10-405: Scalable Sampling
Outline

• Background (today)
  – directed graphical models
  – specifying a learner with a DGM
  – LDA as a DGM
  – a sampler for LDA

• Scalability (Wed)
  – fast sampling for LDA
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:
\[ \Pr(X_i \mid X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n) = \Pr(X_i \mid \text{parents}(X_i)) \]

- **Generative story (and joint distribution):**
  - Pick \( b \sim \Pr(\text{Burglar}) \), a binomial
  - Pick \( e \sim \Pr(\text{Earthquake}) \), a binomial
  - Pick \( a \sim \Pr(\text{Alarm} \mid \text{Burglar} = b, \text{Earthquake} = e) \), four binomials
  - Pick \( c \sim \Pr(\text{PhoneCall} \mid \text{Alarm}) \)
DGMs: The “Burglar Alarm” example

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

\[ \Pr(\text{Burglar}=\text{true} \mid \text{PhoneCall}=\text{true}) \]

\[ \Pr(\text{Burglar}=\text{true} \mid \text{PhoneCall}=\text{true}, \text{and Earthquake}=\text{true}) \]

• Generative story:
  – Pick \( b \sim \Pr(\text{Burglar}) \), a binomial
  – Pick \( e \sim \Pr(\text{Earthquake}) \), a binomial
  – Pick \( a \sim \Pr(\text{Alarm} \mid \text{Burglar}=b, \text{Earthquake}=e) \), four binomials
  – Pick \( c \sim \Pr(\text{PhoneCall} \mid \text{Alarm}) \)
Inference in DGMs

• General problem: given evidence $E_1, \ldots, E_k$
  compute $P(X \mid E_1, \ldots, E_k)$ for any $X$

• Big assumption: graph is “polytree”
  – $\leq 1$ undirected path between any nodes $X,Y$

• Notation:

\[ E_X^+ = "causal\ support" \text{ for } X \]
\[ E_X^- = "evidential\ support" \text{ for } X \]
DGM Inference: \( P(X|E) \)

\[
P(X | E) = P(X | E^+_X, E^-_X)
\]

\[
= \frac{P(E^-_X | X, E^+_X)P(X | E^+_X)}{P(E^-_X | E^+_X)}
\]

\[
P(X | E) \propto P(E^-_X | X)P(X | E^+_X)
\]

- \( E^+ \): causal support
- \( E^- \): evidential support

\[
P(A|B) = \frac{\text{Pr}(B|A) \text{Pr}(A)}{\text{Pr}(B)}
\]
DGM Inference: $P(X|E^+)$

$$P(X | E^+_X) = \sum_{u_1,u_2} P(X | u_1,u_2,E^+_X)P(u_1,u_2 | E^+_X)$$

$$= \sum_{u_1,u_2} P(X | u_1,u_2,\_\_\_\_) \prod_j P(u_j | E^+_{X_j})$$

CPT table lookup

Recursive call to $P(\_ | E^+)$

$$= \sum_u P(X | u) \prod_j P(u_j | E_{U_j \setminus X})$$

Evidence for $U_j$ 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 $\text{Pr}(X|E^+)$ flows down.
Inference in Bayes nets: $P(E^-|X)$ simplified

$$P(X | E) \propto P(E^-_X | X)P(X | E^+_X)$$

$$P(E^-_X | X) = \prod_j P(E^-_{Yj\setminus X} | X)$$

$$= \prod_j \sum_{y_{jk}} P(y_{jk} | X)P(E^-_{Yj\setminus X} | X, y_{jk})$$

$$= \prod_j \sum_{y_{jk}} P(y_{jk} | X)P(E^-_{Yj} | y_{jk})$$

Recursive call to $P(E^-|\cdot)$

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

$E^-_Y$ : evidential support for $Y_j$

$E^-_{Yj\setminus X}$ : evidence for $Y_j$ excluding support through $X$
Message Passing for BP

- We reduced $P(X | E)$ to product of two recursively calculated parts:
  
  $$P(X=x | E^+) = \sum_{u_1, u_2} P(X | u_1, u_2) \prod_j P(u_j | E^+_X)$$

  - i.e., CPT for $X$ and product of “forward” messages from parents

  $$P(E^- | X=x) = \beta \prod_j \sum_{y_{jk}} P(E_{Y_j}^- | y_{jk}) \sum_{z_{jk}} P(y_{jk} | X, z_{jk}) P(z_{jk} | E_{Z_j \backslash Y_k})$$

  - i.e., combination of “backward” messages from parents, CPTs, and $P(Z | E_{Z \backslash Y_k})$, 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 $Pr(X | Y=y_d)$

---

**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     |
| ...     |         |             |

---

```
Pr(Y=y)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>onion</td>
<td>0.3</td>
</tr>
<tr>
<td>economist</td>
<td>0.7</td>
</tr>
</tbody>
</table>
```

