An Intro To Bayesian ML

Prof. Chris Jermaine
cmj4@cs.rice.edu

Before We Begin

- In A8, implement a ML algorithm to extract “topics” from text
- This is not a ML class... so to be fair
  - — Will try to specify algorithm so precisely in the next few lectures
  - — That you can implement it without really understanding what is going on
- That said...
  - — It would be a shame if this is what happend!
  - — So will spend considerable time trying to explain what’s going on
  - — And I hope it’s gonna make sense!
- So sit back, enjoy, and hopefully you’ll learn something!

Modern Machine Learning

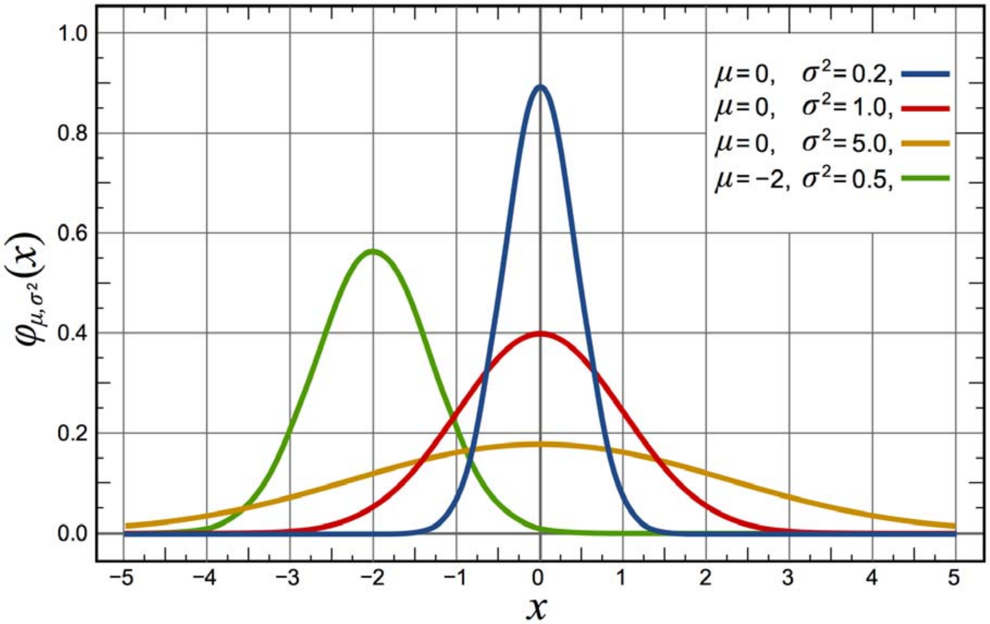
- Is a whole lot like applied statistics
- In fact, I'd argue that CS in general is becoming more statistical
  - That is, based on observation and inference
  - ...and a bit less logic-based
- Ask Google, Facebook, LinkedIn how important stats is!

The Bayesian Approach

- One branch of machine learning utilizes the “Bayesian” approach
- Say our goal was to give E2 to a few students
  - Then use their performance to figure out what the class average will be
  - Want to figure out if E2 is too hard or too easy
- How would a Bayesian do this?

So How Would a Bayesian Do This?

- First imagine a stochastic "generative process" for the data
  - — For the *i*th student, we might imagine $score_i \sim$ Normal (mu, sigma^2)

