EM wrap-up

EM for POS Tagging w/Dict

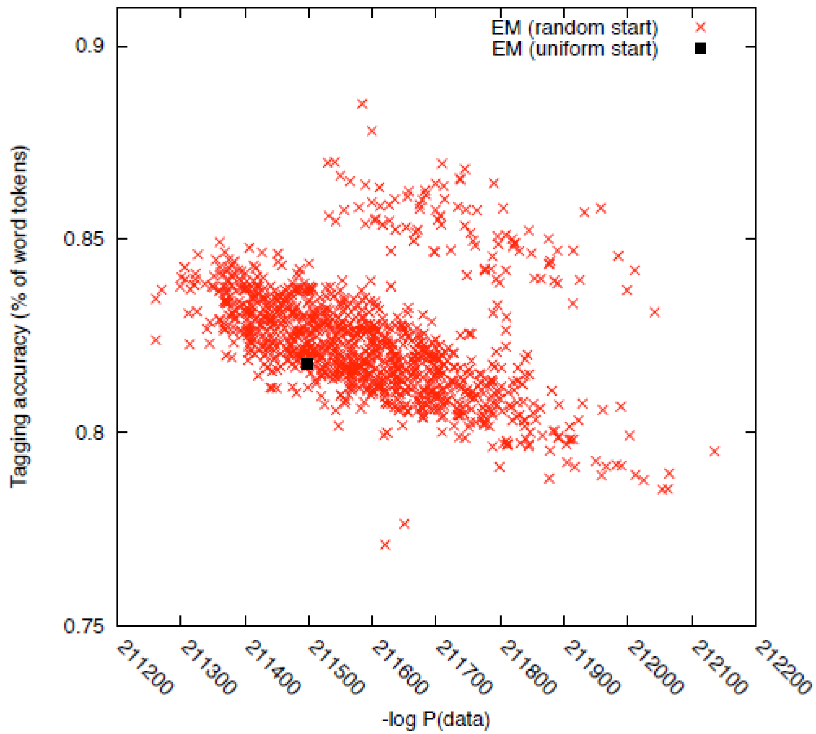
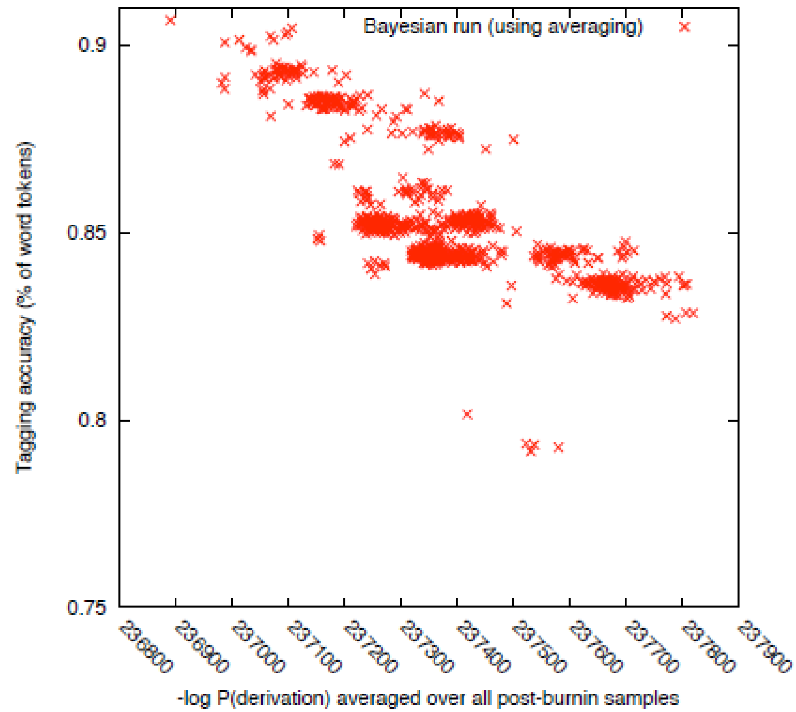


Figure 4: Multiple EM restarts for POS tagging. Each point represents one random restart; the *y*-axis is tagging accuracy and the *x*-axis is EM's objective function, $-\log P(\text{data})$.



Bayesian inference

(Chiang, Graehl, Knight, Pauls, Ravi 2010)