---

```
zymurgy forever!
aardvarks?
learn ai!
```
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 $Pr(X | Y = y_d)$

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

Pr($Y = y$)

<table>
<thead>
<tr>
<th>$Y$</th>
<th>$Pr(Y = y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>onion</td>
<td>0.3</td>
</tr>
<tr>
<td>economist</td>
<td>0.7</td>
</tr>
</tbody>
</table>
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 $Pr(X | Y=y_d)$

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

Plate diagram
Bayesian smoothing for a binomial

MLE: maximize $\text{Pr}(D|\theta)$

MAP: maximize $\text{Pr}(D|\theta)\text{Pr}(\theta)$

Smoothing = prior over the parameter $\theta$

estimate $\Theta=P(\text{heads})$ for a binomial with MLE as:

\[
\hat{\theta} = \frac{\alpha_1}{\alpha_1 + \alpha_0}
\]

and with MAP as:

\[
\hat{\theta} = \frac{(\alpha_1 + \gamma_1)}{(\alpha_1 + \gamma_1) + (\alpha_0 + \gamma_0)}
\]
Smoothing for a binomial as a DGM

MAP for dataset D with $\alpha_1$ heads and $\alpha_2$ tails:

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

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

MAP is a point estimation: want to find max probability parameter $\theta$, to the posterior distribution
Smoothing for a binomial as a DGM

MAP for dataset F of flips with $\alpha_1$ heads and $\alpha_2$ tails:

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

Comment: conjugate for a multinomial is called a **Dirichlet**
Smoothing for a binomial as a DGM

MAP for dataset R with $\alpha_i$ rolls of each face of a multi-sided die...

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

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....
A more Bayesian Naïve Bayes

• From a Dirichlet $\alpha$:
  – Draw a multinomial $\pi$ over the $K$ classes

• From a Dirichlet $\beta$
  – For each class $y=1\ldots K$
    • Draw a multinomial $\gamma[y]$ over the vocabulary

• For each document $d=1\ldots D$:
  – Pick a label $y_d$ from $\pi$
  – For each word in $d$ (of length $N_d$):
    • Pick a word $x_{id}$ from $\gamma[y_d]$
Unsupervised Naïve Bayes

- From a Dirichlet $\alpha$:
  - Draw a multinomial $\pi$ over the $K$ classes
- From a Dirichlet $\beta$
  - For each class $y=1...K$
    - Draw a multinomial $\gamma[y]$ over the vocabulary
- For each document $d=1..D$:
  - Pick a label $y_d$ from $\pi$
  - For each word in $d$ (of length $N_d$):
    - Pick a word $x_{id}$ from $\gamma[y_d]$
Unsupervised Naïve Bayes

• The generative story is the same

• What we do is different
  – We want to both find values of $\gamma$ and $\pi$ 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
Unsupervised NB vs LDA

- One class prior
- One Y per doc
- Different class distrib for each doc
- One Y per word

\[ \alpha \rightarrow \pi \rightarrow Y \rightarrow W \rightarrow N_d \rightarrow D \]

\[ \beta \rightarrow Y \rightarrow K \]
Unsupervised NB vs LDA

- **One class prior**
  - $\alpha \rightarrow \pi$
  - $\pi \rightarrow Y$
  - $Y \rightarrow W$
  - $W \rightarrow Nd$
  - $Nd \rightarrow D$
  - $D \rightarrow \beta$
  - $\beta \rightarrow Y$
  - $Y \rightarrow K$

- **Different class distrib $\theta$ for each doc**
  - $\alpha \rightarrow \theta_d$
  - $\theta_d \rightarrow Z_{di}$
  - $Z_{di} \rightarrow W_d$
  - $W_d \rightarrow Nd$
  - $Nd \rightarrow D$
  - $D \rightarrow \beta$
  - $\beta \rightarrow Y_k$
  - $Y_k \rightarrow K$
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 $d = 1, \ldots, M$
  • Generate $\theta_d \sim D_1(\ldots)$
  • For each word $n = 1, \ldots, N_d$
    • generate $w_n \sim D_2(\cdot | \theta_{dn})$

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 $\beta_k \Rightarrow$ same word suggests same topic
- The smoothness of $\theta_d \Rightarrow$ same document suggests same topic
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.
LDA topics: top words $w$ by $\Pr(w|Z=k)$

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

• Parameter learning:
  – Variational EM
    • ...
  – 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

1. Initialize the variables $x_1, x_2, ..., x_N,$

2. For $r = 1, ..., R$:
   - Sample $x_1^{(r+1)} \sim p(x_1|x_2^{(r)}, x_3^{(r)}, ..., x_N^{(r)})$.
   - Sample $x_2^{(r+1)} \sim p(x_2|x_1^{(r+1)}, x_3^{(r)}, ..., x_N^{(r)})$.
   - ...
   - Sample $x_N^{(r+1)} \sim p(x_N|x_1^{(r+1)}, x_2^{(r+1)}, ..., x_{N-1}^{(r+1)})$.