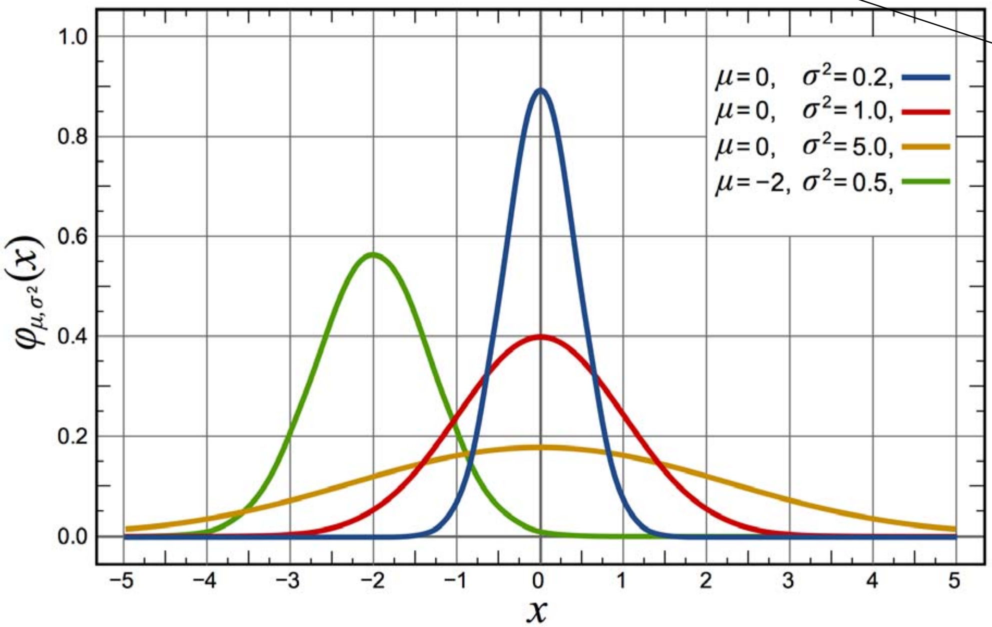


  - — Why normal? Models typical "bell shaped" data
  - — Assume mu, sigma^2 are also generated by sampling from RVs, and are unknown

So How Would a Bayesian Do This?

ultimate goal is to guess the value of mu

- First imagine a stochastic "generative process" for the data
  - — For the ith student, we might imagine $\text{score}_i \sim \text{Normal}(\text{mu}, \text{sigma}^2)$

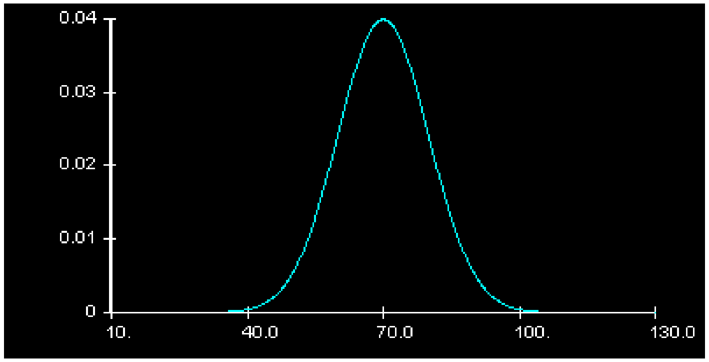


Means "is sampled from" a RV with a normal dist.

  - — Why normal? Models typical "bell shaped" data
  - — Assume mu, sigma^2 are also generated by sampling from RVs, and are unknown

So How Would a Bayesian Do This?

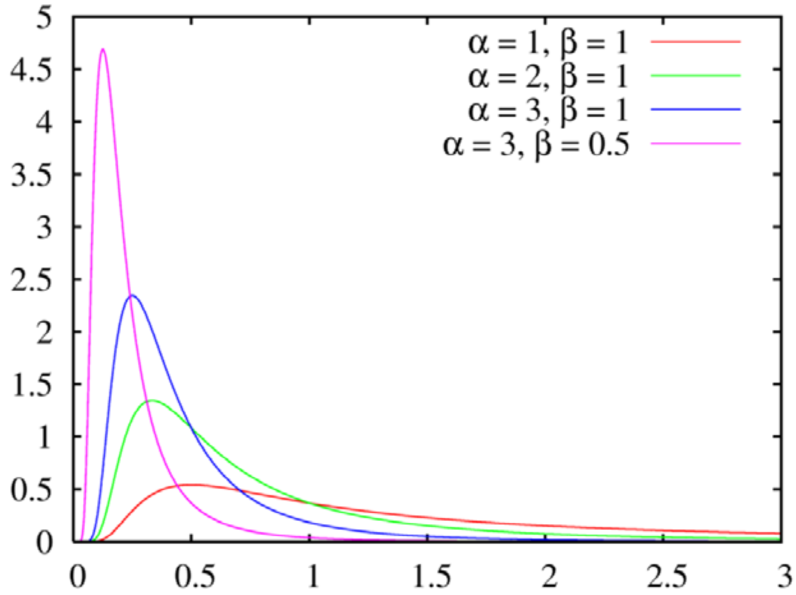
- How is the mean mu generated?
  - — We imagine mu ~ Normal (70, 100)



  - — Why Normal (70, 100)? Allows for all possible, reasonable exam averages

So How Would a Bayesian Do This?

- How is the variance (spread) sigma^2 generated?
  - — We imagine sigma^2 ~ InverseGamma (1, 1)



  - — Why InverseGamma (1, 1)? Allows a really large range of positive sigma^2 vals

Thus, Our Generative Process Is

— Step 1: sigma^2 ~ InverseGamma (1, 1)

— Step 2: mu ~ Normal (75, 100)

— Step 3: for each i, score_i ~ Normal (mu, sigma^2)

Why Have a Generative Process?

- Given gen process, can measure how likely a given pair is
  - — Specifically, *p*(mu, sigma^2) = IG (sigma^2 | 1, 1) x Normal (mu | 75, 100)
  - — So given any mu, sigma^2 combo, we can say how likely we think it is
- This is our "prior" on mean and variance

Note "times" cause the two values are indep.

