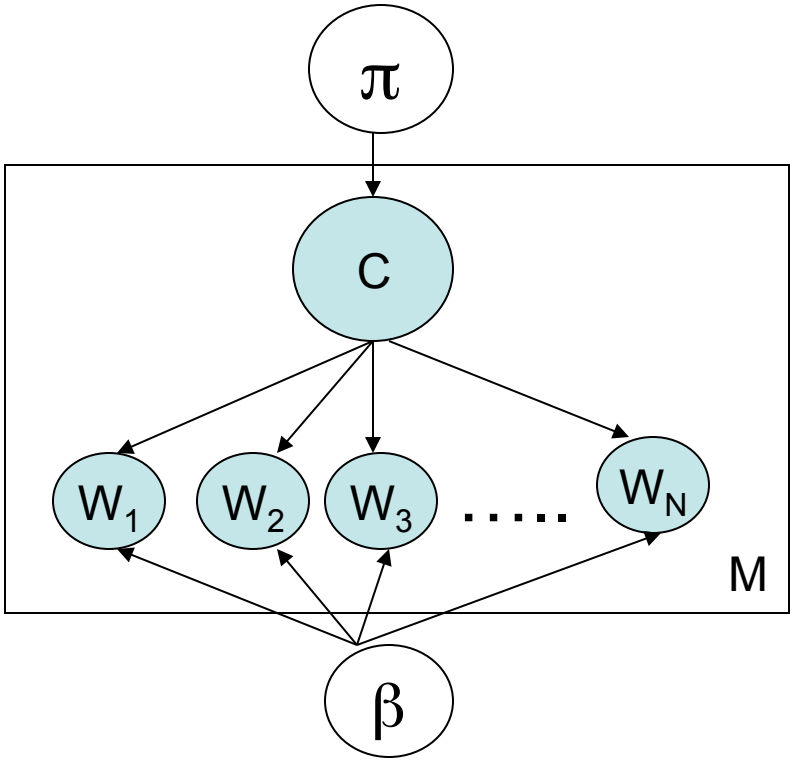


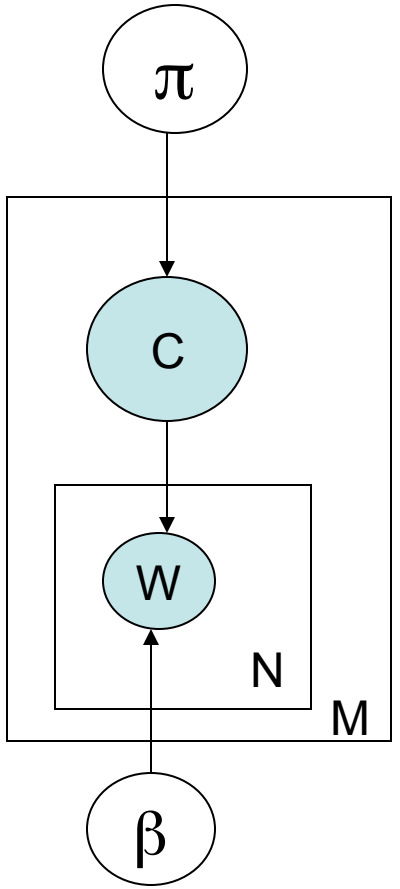
# Outline

- Stochastic block models & inference question
- Review of text models
  - Mixture of multinomials & EM
  - LDA and Gibbs (or variational EM)
- Block models and inference
- Mixed-membership block models
- Multinomial block models and inference w/ Gibbs

# Review – supervised Naïve Bayes

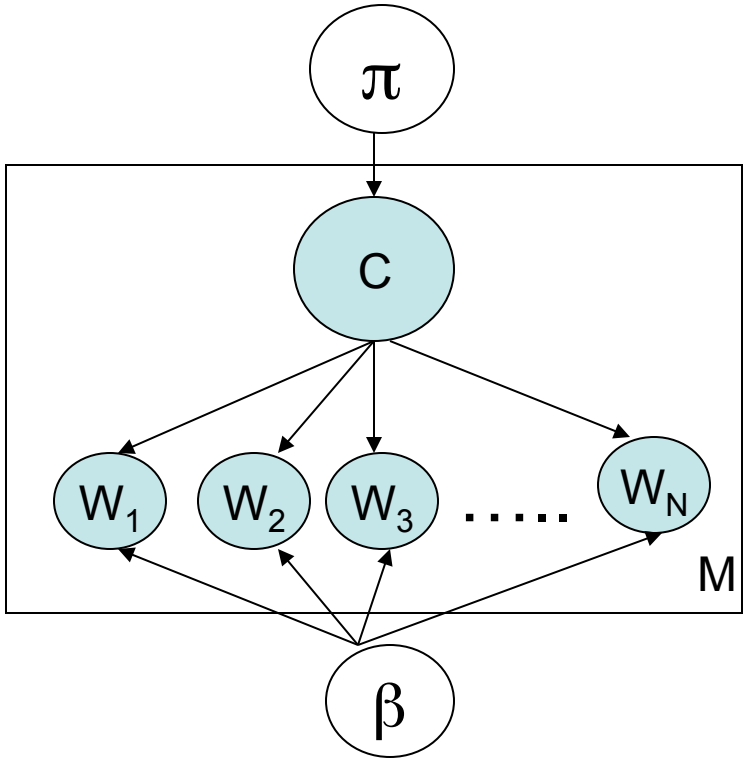
- Naïve Bayes Model: Compact representation



# Review – supervised Naïve Bayes

- Multinomial Naïve Bayes




- For each document d = 1,⋯, M
  - Generate $C_d$ ~ Mult( | π)
  - For each position n = 1,⋯, $N_d$
    - Generate $w_n$ ~ Mult( |β,$C_d$)

# Review – supervised Naïve Bayes

- Multinomial naïve Bayes: Learning
  - Maximize the log-likelihood of observed variables w.r.t. the parameters:

$$\sum_{d=1}^{M} \log P(w_1, \cdots, w_{N_d}, C_d | \beta, \pi) = \sum_{d=1}^{M} \left\{ \log(\pi_{C_d}) + \sum_{n=1}^{N_d} \log(\beta_{C_d, w_n}) \right\}$$

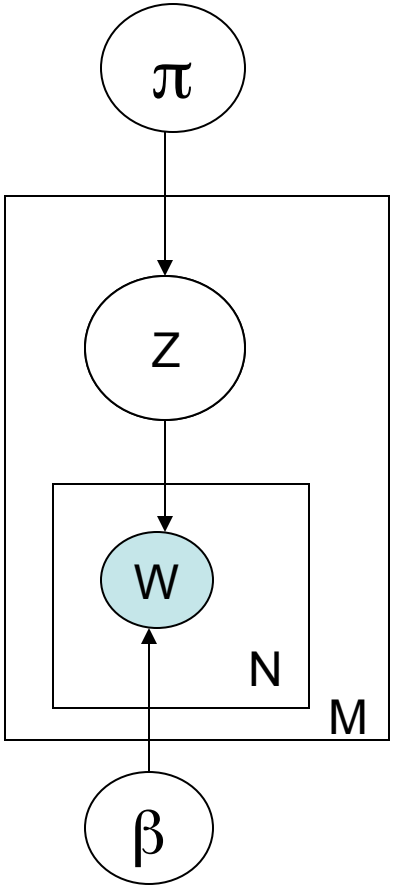
- Convex function: global optimum
- Solution:

$$\pi_C = \frac{\sum_{d=1}^{N} \delta_C(C_d)}{M}$$

$$\beta_{C,w} = \frac{\sum_{d:C_d=C} n(d,w)}{\sum_{d:C_d=C} \sum_w n(d,w)}$$

# Review – unsupervised Naïve Bayes

- Mixture model: unsupervised naïve Bayes model




- Joint probability of words and classes:

$$\prod_{d=1}^{M} P(w_1, \cdots, w_{N_d}, z_d | \beta, \pi) = \prod_{d=1}^{M} \left\{ \pi_{z_d} \prod_{n=1}^{N_d} \beta_{z_d, w_n} \right\}$$

- But classes are not visible:

$$\prod_{d=1}^{M} P(w_1, \cdots, w_{N_d} | \pi, \beta) = \prod_{d=1}^{N_d} \left\{ \sum_{k=1}^{K} \left( \pi_k \prod_{n=1}^{N_d} \beta_{k, w_n} \right) \right\}$$

# LDA

## Latent Dirichlet Allocation

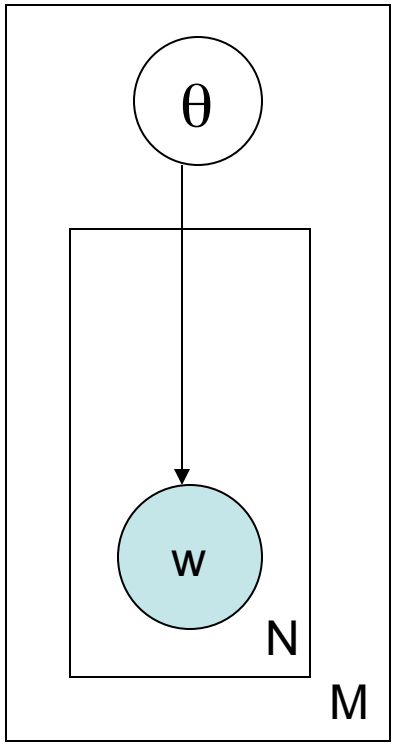
**David M. Blei**
*Computer Science Division*
*University of California*
*Berkeley, CA 94720, USA*