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 $\theta$

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 \textbf{if} $y_1,\ldots,y_k$ were sampled from the true joint \textbf{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 \textit{space of samples}: how well-connected are they by the sampling steps?
LDA

• Latent Dirichlet Allocation
  • Collapsed Gibbs Sampling
    – What is collapsed Gibbs sampling?
    
    – Here “collapsing” means “marginalizing out” a latent variable
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 $\Pr(\theta|\alpha, \text{sample of } Z’s)$
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 \( Z_1 \sim \Pr(Z_1|z_2,z_3,\alpha) \)

called a Dirichlet-multinomial and it looks like this:

\[
\Pr(Z_1 = k_1, Z_2 = k_2, Z_3 = k_3 | \alpha) = \int_{\theta} \Pr(Z_1 = k_1, Z_2 = k_2, Z_3 = k_3 | \theta) \Pr(\theta | \alpha) d\theta
\]

If there are \( k \) values for the Z’s and \( n_k = \# \text{Z’s with value } k \), then it turns out:

\[
\Pr(Z | \alpha) = \int_{\theta} \Pr(Z | \theta) \Pr(\theta | \alpha) d\theta = \frac{\Gamma\left(\sum_k \alpha_k\right)}{\Gamma\left(\sum_k \alpha_k + \sum_k n_k\right)} \prod_k \frac{\Gamma(n_k + \alpha_k)}{\Gamma(\alpha_k)}
\]
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 $k$
- $Z = (Z_1, \ldots, Z_m)$
- $Z^{(-i)} = (Z_1, \ldots, Z_{i-1}, Z_{i+1}, \ldots, Z_m)$
- $n_k^{(-i)} = \# Z$’s with value $k$ excluding $Z_i$

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

$$Pr(Z | \alpha) = \int_\theta Pr(Z | \theta) Pr(\theta | \alpha) d\theta = \frac{\Gamma \left( \sum_k \alpha_k \right) \prod_k \frac{\Gamma(n_k + \alpha_k)}{\Gamma(\alpha_k)} \Gamma \left( \sum_k \alpha_k + \sum_k n_k \right)}{\Gamma \left( \sum_k \alpha_k \right) \Gamma \left( \sum_k n_k \right)}$$
What about with downstream evidence?

\[
\Pr(Z_i = k \mid Z^{(-i)}, \alpha) \propto n_k^{(-i)} + \alpha_k
\]
captures the constraints on \( Z_i \) via \( \theta \)
(from “above”, “causal” direction)

what about via \( \beta \)?

\[
\Pr(Z) = (1/c) \ast \Pr(Z \mid E^+) \Pr(E^- \mid Z)
\]

\[
\Pr(Z_i = k \mid Z^{(-i)}, X_i = x, \eta) \propto \frac{\#(X = x \text{ with } Z = k \text{ in } Z^{(-i)}) + \text{smoothing}}{\#(X = x) + \text{smoothing}}
\]
Sampling for LDA

Notation:
- $k$ values for the $Z_{d,i}$’s
- $Z^{(-d,i)} = \text{all the Z’s but } Z_{d,i}$
- $n_{w,k} = \# Z_{d,i}$’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 $Z_{i,d}$

\[
Pr(Z_{d,i} = k \mid Z^{(-d,i)}, W_{d,i} = w, \alpha) \propto \frac{Pr(E \mid Z)}{(n^{*}_{k}^{(-i)} + \alpha_{k}) \left( \sum_{w',k} n_{w',k}^{(-d,i)} + \eta_{w'} \right)}
\]

- $\text{fraction of time } Z=k \text{ in doc } d$
- $\text{fraction of time } W=w \text{ in topic } k$
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
# totalWordsCount = 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.....

```python
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 under topic k""
    num = (self.docTopicCount[d][k] + self.alpha)
    denom = (sum(self.docTopicCount[d]) + self.numTopics*self.alpha)
    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

$N[*,*]=V$  
$N[*,k]$  
$N[d,k]$  
$M[w,k]$
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 $d$

```python
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
\[
p(Z_{d,j} = k \mid ..) \propto \Pr(Z_{d,j} = k \mid "d") \times \Pr(W_{d,k} = w \mid Z_{d,j} = k, ..)
= \frac{N[k,d] - C_{d,j,k} + \alpha}{Z} \times \frac{W[w,k] - C_{d,j,k} + \beta}{(W[* ,k] - C_{d,j,k}) + \beta N[* ,* ]}
\]

```python
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):
    
    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
        ak = (self.docTopicCount[d][k] - z_dj_equals_k + self.alpha)
        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