Bayesian Inference

- To a Bayesian, "inference" is then the process of updating prior expectations after seeing the data
- After we see the data:

test scores

— $p(\text{mu}, \text{sigma}^2 | \text{data}) = \dfrac{p(\text{data} | \text{mu}, \text{sigma}^2) \times p(\text{mu}, \text{sigma}^2)}{p(\text{data})}$

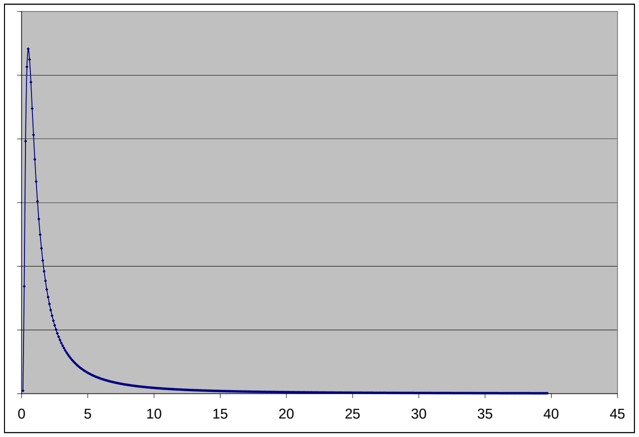
— This is equation follows directly from "Bayes' Theorem"

— Note: this is also a distribution

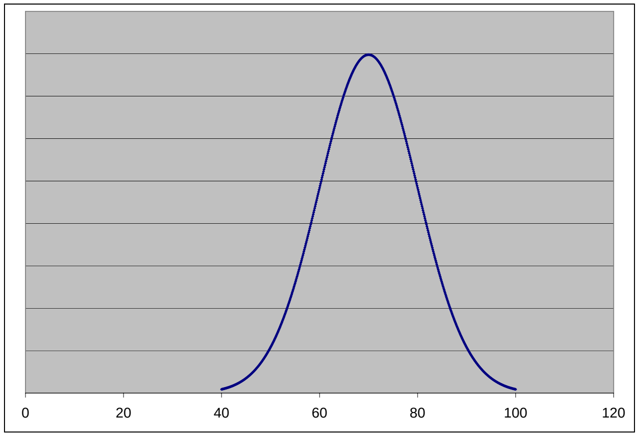
— Known as a "posterior" distribution

Pictorially

- You have a prior distribution on $sigma^2$ and mu:



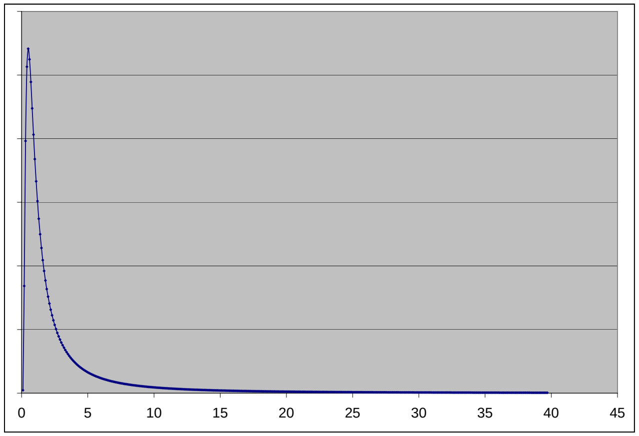
$sigma^2$
(variance of test scores)

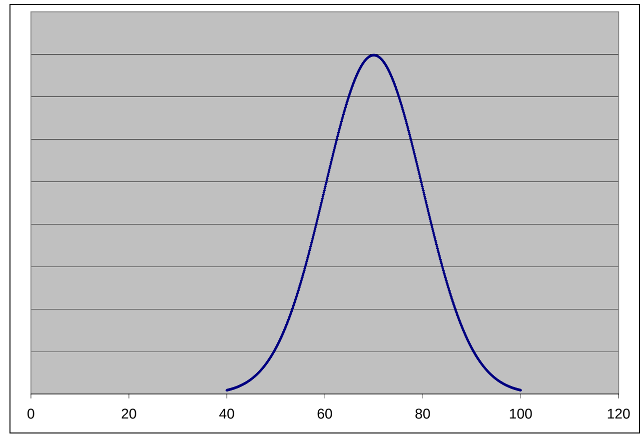


mu
(average of test scores)