**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*

# Review - LDA

- Motivation




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

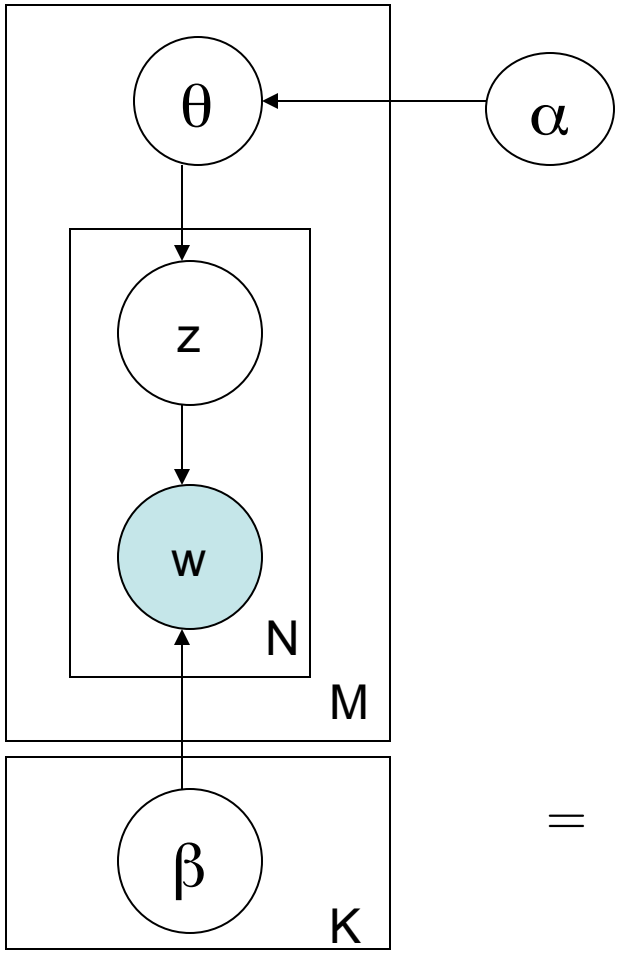
- Generate $\theta_d \sim D_1(\ldots)$
- For each word n = 1,⋯, $N_d$
  - •generate $w_n \sim D_2(\ \ | \vartheta_{d_n})$

Now pick your favorite distributions for $D_1$, $D_2$

$$\Pr(z = j \mid n_1, n_2, \ldots, n_k, \alpha) = \frac{n_j + \alpha_j}{n_1 + \alpha_1 + \ldots n_k + \alpha_k}$$

"Mixed membership"

- Latent Dirichlet Allocation

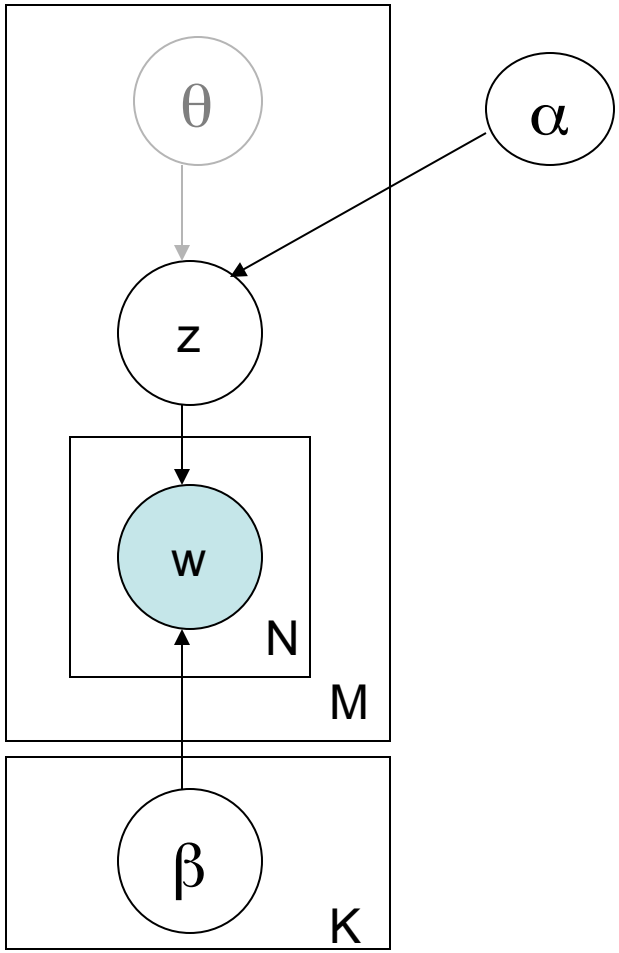



- For each document d = 1,⋯,M
  - Generate $\theta_d$ ~ Dir( | $\alpha$)
  - For each position n = 1,⋯, $N_d$
    - generate $z_n$ ~ Mult( | $\theta_d$)
    - generate $w_n$ ~ Mult( | $\beta_{z_n}$)

$$\prod_{d=1}^{N_d} P(w_1, \cdots, w_{N_d} | \beta, \alpha)$$

$$= \prod_{d=1}^{N_d} \int_{\theta_d} P(\theta_d | \alpha) \left\{ \prod_{n=1}^{N_d} \left( \sum_k \theta_{dk} \beta_{k w_n} \right) \right\} d\theta_d$$

- vs Naïve Bayes…



- LDA's view of a document

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

| "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 |

# Review - LDA

- Latent Dirichlet Allocation
  - Parameter learning:
    - Variational EM
      - Numerical approximation using lower-bounds
      - Results in biased solutions
      - Convergence has numerical guarantees
    - Gibbs Sampling
      - Stochastic simulation
      - unbiased solutions
      - Stochastic convergence

# Review - LDA

- Gibbs sampling
  - 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.

# Why does Gibbs sampling work?

- What's the fixed point?
  - Stationary distribution of the chain is the joint distribution
- When will it converge (in the limit)?
  - Graph defined by the chain is connected
- How long will it take to converge?
  - Depends on second eigenvector of that graph

⊡ initialisation
zero all count variables, $n_m^{(k)}, n_m, n_k^{(t)}, n_k$
**for** all documents $m \in [1, M]$ **do**
  **for** all words $n \in [1, N_m]$ in document $m$ **do**
    sample topic index $z_{m,n}=k \sim \text{Mult}(1/K)$
    increment document–topic count: $n_m^{(k)} + 1$
    increment document–topic sum: $n_m + 1$
    increment topic–term count: $n_k^{(t)} + 1$
    increment topic–term sum: $n_k + 1$
  **end for**
**end for**
⊡ Gibbs sampling over burn-in period and sampling period
**while** not finished **do**
  **for** all documents $m \in [1, M]$ **do**
    **for** all words $n \in [1, N_m]$ in document $m$ **do**
      ⊡ for the current assignment of $k$ to a term $t$ for word $w_{m,n}$:
      decrement counts and sums: $n_m^{(k)} - 1; n_m - 1; n_k^{(t)} - 1; n_k - 1$
      sample topic index $\tilde{k} \sim p(z_i|\vec{z}_{\neg i}, \vec{w})$ ep):
      ⊡ use the new assignment of $z_{m,n}$ to the term $t$ for word $w_{m,n}$ to:
      increment counts and sums: $n_m^{(\tilde{k})} + 1; n_m + 1; n_{\tilde{k}}^{(t)} + 1; n_{\tilde{k}} + 1$
    **end for**
  **end for**
  ⊡ check convergence and read out parameters

$$p(z_i{=}k|\vec{z}_{\neg i},\vec{w}) = \frac{p(\vec{w},\vec{z})}{p(\vec{w},\vec{z}_{\neg i})} = \frac{p(\vec{w}|\vec{z})}{p(\vec{w}_{\neg i}|\vec{z}_{\neg i})p(w_i)} \cdot \frac{p(\vec{z})}{p(\vec{z}_{\neg i})}$$

$$\propto \frac{\Delta(\vec{n}_z+\vec{\beta})}{\Delta(\vec{n}_{z,\neg i}+\vec{\beta})} \cdot \frac{\Delta(\vec{n}_m+\vec{\alpha})}{\Delta(\vec{n}_{m,\neg i}+\vec{\alpha})}$$

$$\propto \frac{\Gamma(n_k^{(t)}+\beta_t)\,\Gamma(\sum_{t=1}^{V} n_{k,\neg i}^{(t)}+\beta_t)}{\Gamma(n_{k,\neg i}^{(t)}+\beta_t)\,\Gamma(\sum_{t=1}^{V} n_k^{(t)}+\beta_t)} \cdot \frac{\Gamma(n_m^{(k)}+\alpha_k)\,\Gamma(\sum_{k=1}^{K} n_{m,\neg i}^{(k)}+\alpha_k)}{\Gamma(n_{m,\neg i}^{(k)}+\alpha_k)\,\Gamma(\sum_{k=1}^{K} n_m^{(k)}+\alpha_k)}$$

$$\propto \frac{n_{k,\neg i}^{(t)}+\beta_t}{\sum_{t=1}^{V} n_{k,\neg i}^{(t)}+\beta_t} \cdot \frac{n_{m,\neg i}^{(t)}+\alpha_k}{[\sum_{k=1}^{K} n_m^{(k)}+\alpha_k]-1}$$

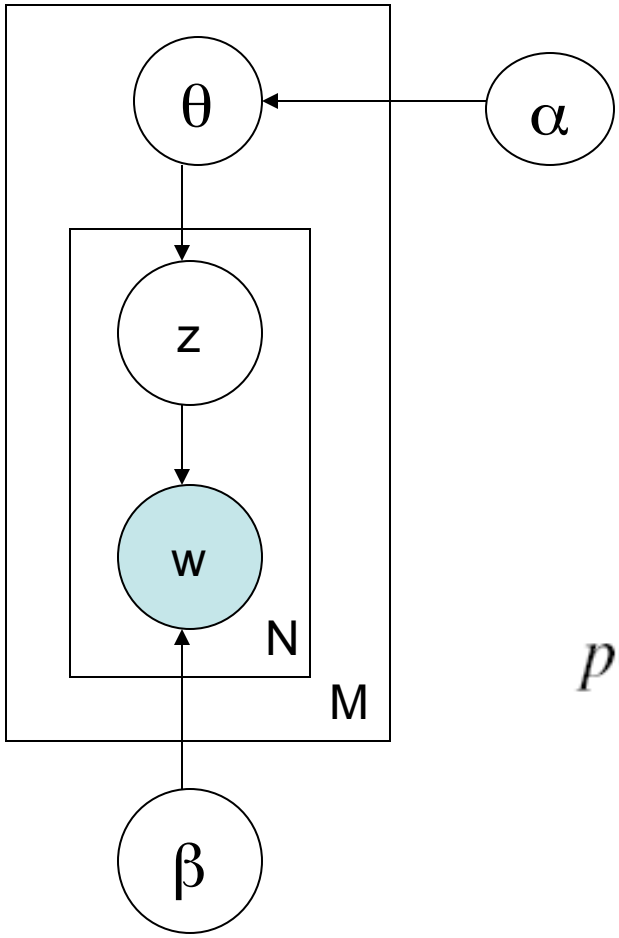
Called “collapsed Gibbs sampling” since you’ve marginalized away some variables