Pictorially

- You see some test scores



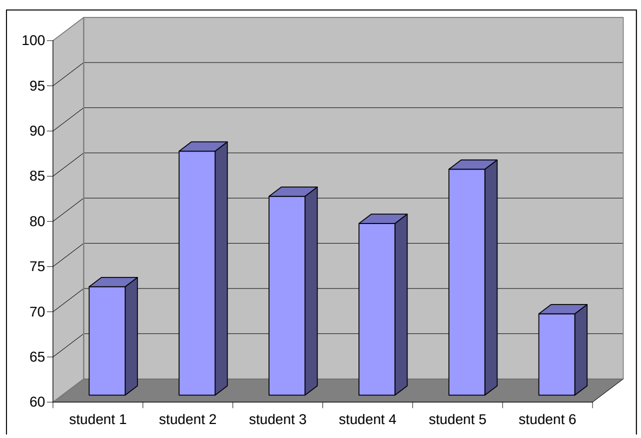


sigma^2

(variance of

mu

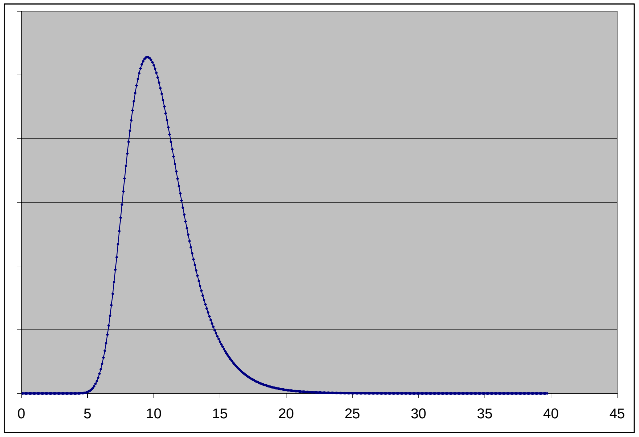
rage of test scores)

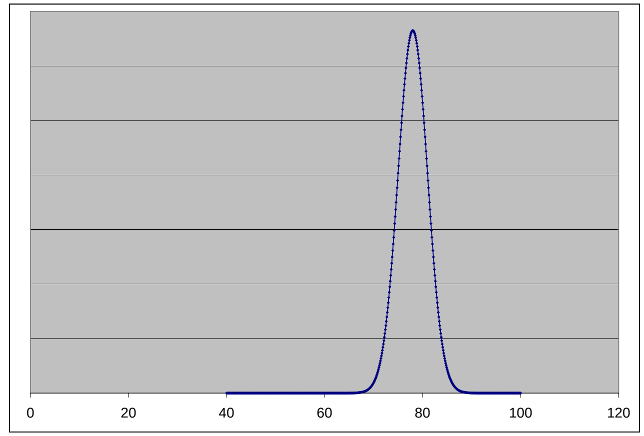


seem to average in the high 70’s

Pictorially

- Then use Bayes' to get a posterior distribution on sigma^2 and mu:



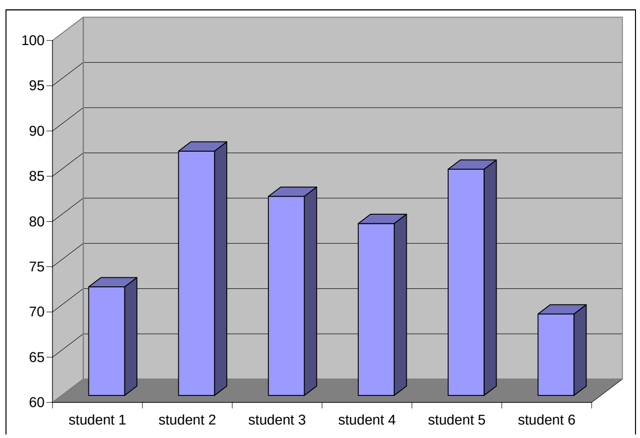


sigma^2

(variance of

mu

erage of test scores)



much narrower
after seeing data!

That’s the Bayesian Approach

- Come up with a generative process
- Which includes prior distributions on the quantities like to est
  - — In our example, the variance and especially the mean
- See some data
- Use Bayes’ Theorem and data to “update” the priors
  - — This gives you a posterior dist
  - — The posterior contains your estimate

OK, So What Does This Have To Do W Text?

- Can apply same methodology to learning topics in text
- This is exactly what “LDA” does
  - — Stands for “Latent Dirichlet Allocation”... fancy!
- First, we need a generative process
  - — LDA will generate *n* random documents given a dictionary
  - — Dictionary is of size num_words
  - — Best shown thru an example
  - — In our example: dictionary will have: (0, “bad”) (1, “I”) (2, “can’t”) (3, “stand”) (4, “comp 215”), (5, “to”) (6, “leave”) (7, “love”) (8, “beer”) (9, “humanities”) (10, “classes”)

LDA Step One

- Generate each of the k “topics”
  - — Each topic is represented by a vector of probabilities
  - — The wth entry in the vector is associated with the wth word in the dictionary
  - — $\text{word\_probs}_t[w]$ is the probability that topic t would produce word w
  - — Vector is sampled from a Dirichlet (alpha) distribution
  - — So, for each t in $\{0...k - 1\}$, $\text{word\_probs}_t \sim$ Dirichlet (alpha)

LDA Step One

- Generate each of the k "topics"
  - — Each topic is represented by a vector of probabilities
  - — The wth entry in the vector is associated with the wth word in the dictionary
  - — $\text{word\_probs}_t[w]$ is the probability that topic t would produce word w
  - — Vector is sampled from a Dirichlet (alpha) distribution
  - — So, for each t in $\{0...k - 1\}$, $\text{word\_probs}_t \sim$ Dirichlet (alpha)
- Ex: $k = 3$
  - — $\text{word\_probs}_0 = (.2, .2, .2, .2, 0, 0, 0, 0, .2, 0, 0)$
  - — $\text{word\_probs}_1 = (0, .2, .2, .2, 0, 0, 0, 0, 0, .2, .2)$
  - — $\text{word\_probs}_2 = (0, .2, .2, 0, .2, 0, .2, .2, 0, 0, 0)$

LDA Step Two

- Generate the topic proportions for each document
  - — Each topic “controls” a subset of the words in a document
  - — topic_probs$_d$[t] is the probability that an arbitrary word in document d will be controlled by topic t
  - — Vector is sampled from a Dirichlet (beta) distribution
  - — So, for each d in {0...n - 1}, topic_probs$_d$ ~ Dirichlet (beta)

LDA Step Two

- Generate the topic proportions for each document
  - — Each topic “controls” a subset of the words in a document
  - — topic_probs$_d$[t] is the probability that an arbitrary word in document d will be controlled by topic t
  - — Vector is sampled from a Dirichlet (beta) distribution
  - — So, for each d in {0...n - 1}, topic_probs$_d$ ~ Dirichlet (beta)
- Ex: $n = 4$
  - — topic_probs$_0$ = (.98, 0.01, 0.01)
  - — topic_probs$_1$ = (0.01, .98, 0.01)
  - — topic_probs$_2$ = (0.02. .49, .49)
  - — topic_probs$_3$ = (.98, 0.01, 0.01)

LDA Step Three

- Generate the words in each document
  - — Each topic “controls” a subset of the words in a document
  - — words_in_doc$_d$[w] is the number of occurences of word w in document d
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc d:

    (1) Figure out the topic t that controls it by sampling from a Multinomial (topic_probs$_d$, 1) distribution

    (2) Generate the word by sampling from a Multinomial (word_probs$_t$, 1) distribution

LDA Step Three

- Generate the words in each document
  - — Each topic "controls" a subset of the words in a document
  - — $\text{words\_in\_doc}_d[w]$ is the number of occurences of word w in document d
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc d:

    (1) Figure out the topic t that controls it by sampling from a Multinomial (topic_probs_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (word_probs_t, 1) distribution
- Ex: doc 0... $\text{topic\_probs}_0 = (.98, 0.01, 0.01)$
  - — t for word zero is...

LDA Step Three

- Generate the words in each document
  - — Each topic “controls” a subset of the words in a document
  - — $\text{words\_in\_doc}_d[w]$ is the number of occurences of word w in document d
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc d:

    (1) Figure out the topic t that controls it by sampling from a Multinomial (topic_probs_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (word_probs_t, 1) distribution

- Ex: doc 0... topic_probs_0 = = (.98, 0.01, 0.01)
  - — t for word zero is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]
  - — So we generate the word using word_probs_0 = (.2, .2, .2, .2, 0, 0, 0, 0, .2, 0, 0)

LDA Step Three

- Generate the words in each document
  - — Each topic "controls" a subset of the words in a document
  - — words_in_doc$_d$[w] is the number of occurences of word w in document d
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc d:

    (1) Figure out the topic t that controls it by sampling from a Multinomial (topic_probs$_d$, 1) distribution

    (2) Generate the word by sampling from a Multinomial (word_probs$_t$, 1) distribution

- Ex: doc 0... topic_probs$_0$ = = (.98, 0.01, 0.01) "I"
  - — t for word zero is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]
  - — So we generate the word using word_probs$_0$ = (.2, .2, .2, .2, 0, 0, 0, 0, .2, 0, 0)
  - — And we get (0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0), which is equivalent to "I"

LDA Step Three

- Generate the words in each document
  - — Each topic “controls” a subset of the words in a document
  - — words_in_doc$_d$[w] is the number of occurences of word w in document d
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc d:
    - (1) Figure out the topic t that controls it by sampling from a Multinomial (topic_probs$_d$, 1) distribution
    - (2) Generate the word by sampling from a Multinomial (word_probs$_t$, 1) distribution
- Ex: doc 0... topic_probs$_0$ = = (.98, 0.01, 0.01) “I”
  - — Now onto the next word