Fr*: Parameter estimation for text analysis* - Gregor Heinrich

# Review - LDA

"Mixed membership"

- Latent Dirichlet Allocation




- Randomly initialize each $z_{m,n}$
- Repeat for t=1,….
  - For each doc *m,* word *n*
    - Find Pr($z_{mn}$=*k*|other z's)
    - Sample $z_{mn}$ according to that distr.

$$p(z_i=k|\vec{z}_{\neg i},\vec{w}) =$$

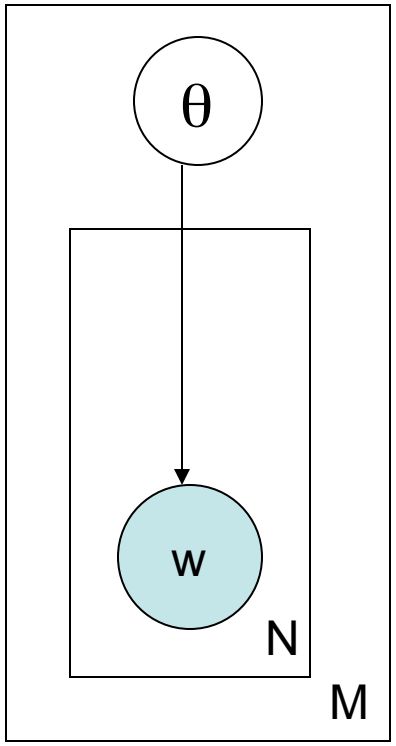
$$\propto \frac{n^{(t)}_{k,\neg i}+\beta_t}{\sum_{t=1}^{V} n^{(t)}_{k,\neg i}+\beta_t} \cdot \frac{n^{(t)}_{m,\neg i}+\alpha_k}{[\sum_{k=1}^{K} n^{(k)}_{m}+\alpha_k]-1}$$

# Outline

- Stochastic block models & inference question
- Review of text models
  - Mixture of multinomials & EM
  - LDA and Gibbs (or variational EM)
- **Block models and inference**
- Mixed-membership block models
- Multinomial block models and inference w/ Gibbs
- Beastiary of other probabilistic graph models
  - Latent-space models, exchangeable graphs, p1, ERGM

# Review - LDA

- Motivation




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

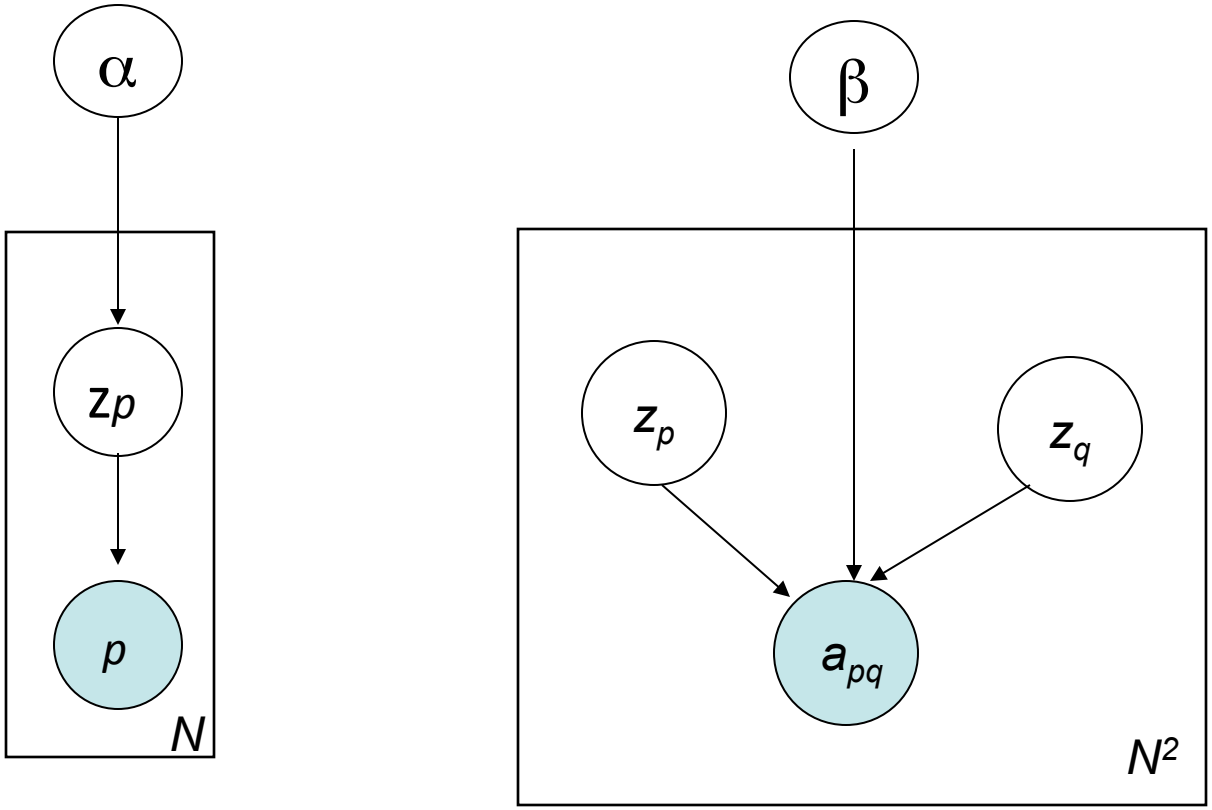
•For each document d = 1,⋯,M

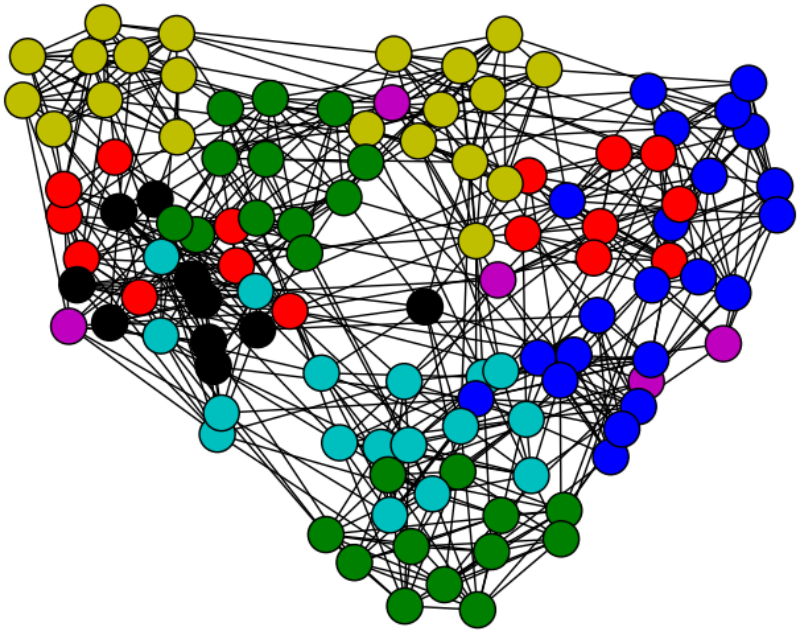
- Generate $\theta_d \sim D_1(\ldots)$
- For each word n = 1,⋯, $N_d$
  - •generate $w_n \sim D_2(\ \ | \vartheta_{d_n})$

Docs and words are *exchangeable*.

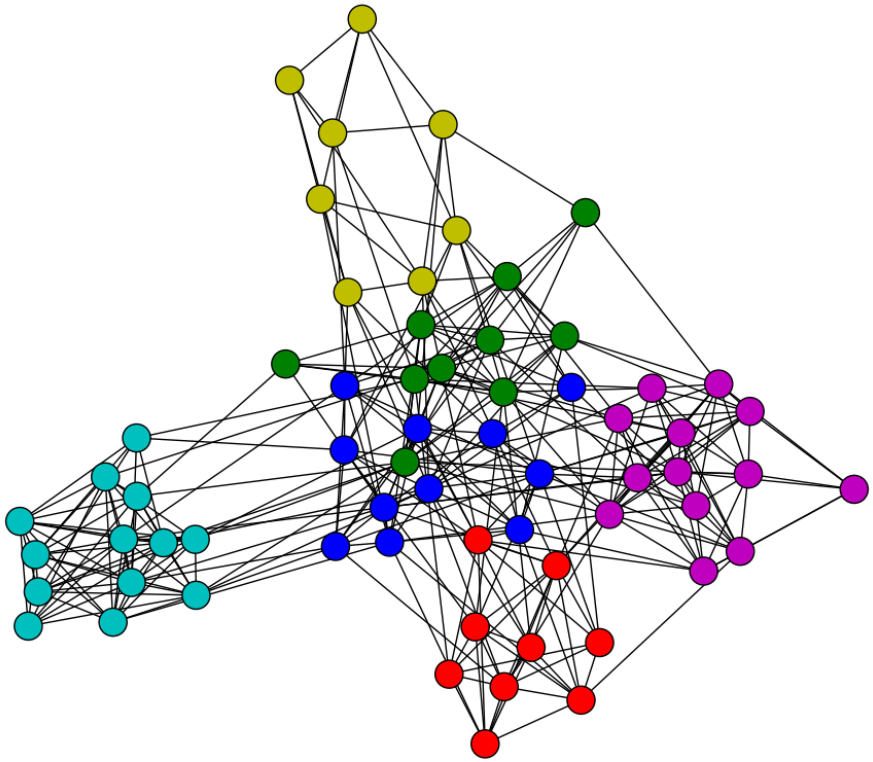
# Stochastic Block models:

assume 1) nodes w/in a block $z$ and
2) edges between blocks $z_p$,$z_q$ are *exchangeable*