LDA Step Three

- Generate the words in each document
  - — Each topic “controls” a subset of the words in a document
  - — words_in_doc$_d$[w] is the number of occurences of word w in document d
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc d:

    (1) Figure out the topic t that controls it by sampling from a Multinomial (topic_probs$_d$, 1) distribution

    (2) Generate the word by sampling from a Multinomial (word_probs$_t$, 1) distribution
- Ex: doc 0... topic_probs$_0$ = = (.98, 0.01, 0.01) “I”
  - — t for word one is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]

LDA Step Three

- Generate the words in each document
  - — Each topic “controls” a subset of the words in a document
  - — words_in_doc$_d$[w] is the number of occurences of word w in document d
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc d:
    - (1) Figure out the topic t that controls it by sampling from a Multinomial (topic_probs$_d$, 1) distribution
    - (2) Generate the word by sampling from a Multinomial (word_probs$_t$, 1) distribution
- Ex: doc doc 0... topic_probs$_0$ = = (.98, 0.01, 0.01) “I can’t”
  - — t for word one is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]
  - — So we generate the word using word_probs$_0$ = (.2, .2, .2, .2, 0, 0, 0, 0, .2, 0, 0)
  - — And we get (0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0), which is equivalent to “can’t”

LDA Step Three

- Generate the words in each document
  - — Each topic “controls” a subset of the words in a document
  - — $\text{words\_in\_doc}_d[w]$ is the number of occurences of word w in document d
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc d:

    (1) Figure out the topic t that controls it by sampling from a Multinomial (topic_probs_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (word_probs_t, 1) distribution
- Ex: doc doc 0... topic_probs_0 = = (.98, 0.01, 0.01) “I can’t”
  - — Now onto the next word

LDA Step Three

- Generate the words in each document
  - — Each topic “controls” a subset of the words in a document
  - — words_in_doc$_d$[w] is the number of occurences of word w in document d
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc d:

    (1) Figure out the topic t that controls it by sampling from a Multinomial (topic_probs$_d$, 1) distribution

    (2) Generate the word by sampling from a Multinomial (word_probs$_t$, 1) distribution

- Ex: doc 0... topic_probs$_0$ = (.98, 0.01, 0.01) “I can’t”
  - — t for word two is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]

LDA Step Three

- Generate the words in each document
  - — Each topic “controls” a subset of the words in a document
  - — $\text{words\_in\_doc}_d[w]$ is the number of occurences of word *w* in document *d*
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc *d*:

    (1) Figure out the topic *t* that controls it by sampling from a Multinomial (topic_probs_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (word_probs_t, 1) distribution

- Ex: doc 0... topic_probs_0 = (.98, 0.01, 0.01) “I can’t stand”
  - — *t* for word two is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]
  - — So we generate the word using word_probs_0 = (.2, .2, .2, .2, 0, 0, 0, 0, .2, 0, 0)
  - — And we get (0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0), which is equivalent to “stand”

LDA Step Three

- Generate the words in each document
  - — Each topic “controls” a subset of the words in a document
  - — $\text{words\_in\_doc}_d[w]$ is the number of occurences of word *w* in document *d*
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc *d*:

    (1) Figure out the topic *t* that controls it by sampling from a Multinomial (topic_probs_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (word_probs_t, 1) distribution

- Ex: doc 0... topic_probs_0 = (.98, 0.01, 0.01) “I can’t stand”
  - — Onto next word

LDA Step Three

- Generate the words in each document
  - — Each topic "controls" a subset of the words in a document
  - — words_in_doc$_d$[w] is the number of occurences of word w in document d
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc d:

    (1) Figure out the topic t that controls it by sampling from a

    Multinomial (topic_probs$_d$, 1) distribution

    (2) Generate the word by sampling from a

    Multinomial (word_probs$_t$, 1) distribution

- Ex: doc 0... topic_probs$_0$ = (.98, 0.01, 0.01) "I can't stand"
  - — t for word three is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]

LDA Step Three

- Generate the words in each document
  - — Each topic "controls" a subset of the words in a document
  - — $\text{words\_in\_doc}_d[w]$ is the number of occurences of word *w* in document *d*
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc *d*:

    (1) Figure out the topic *t* that controls it by sampling from a Multinomial (topic_probs_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (word_probs_t, 1) distribution

- Ex: doc 0... topic_probs_0 = (.98, 0.01, 0.01) "I can't stand bad"
  - — *t* for word three is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]
  - — So we generate the word using word_probs_0 = (.2, .2, .2, .2, 0, 0, 0, 0, .2, 0, 0)
  - — And we get (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), which is equivalent to "bad"

LDA Step Three

- Generate the words in each document
  - — Each topic “controls” a subset of the words in a document
  - — words_in_doc$_d$[w] is the number of occurences of word w in document d
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc d:

    (1) Figure out the topic t that controls it by sampling from a Multinomial (topic_probs$_d$, 1) distribution

    (2) Generate the word by sampling from a Multinomial (word_probs$_t$, 1) distribution
- Ex: doc 0... topic_probs$_0$ = (.98, 0.01, 0.01) “I can’t stand bad”
  - — Onto the last word in the document

LDA Step Three

- Generate the words in each document
  - — Each topic “controls” a subset of the words in a document
  - — $\text{words\_in\_doc}_d[w]$ is the number of occurences of word w in document d
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc d:

    (1) Figure out the topic t that controls it by sampling from a Multinomial (topic_probs_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (word_probs_t, 1) distribution

- Ex: doc 0... topic_probs_0 = (.98, 0.01, 0.01) “I can’t stand bad”
  - — t for word three is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]

LDA Step Three

- Generate the words in each document
  - — Each topic “controls” a subset of the words in a document
  - — $\text{words\_in\_doc}_d[w]$ is the number of occurences of word *w* in document *d*
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc *d*:

    (1) Figure out the topic *t* that controls it by sampling from a Multinomial (topic_probs_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (word_probs_t, 1) distribution

- Ex: doc 0... topic_probs_0 = (.98, 0.01, 0.01) “I can’t stand bad beer”
  - — *t* for word three is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]
  - — So we generate the word using word_probs_0 = (.2, .2, .2, .2, 0, 0, 0, 0, .2, 0, 0)
  - — And we get (0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0), which is equivalent to “beer”

In The End… For Doc 0…

- text is "I can't stand bad beer" (equiv. to "1 2 3 0 8")
- topic_probs_0 = (.98, 0.01, 0.01)
- words_in_doc_0 = (1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0)
  - — Why? Word 0 appears once, word 1 appears once, word 4 zero times, etc.
- produced_0= (1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0)

  (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

  (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
  - — Why? Topic 0 (associated with first line) produced 5 words

    Those words were (1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0)
  - — Topic 1, topic 2 produced no words
  - — "produced" always a matrix with num_words cols, *k* rows

Repeat For Each Doc in the Corpus!

For Example, Let's Look At Doc 2…

- topic_probs$_2$ = (.02, 0.49, 0.49)
- Imagine that when we generate doc 2, we get:
  - — Word 0: produced by topic 2, is 1 or "I"
  - — Word 1: produced by topic 2, is 7 or "love"
  - — Word 2: produced by topic 2, is 8 or "beer"
  - — Word 3: produced by topic 1, is 1 or "I"
  - — Word 4: produced by topic 1, is 2 or "can't"
  - — Word 5: produced by topic 2, is 7 or "love"
  - — Word 6: produced by topic 1, is 9 or "humanities"
  - — Word 7: produced by topic 1, is 10 or "classes"
- words_in_doc$_2$ = (0, 2, 1, 0, 0, 0, 0, 2, 1, 1, 1)
- produced$_2$= (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
  (0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1)
  (0, 1, 0, 0, 0, 0, 0, 2, 1, 0, 0)

OK We've Got Our Generative Process

- Now, someone gives us an actual, real-life corpus
  - — This means we have got a dictionary
  - — And we have words_in_doc$_d$ for all d in $\{0 \ldots n - 1\}$
- Our goal is to figure out a posterior distribution on
  - — word_probs$_t$ for all t in $\{0 \ldots k - 1\}$
  - — topic_probs$_d$ for all d in $\{0 \ldots n - 1\}$
  - — produced$_d$ for all d in $\{0 \ldots n - 1\}$
- Why?
  - — The hope is this will reveal something about the corpus
  - — For example, what the different topics in the corpus are
  - — And what topics are present in each document

As In All Applications of Bayesian ML

- The posterior is derived using Bayes' Theorem...

$$p\,(\text{word\_probs}_{all}, \text{topic\_probs}_{all}, \text{produced}_{all} \mid \text{words\_in\_doc}_{all}) =$$

$$p\,(\text{words\_in\_doc}_{all} \mid \text{word\_probs}_{all}, \text{topic\_probs}_{all}, \text{produced}_{all}) \quad \times$$

$$p\,(\text{word\_probs}_{all}, \text{topic\_probs}_{all}, \text{produced}_{all}) \,/\, p\,(\text{words\_in\_doc}_{all})$$

We Can Write This Formula

- But what does this do for us?
- In the simple “guess the test score average” case...
  - — We had a couple of posterior distributions that we could plot
  - — It was quite intuitive
- Not the case here!
  - — Got all kinds of weird, multi-dimensions distributions
  - — How to proceed?