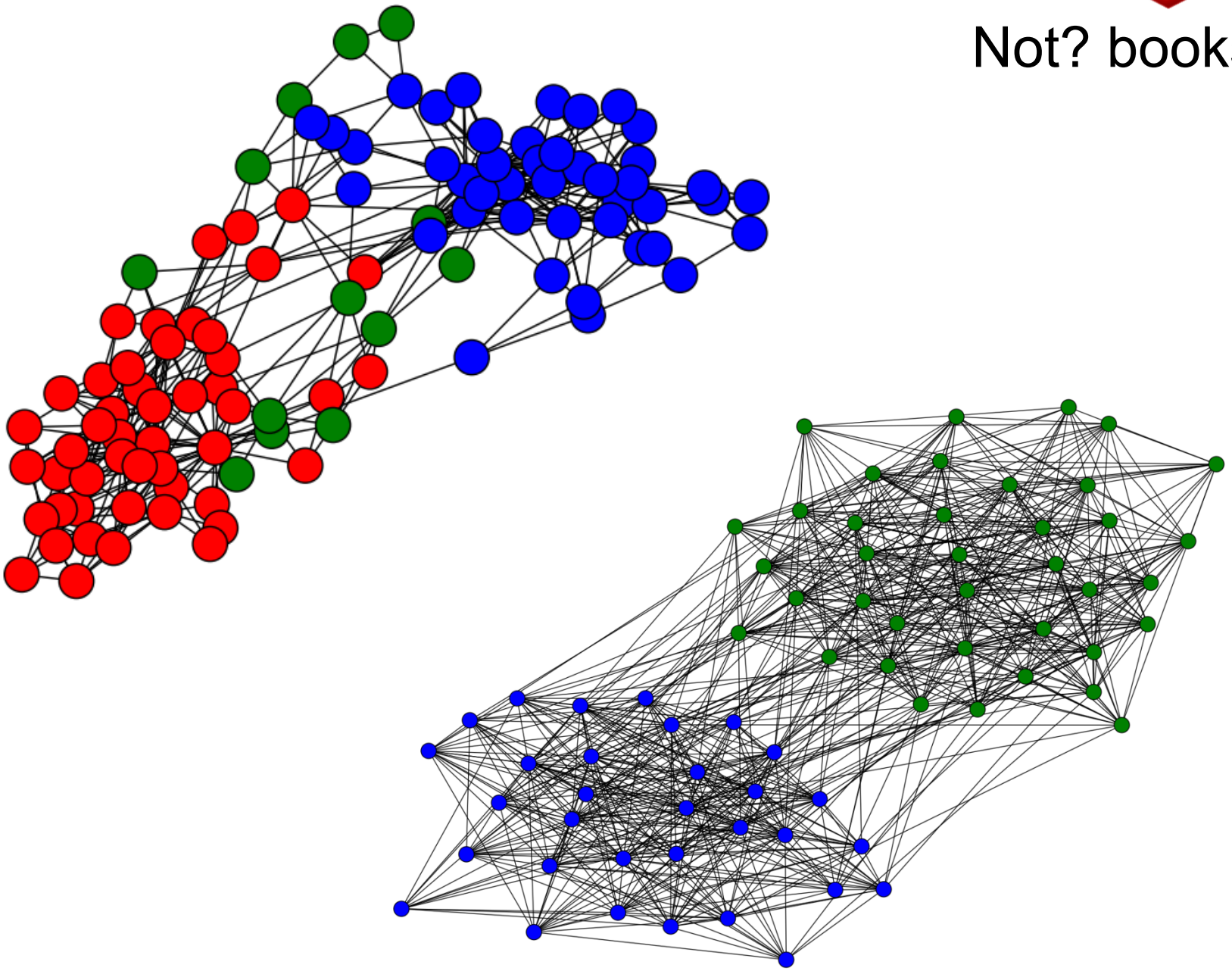




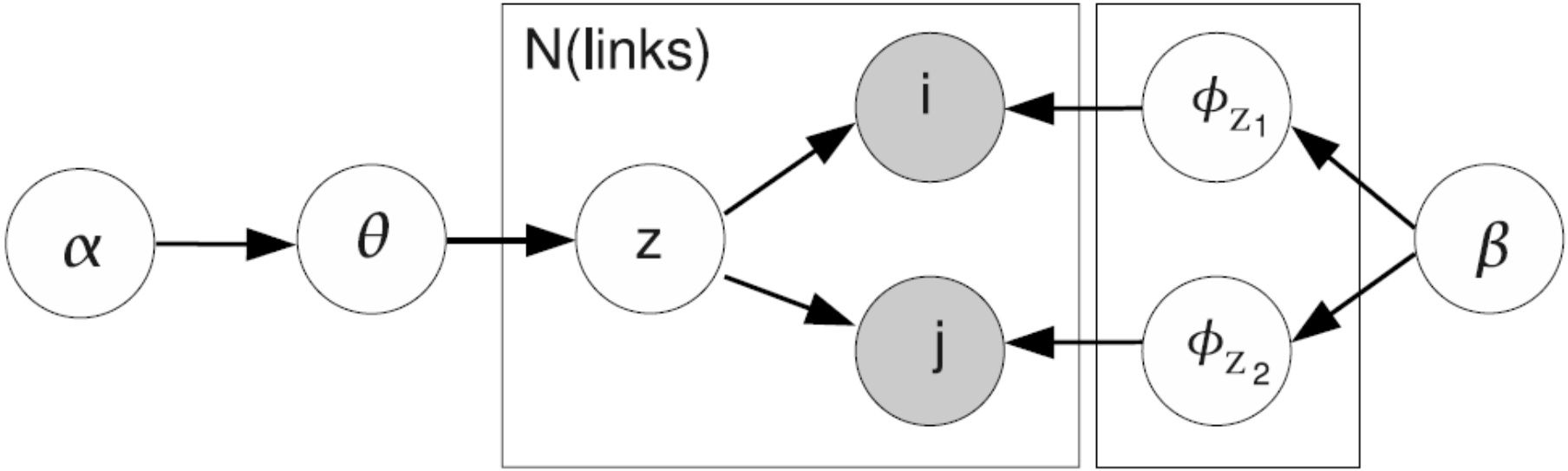
Not? football



Not? books



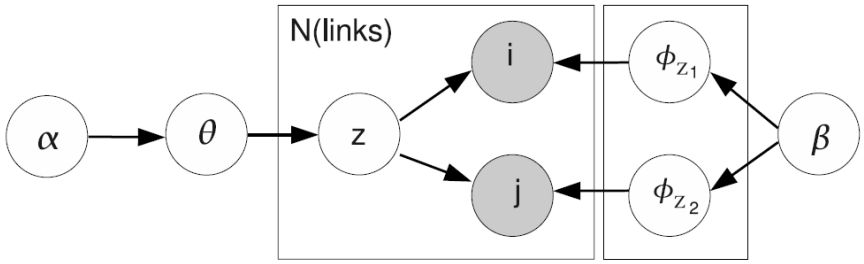
# Another mixed membership block model




$$p(z_l|\{z\}^{\neg l}, \{(i,j)\}^{\neg l}, \alpha, \beta) \propto$$

$$(n_z^{\neg l} + \alpha) \cdot \frac{(q_{z_1 i}^{\neg l} + \beta)(q_{z_2 j}^{\neg l} + \beta)}{(q_{z_1 \cdot}^{\neg l} + M\beta)(q_{z_2 \cdot}^{\neg l} + M\beta + \delta_z)},$$

# Another mixed membership block model




*z=(zi,zj)* is a pair of block ids

$n_z$ = #pairs *z*

$q_{z1,i}$ = #links to *i* from block z1

$q_{z1,.}$ = #outlinks in block z1

δ = indicator for diagonal

M = #nodes

$$p(z_l|\{z\}^{\neg l},\{(i,j)\}^{\neg l},\alpha,\beta) \propto$$

$$(n_z^{\neg l}+\alpha)\cdot\frac{(q_{z_1 i}^{\neg l}+\beta)(q_{z_2 j}^{\neg l}+\beta)}{(q_{z_1 .}^{\neg l}+M\beta)(q_{z_2 .}^{\neg l}+M\beta+\delta_z)},$$

# Experiments

(e) 1-NN: Social networks

| Dataset | PSK | $PIC_D$ | $PIC_R$ | $PIC_{R4}$ | NCut | NJW |
|---|---|---|---|---|---|---|
| Karate | **1.00** | **1.00** | 0.99 | 0.99 | 1.00 | 0.97 |
| Dolphin | 0.89 | 0.95 | 0.95 | 0.95 | 0.95 | **0.98** |
| UMBC | 0.92 | 0.93 | 0.93 | 0.93 | 0.92 | **0.94** |
| AG | 0.92 | **0.94** | 0.93 | 0.93 | 0.88 | 0.89 |
| MSP | 0.84 | 0.76 | 0.73 | **0.86** | 0.64 | 0.59 |
| Senate | 0.97 | 1.00 | **1.00** | **1.00** | **1.00** | **1.00** |
| PolBook | 0.79 | 0.68 | 0.76 | 0.80 | **0.84** | 0.78 |
| Football | 0.89 | 0.43 | 0.45 | 0.85 | 0.94 | **0.95** |
| MGEmail | 0.22 | 0.27 | 0.26 | 0.72 | 0.80 | **0.81** |
| CiteSeer | 0.34 | 0.55 | 0.54 | **0.71** | 0.69 | 0.66 |
| Cora | 0.45 | 0.56 | 0.51 | **0.80** | 0.47 | 0.75 |
| **Average** | 0.75 | 0.73 | 0.73 | **0.87** | 0.83 | 0.85 |