With More Complicated Models Such as LDA

- It is common to resort to something called “MCMC”
  — stands for “Markov Chain Monte Carlo”
- MCMC is a class of algorithms
  — That can often be used to draw samples from even the most complex distributions
  — Many of the ideas behind MCMC came out the the Manhatten project
- Using MCMC, we can actually draw samples from

  p (word_probs$_{all}$, topic_probs$_{all}$, produced$_{all}$ | words_in_doc$_{all}$)
- Each “sample” will contain everything in the model!
  — word_probs$_t$ for all t in {0...k - 1}
  — topic_probs$_d$ for all d in {0...n - 1}
  — produced$_d$ for all d in {0...n - 1}

Why Useful?

- Can take one sample, use it as a representative set of values
- Or take many samples, use to get a summary of the distribution

How Does MCMC Work?

- Could spend an entire semester on MCMC alone
- Many flavors of MCMC
- For LDA, we'll employ a simple MCMC methodology
  - — A "Gibbs Sampler"

In the Case of LDA

- Here is pseudo-code for the Gibbs sampler:

  Choose consistent, initial values for word_probs$_{all}$, topic_probs$_{all}$, produced$_{all}$

  For i = 1 to num_iters do:

  For all t in {0...k - 1}:

  word_probs$_t$ ~ p (word_probs$_t$ | topic_probs$_{all}$, produced$_{all}$, words_in_doc$_{all}$)

  For all d in {0...n - 1}:

  topic_probs$_d$ ~ p (topic_probs$_d$ | word_probs$_{all}$, produced$_{all}$, words_in_doc$_{all}$)

  For all d in {0...n - 1}:

  produced$_d$ ~ p (produced$_d$ | word_probs$_{all}$, topic_probs$_{all}$, words_in_doc$_{all}$)

- Run this loop for awhile
  - Then at any point, stop
  - Current word_probs$_{all}$, topic_probs$_{all}$, produced$_{all}$ are a sample from posterior!

So All That's Left

- Is to give pseudo-code for each of the three steps

Word_Probs

- To sample each word_probs$_t$

  - — Create a vector called "counter", where counter is $\sum_d$ produced$_{d,t}$
  - — Note that produced$_{d,t}$ denotes the tth row of the produced matrix for doc d
  - — This counts the number of times topic t produced each word w
  - — Then word_probs$_t$ ~ Dirichlet (counter + alpha)

Constant used in generative process
We assume we know this,
or can guess it

Word_Probs

- To sample each word_probs$_t$
  - — Create a vector called "counter", where counter is $\sum_d$ produced$_{d,t}$
  - — Note that produced$_{d,t}$ denotes the tth row of the produced matrix for doc d
  - — This counts the number of times topic t produced each word w
  - — Then word_probs$_t$ ~ Dirichlet (counter + alpha)
- Intuitively
  - — You are "guessing" the probability topic t will produce each word
  - — If counter[w] is large, then topic t produced w quite often in practice...
  - — And the Dirichlet is then likely to give that word a high probability

Topic_Probs

- To sample each topic_probs$_d$
  - — Create a vector "counter", where counter[t] is $\sum_w$ produced$_{d,t}$[w]
  - — That is, counter[t] is the sum over the tth row in the produced$_d$ matrix
  - — This counts the number of times topic t was used to produce a word in d
  - — Then topic_probs$_d$ ~ Dirichlet (counter + beta)

Again, constant used in generative process
We assume we know this,
or can guess it

Topic_Probs

- To sample each topic_probs$_d$
  - — Create a vector “counter”, where counter[t] is $\sum_w$ produced$_{d,t}$[w]
  - — That is, counter[t] is the sum over the tth row in the produced$_d$ matrix
  - — This counts the number of times topic t was used to produce a word in d
  - — Then topic_probs$_d$ ~ Dirichlet (counter + beta)
- Intuitively
  - — You are “guessing” the probability a word in d will come from each topic
  - — If counter[t] is large, then topic t produced a lot of words in d...
  - — And the Dirichlet is then likely to give that topic a high probability

Produced

- To sample each produced_d
  - — For each word w:
    - (1) Create a vector "probs", where $\text{probs}[t] = \text{word\_probs}_t[w] \times \text{topic\_probs}_d[t]$
    - (2) Normalize probs
    - (3) Then (wth column in produced_d) ~ Multinomial (words_in_doc_d, probs)

Produced

- To sample each produced_d
  - — For each word w:
    - (1) Create a vector "probs", where $\text{probs}[t] = \text{word\_probs}_t[w] \times \text{topic\_probs}_d[t]$
    - (2) Normalize probs
    - (3) Then (wth column in produced_d) ~ Multinomial (words_in_doc_d, probs)
- Intuitively
  - — "probs" is telling you how likely each topic is to be the one responsible for an occurence of *w* in *d*
  - — Then you are using the multinomial to guess how many occurs of *w* each topic is reponsible for
  - — probs[*t*] large means *t* expectedly responsible for more occurs of *w*

This Completes LDA. Questions?