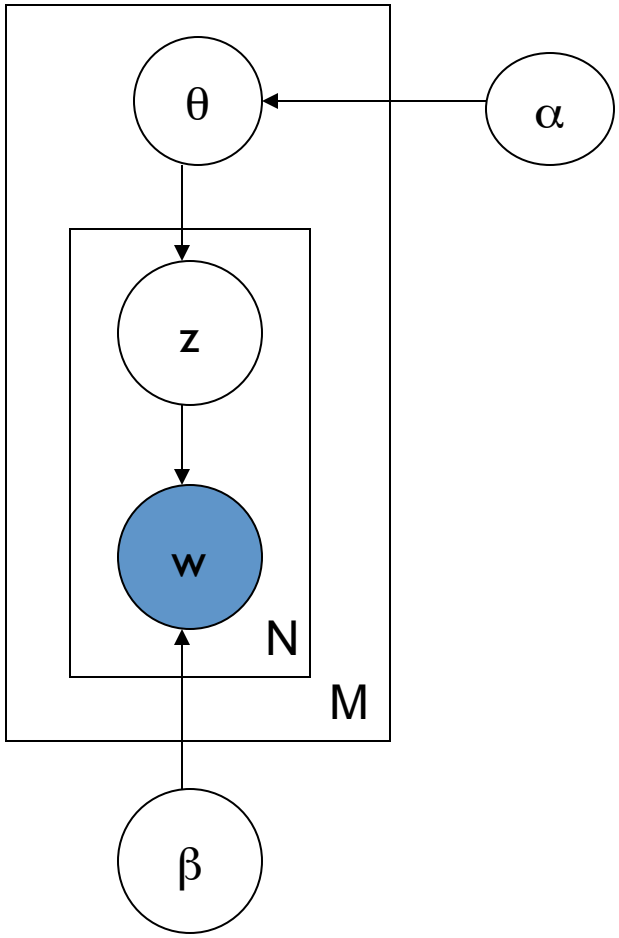
(f) 1-NN: Author disambiguation

| Dataset | PSK | $PIC_D$ | $PIC_R$ | $PIC_{R4}$ | NCut | NJW |
|---|---|---|---|---|---|---|
| AGupta | 0.68 | 0.74 | 0.72 | **0.95** | 0.79 | 0.91 |
| AKumar | 0.82 | 0.69 | 0.74 | **0.85** | 0.79 | 0.81 |
| CChen | 0.77 | 0.73 | 0.74 | **0.89** | 0.75 | 0.85 |
| DJohnson | 0.81 | 0.81 | 0.83 | **0.95** | 0.85 | 0.92 |
| JLee | 0.55 | 0.61 | 0.68 | **0.92** | 0.79 | 0.91 |
| JMartin | 0.77 | 0.73 | 0.73 | **0.88** | 0.75 | 0.85 |
| JRobinson | 0.86 | 0.75 | 0.80 | **0.92** | 0.83 | 0.85 |
| JSmith | 0.65 | 0.75 | 0.67 | **0.93** | 0.85 | 0.91 |
| KTanaka | 0.81 | 0.84 | 0.86 | **0.95** | 0.90 | 0.90 |
| MBrown | 0.83 | 0.78 | 0.82 | **0.93** | 0.86 | 0.89 |
| MJones | 0.79 | 0.69 | 0.71 | **0.91** | 0.90 | 0.89 |
| MMiller | 0.81 | 0.83 | 0.81 | **0.99** | 0.97 | 0.98 |
| SLee | 0.59 | 0.69 | 0.77 | **0.92** | 0.85 | 0.92 |
| YChen | 0.57 | 0.73 | 0.79 | **0.95** | 0.84 | 0.94 |
| **Average** | 0.74 | 0.74 | 0.76 | **0.92** | 0.84 | 0.90 |

Balasubramanyan, Lin, Cohen, NIPS w/s 2010

# Review - LDA

- Latent Dirichlet Allocation with Gibbs




- Randomly initialize each $z_{m,n}$
- Repeat for t=1,….
  - For each doc *m,* word *n*
    - Find Pr($z_{mn}$=*k*|other z's)
    - Sample $z_{mn}$ according to that distr.

# 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
```

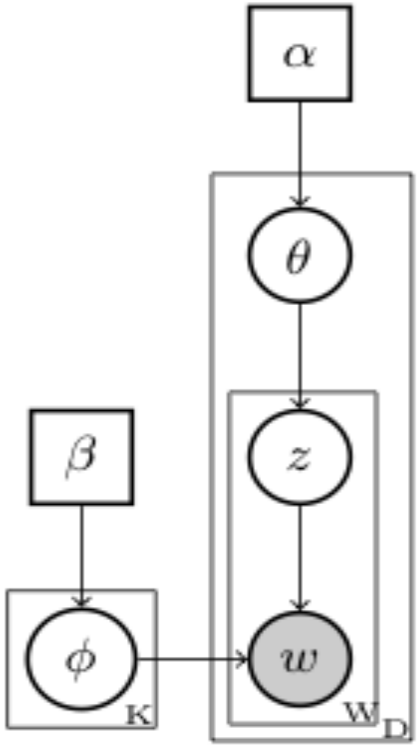



**Figure 1: Graphical model for LDA.**

# 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 <u>new random topic</u>
- 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]*

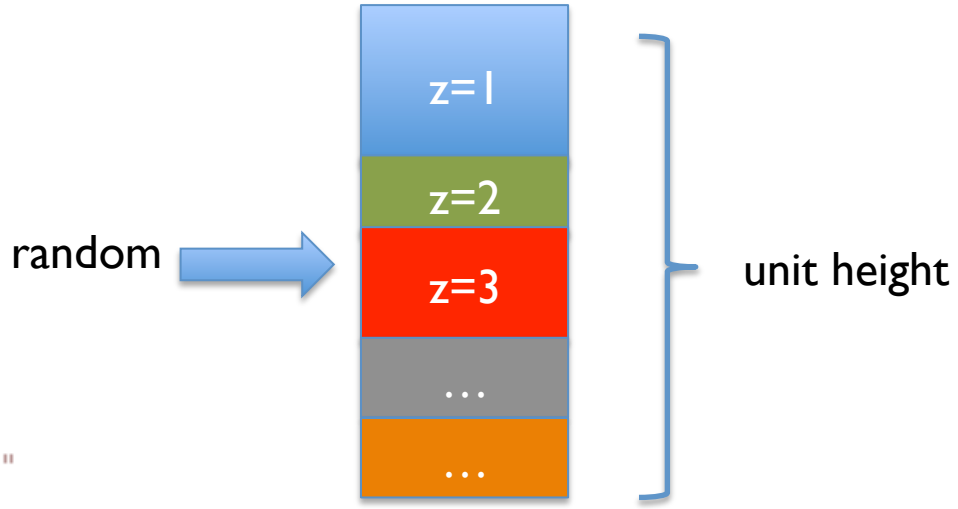
```
def flip(self, d, j, k_old, k_new):
    """update counts to reflect a changed value of z[d][j]"""
    if k_old != k_new:
        w = self.x[d][j]
        self.docTopicCount[d].add(k_old, -1)
        self.wordTopicCount[w].add(k_old, -1)
        self.wordTopicCount[w].add(k_new, +1)
        self.totalTopicCount.add(k_old,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

# Distributed Algorithms for Topic Models

**David Newman** NEWMAN@UCI.EDU
**Arthur Asuncion** ASUNCION@ICS.UCI.EDU
**Padhraic Smyth** SMYTH@ICS.UCI.EDU
**Max Welling** WELLING@ICS.UCI.EDU
*Department of Computer Science*
*University of California, Irvine*
*Irvine, CA 92697, USA*

JMLR 2009

# Observation

- How much does the choice of *z* depend on the other *z's* in the same document?
  - quite a lot
- How much does the choice of *z* depend on the other *z's* in elsewhere in the corpus?
  - maybe not so much
  - depends on Pr(w|t) but that changes slowly
- Can we parallelize Gibbs and still get good results?

# Question

- Can we parallelize Gibbs sampling?
  - formally, no: every choice of $z$ depends on all the other $z$'s
  - Gibbs needs to be sequential
    - just like SGD

# What if you try and parallelize?

Split document/term matrix randomly and distribute to $p$ processors .. then run "Approximate Distributed LDA"

**Algorithm 1** AD-LDA

**repeat**
  **for** each processor $p$ in parallel **do**
    Copy global counts: $N_{wkp} \leftarrow N_{wk}$
    Sample $\mathbf{z}_p$ locally: LDA-Gibbs-Iteration($\mathbf{x}_p, \mathbf{z}_p, N_{kjp}, N_{wkp}, \alpha, \beta$)
  **end for**
  Synchronize
  Update global counts: $N_{wk} \leftarrow N_{wk} + \sum_p (N_{wkp} - N_{wk})$
**until** termination criterion satisfied

# What if you try and parallelize?

| | LDA | AD-LDA |
|---|---|---|
| Space | $N + K(D + W)$ | $\frac{1}{P}(N + KD) + KW$ |
| Time | $NK$ | $\frac{1}{P}NK + KW + C$ |

Table 3: Space and time complexity of LDA and AD-LDA.

D=#docs W=#word(types) K=#topics N=words in corpus

| | KOS | NIPS | WIKIPEDIA | PUBMED | NEWSGROUPS |
|---|---|---|---|---|---|
| $D_{\text{train}}$ | 3,000 | 1,500 | 2,051,929 | 8,200,000 | 19500 |
| $W$ | 6,906 | 12,419 | 120,927 | 141,043 | 27,059 |
| $N$ | 467,714 | 2,166,058 | 344,941,756 | 737,869,083 | 2,057,207 |
| $D_{\text{test}}$ | 430 | 184 | - | - | 498 |

Table 2: Characteristics of data sets used in experiments.

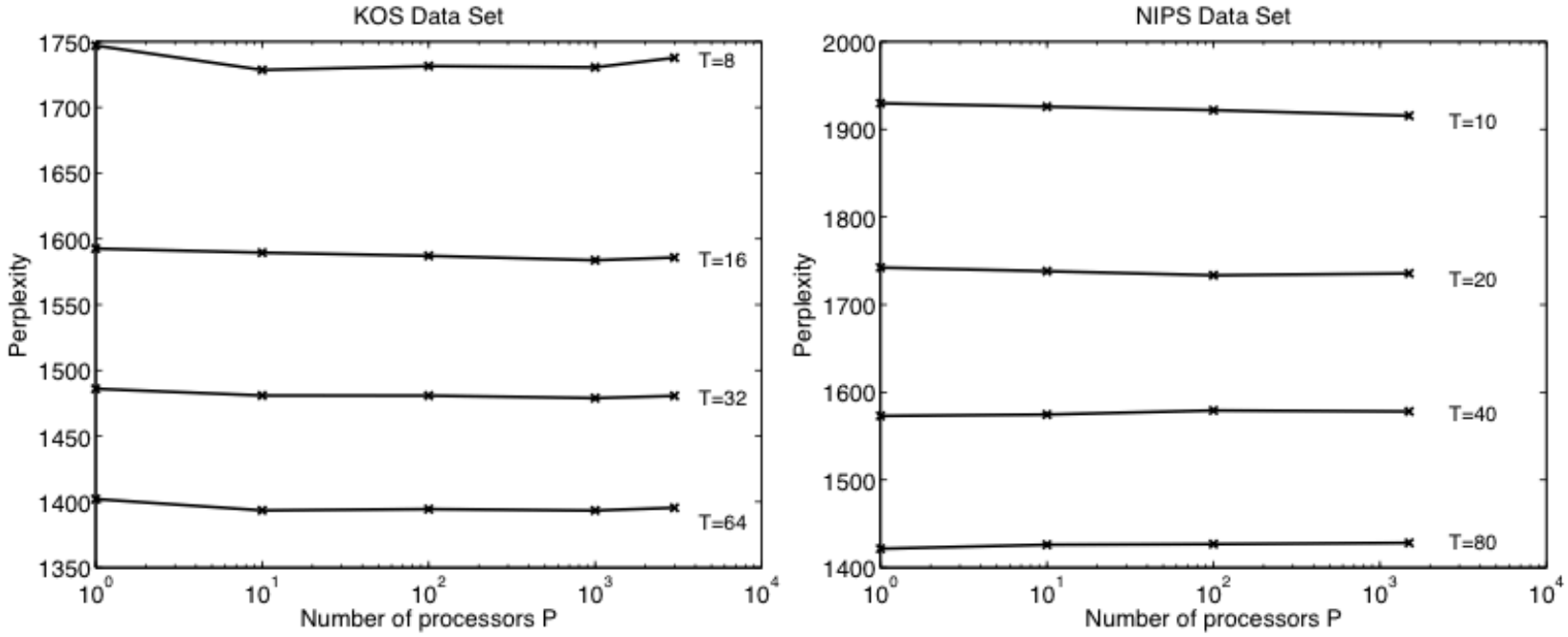
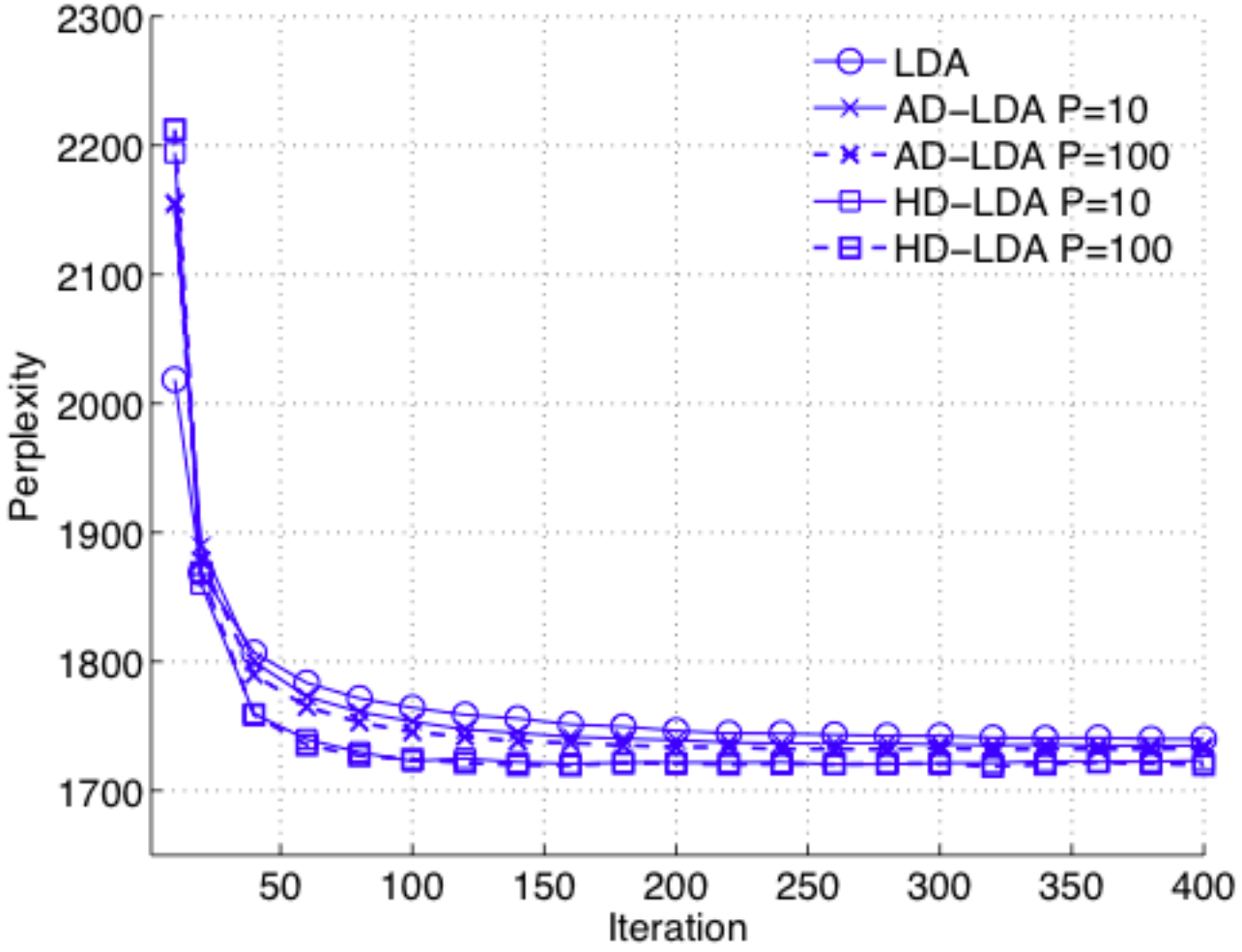



Figure 6: AD-LDA test perplexity versus number of processors up to the limiting case of number of processors equal to number of documents in collection. Left plot shows perplexity for KOS and right plot shows perplexity for NIPS.


2300
2200
2100
2000
1900
1800
1700
Perplexity
50
100
150
200
250
300
350
400
Iteration
LDA
AD−LDA P=10
AD−LDA P=100
HD−LDA P=10
HD−LDA P=100

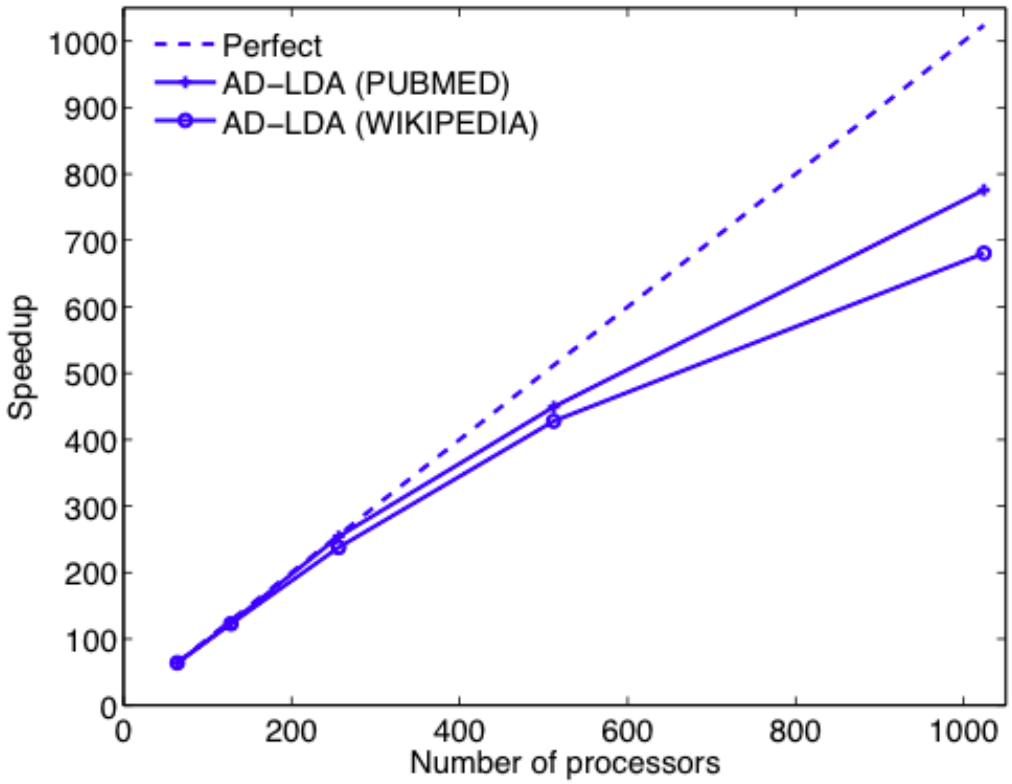



Figure 12: Parallel speedup results for 64 to 1024 processors on multi-million document data sets WIKIPEDIA and PUBMED.

# Update *c. 2014*

- Algorithms:
  - Distributed variational EM
  - Asynchronous LDA (AS-LDA)
  - Approximate Distributed LDA (AD-LDA)
  - Ensemble versions of LDA: HLDA, DCM-LDA
- Implementations:
  - GitHub Yahoo_LDA
    - not Hadoop, special-purpose communication code for synchronizing the global counts
    - Alex Smola, Yahoo→CMU
  - Mahout LDA
    - Andy Schlaikjer, CMU→Twitter

# Efficient Methods for Topic Model Inference on Streaming Document Collections

Limin Yao, David Mimno, and Andrew McCallum
Department of Computer Science
University of Massachusetts, Amherst
{lmyao, mimno, mccallum}@cs.umass.edu

KDD 09

$$P(z = t|w) \quad \propto \quad (\alpha_t + n_{t|d}) \frac{\beta + n_{w|t}}{\beta V + n_{\cdot|t}}.$$

$$P(z = t|w) \propto \frac{\alpha_t \beta}{\beta V + n_{\cdot|t}} + \frac{n_{t|d} \beta}{\beta V + n_{\cdot|t}} + \frac{(\alpha_t + n_{t|d}) n_{w|t}}{\beta V + n_{\cdot|t}}.$$

z=s+r+q

$$\begin{aligned} s &= \sum_t \frac{\alpha_t \beta}{\beta V + n_{\cdot|t}} \\ r &= \sum_t \frac{n_{t|d} \beta}{\beta V + n_{\cdot|t}} \\ q &= \sum_t \frac{(\alpha_t + n_{t|d}) n_{w|t}}{\beta V + n_{\cdot|t}}. \end{aligned}$$

- If $U<s$:
  - lookup $U$ on line segment with tic-marks at $\alpha_1\beta/(\beta V + n_{\cdot|1})$, $\alpha_2\beta/(\beta V + n_{\cdot|2})$, ...
- If $s<U<r$:
  - lookup $U$ on line segment for $r$

$z=s+r+q$

$$
\begin{aligned}
s &= \sum_t \frac{\alpha_t \beta}{\beta V + n_{\cdot|t}} \\
r &= \sum_t \frac{n_{t|d} \beta}{\beta V + n_{\cdot|t}} \\
q &= \sum_t \frac{(\alpha_t + n_{t|d}) n_{w|t}}{\beta V + n_{\cdot|t}}.
\end{aligned}
$$

Only need to check t such that $n_{t|d}>0$

- If $U<s$:
  - lookup $U$ on line segment with tic-marks at $\alpha_1\beta/(\beta V + n_{\cdot|1})$, $\alpha_2\beta/(\beta V + n_{\cdot|2})$, ...
- If $s<U<s+r$:
  - lookup $U$ on line segment for $r$
- If $s+r<U$:
  - lookup $U$ on line segment for $q$

$z=s+r+q$

$$r = \sum_t \frac{n_{t|d}\beta}{\beta V + n_{\cdot|t}}$$

$$q = \sum_t \frac{(\alpha_t + n_{t|d})n_{w|t}}{\beta V + n_{\cdot|t}}.$$

Only need to check t such that $n_{w|t}>0$

Only need to check occasionally (< 10% of the time)

Only need to check t such that $n_{t|d}>0$

Only need to check t such that $n_{w|t}>0$

$z=s+r+q$

$$
\begin{aligned}
s &= \sum_t \frac{\alpha_t \beta}{\beta V + n_{\cdot|t}} \\
r &= \sum_t \frac{n_{t|d} \beta}{\beta V + n_{\cdot|t}} \\
q &= \sum_t \frac{(\alpha_t + n_{t|d}) n_{w|t}}{\beta V + n_{\cdot|t}}.
\end{aligned}
$$

Only need to **store** (and maintain) total words per topic and $\alpha$'s, $\beta$, $V$

Trick; count up $n_{t|d}$ for $d$ when you start working on $d$ and update incrementally

Only need to **store** $n_{t|d}$ for current $d$

Need to **store** $n_{w|t}$ for each word, topic pair …???

$$z=s+r+q \quad \begin{cases} s = \sum_t \frac{\alpha_t \beta}{\beta V + n_{\cdot|t}} \\ r = \sum_t \frac{n_{t|d} \beta}{\beta V + n_{\cdot|t}} \\ q = \sum_t \frac{(\alpha_t + n_{t|d}) n_{w|t}}{\beta V + n_{\cdot|t}}. \end{cases}$$

$$q = \sum_t \left[ \frac{\alpha_t + n_{t|d}}{\beta V + n_{\cdot|t}} \times n_{w|t} \right].$$

1. *Precompute, for each t,* $\frac{\alpha_t + n_{t|d}}{\beta V + n_{\cdot|t}}$

2. *Quickly find t's such that* $n_{w|t}$ *is large for w*

$z=s+r+q$

$$s = \sum_t \frac{\alpha_t \beta}{\beta V + n_{\cdot|t}}$$

$$r = \sum_t \frac{n_{t|d} \beta}{\beta V + n_{\cdot|t}}$$

$$q = \sum_t \frac{(\alpha_t + n_{t|d}) n_{w|t}}{\beta V + n_{\cdot|t}}.$$

Most (>90%) of the time and space is here…

**store** $n_{w|t}$ for each word, topic pair …???

$$q = \sum_t \left[ \frac{\alpha_t + n_{t|d}}{\beta V + n_{\cdot|t}} \times n_{w|t} \right].$$

*1. Precompute, for each t,* $\frac{\alpha_t + n_{t|d}}{\beta V + n_{\cdot|t}}$

*2. Quickly find t's such that* $n_{w|t}$ *is large for w*

- map w to an int array
  - no larger than frequency w
  - no larger than #topics
- encode *(t,n)* as a bit vector
  - *n* in the high-order bits
  - *t* in the low-order bits
- keep ints sorted in descending order

Most (>90%) of the time and space is here…

**store** $n_{w|t}$ for each word, topic pair …???

$$q = \sum_t \frac{(\alpha_t + n_{t|d}) n_{w|t}}{\beta V + n_{\cdot|t}}.$$

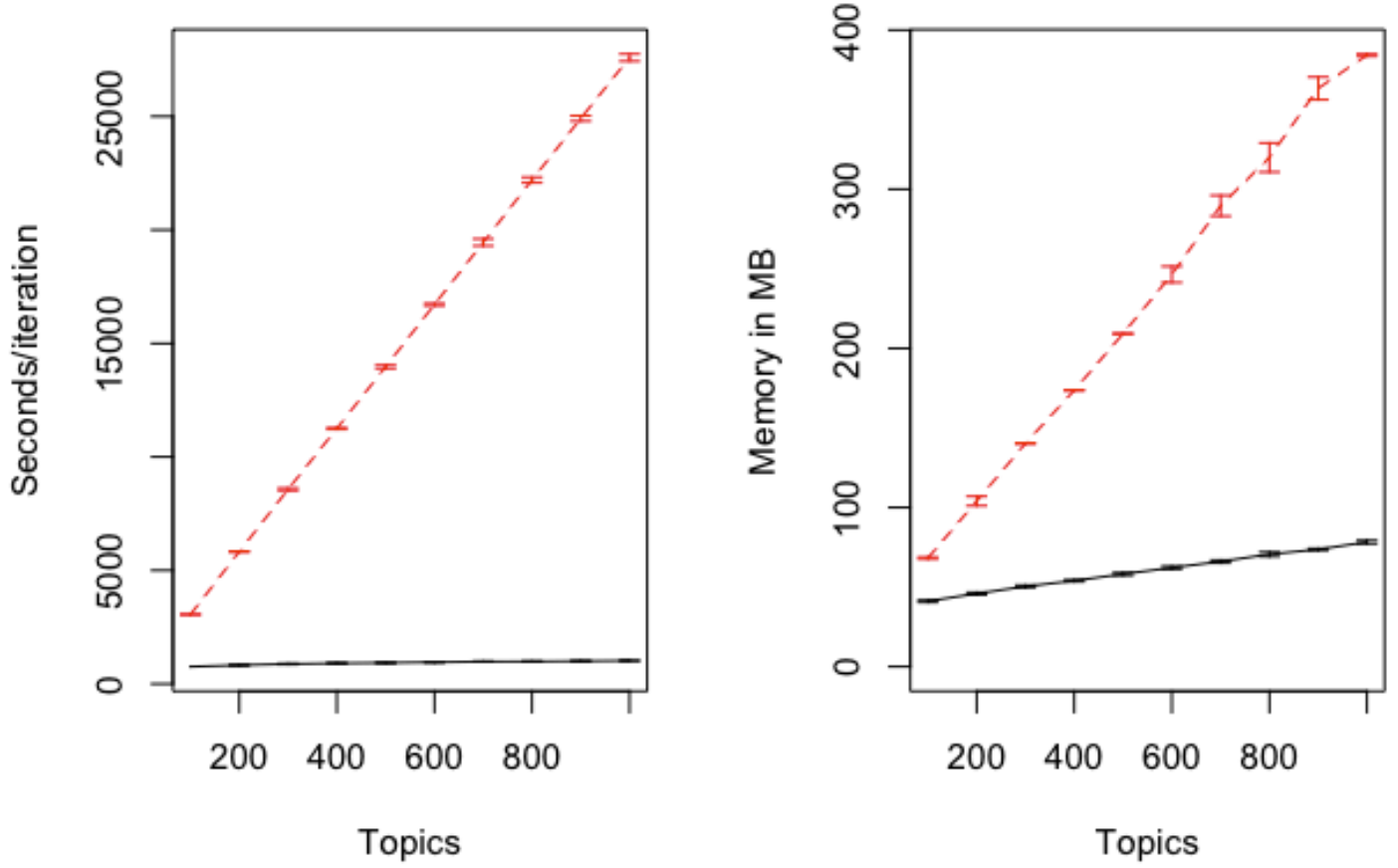



**Figure 2: A comparison of time and space efficiency between standard Gibbs sampling (dashed red lines) and the SparseLDA algorithm and data structure presented in this paper (solid black lines). Error bars show the standard deviation over five runs.**

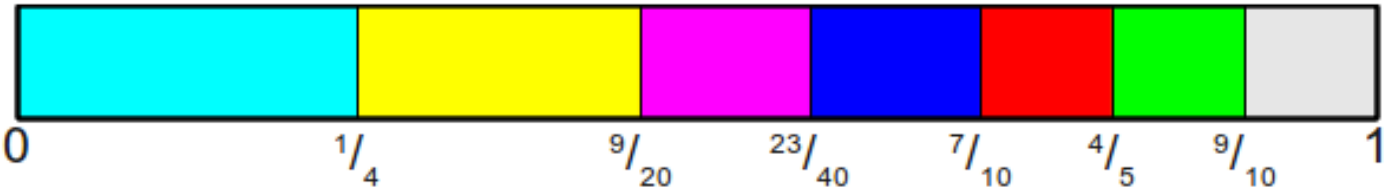
# Outline

- LDA/Gibbs algorithm details
- How to speed it up by parallelizing
- How to speed it up by faster sampling
  - Why sampling is key
  - Some sampling ideas for LDA
    - The Mimno/McCallum decomposition (SparseLDA)
    - **Alias tables** (Walker 1977; Li, Ahmed, Ravi, Smola KDD 2014)

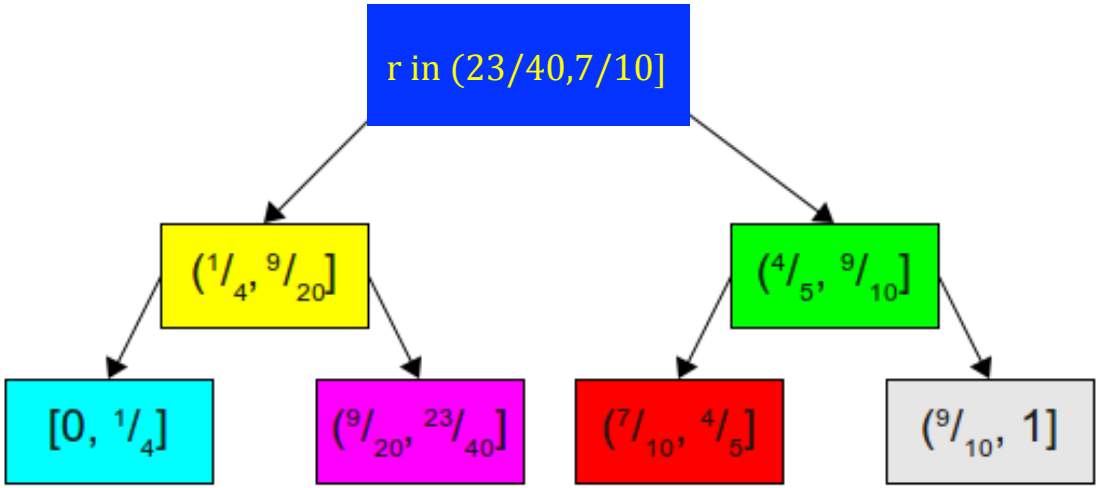
# Alias tables

O(K)

Basic problem: how can we sample from a biased coin quickly?




If the distribution changes slowly maybe we can do some preprocessing and then sample multiple times. Proof of concept: generate *r~uniform* and use a binary tree

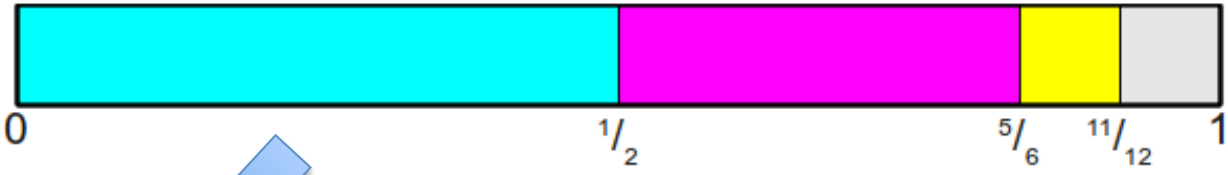


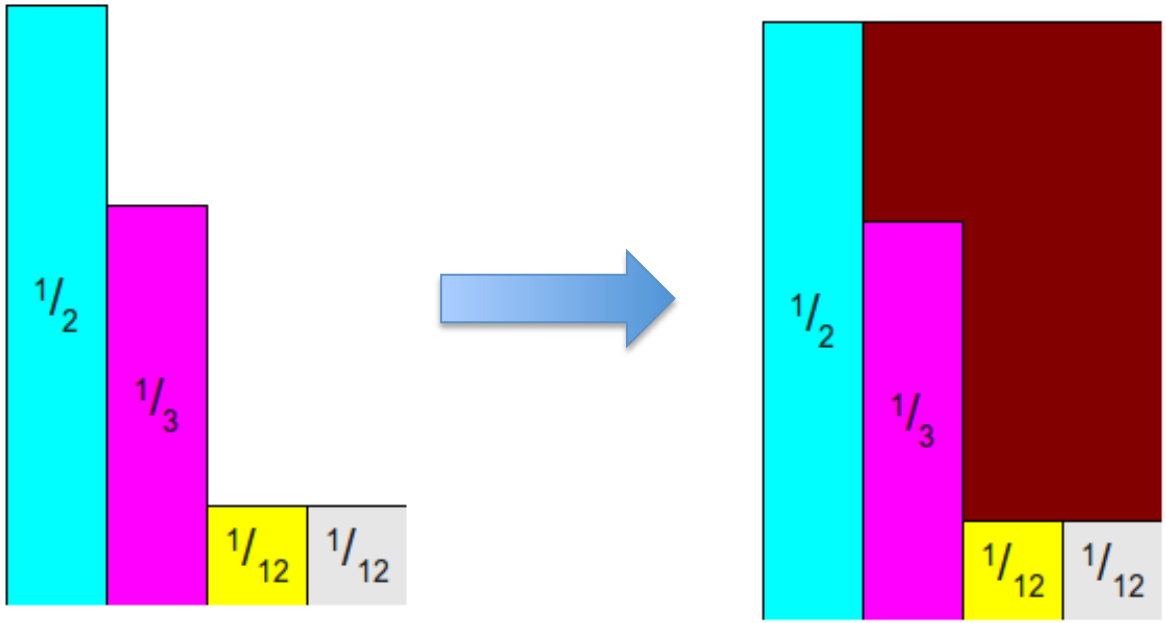

O(log2K)

http://www.keithschwarz.com/darts-dice-coins/

# Alias tables

Another idea…




Simulate the dart with two drawn values:

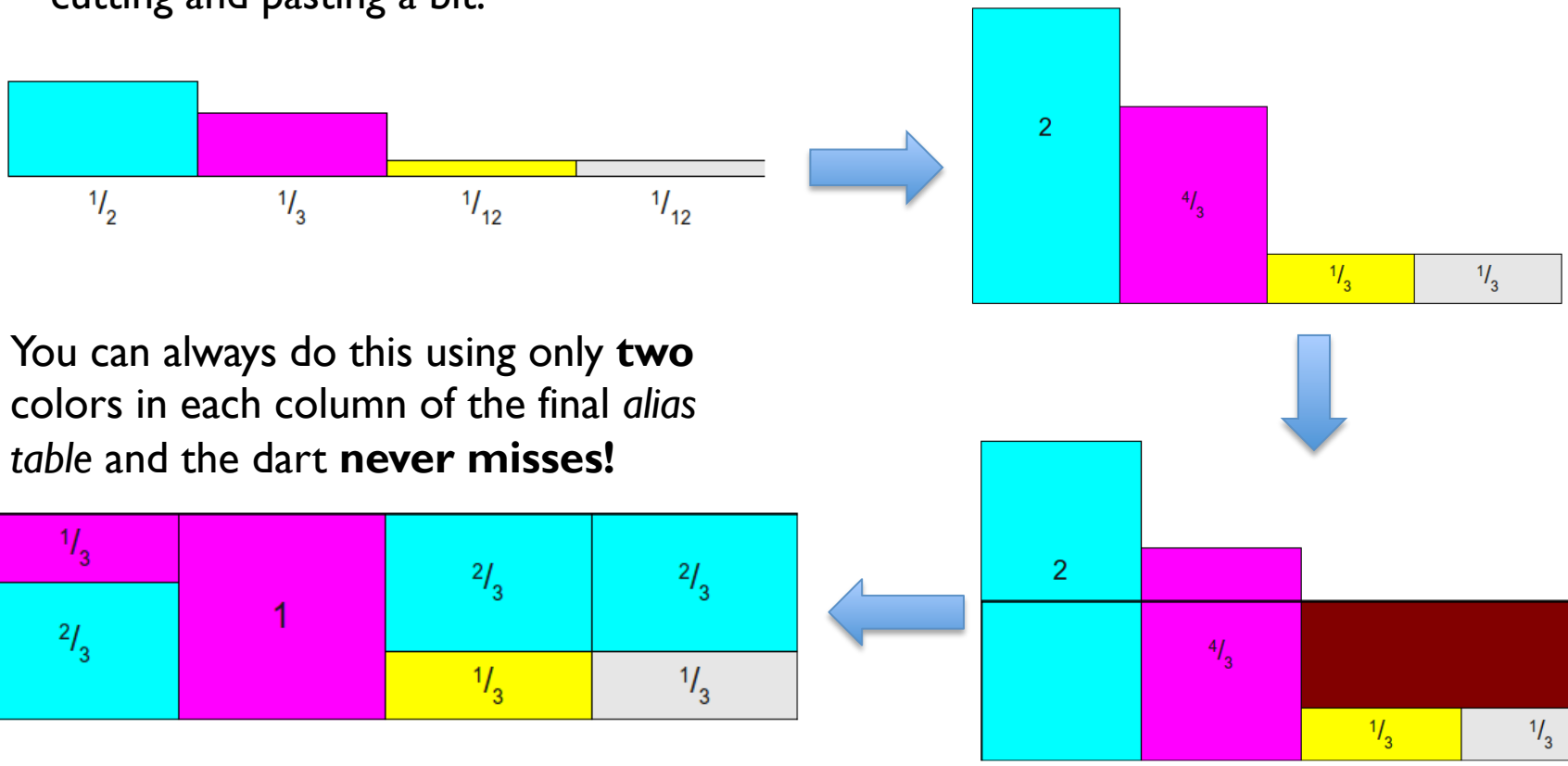
rx ➔ int(u1*K)
ry ➔ u1*$p_{max}$

keep throwing till you hit a stripe

http://www.keithschwarz.com/darts-dice-coins/

# Alias tables

An even more clever idea: minimize the brown space (where the dart "misses") by sizing the rectangle's height to the *average* probability, not the *maximum* probability, and cutting and pasting a bit.




You can always do this using only **two** colors in each column of the final *alias table* and the dart **never misses!**

mathematically speaking…

http://www.keithschwarz.com/darts-dice-coins/