EM for POS Tagging w/o Dict

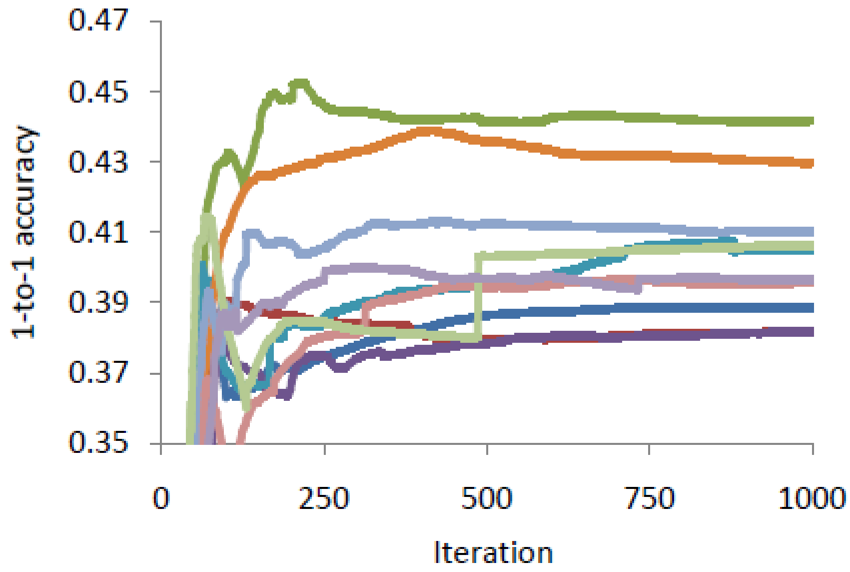
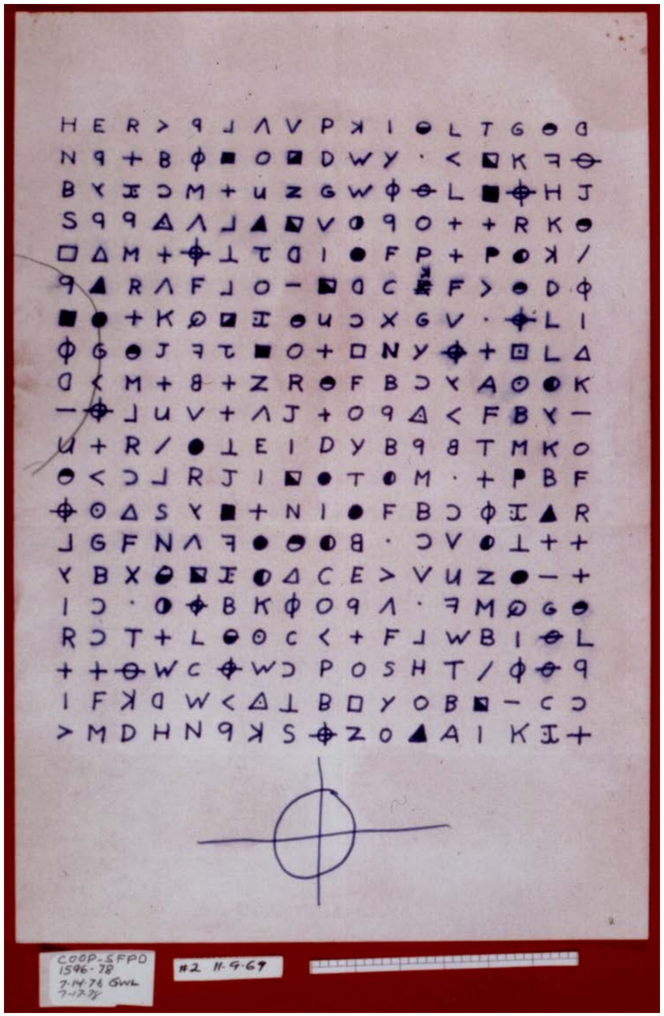


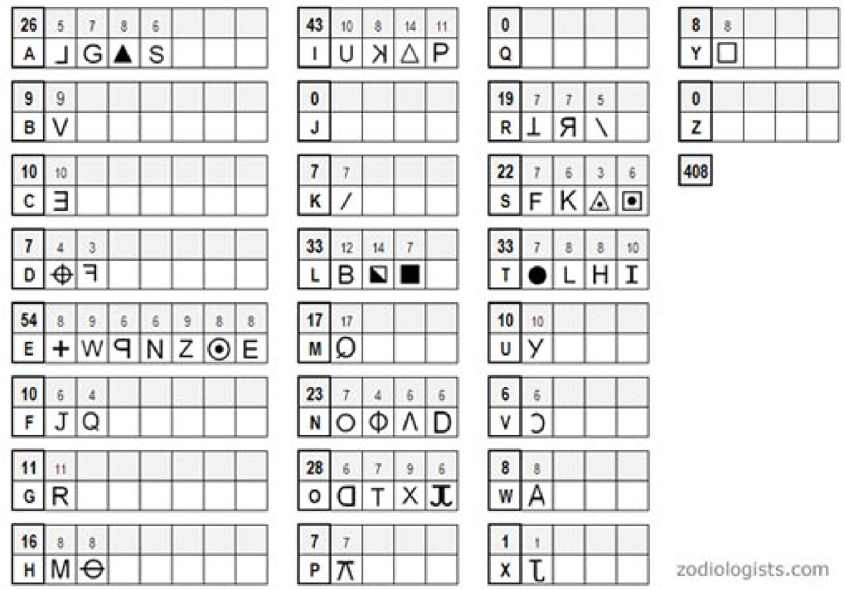
Figure 2: Variation in 1-to-1 accuracy with increasing iterations for 10 EM runs from different random starting points.

(Johnson 2007)

Zodiac 340 Cipher

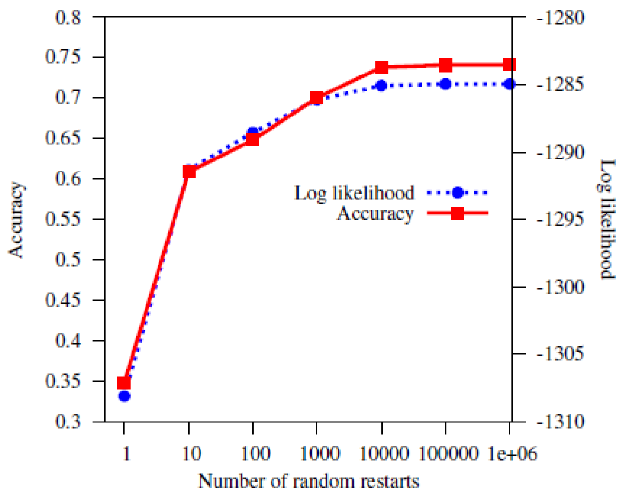


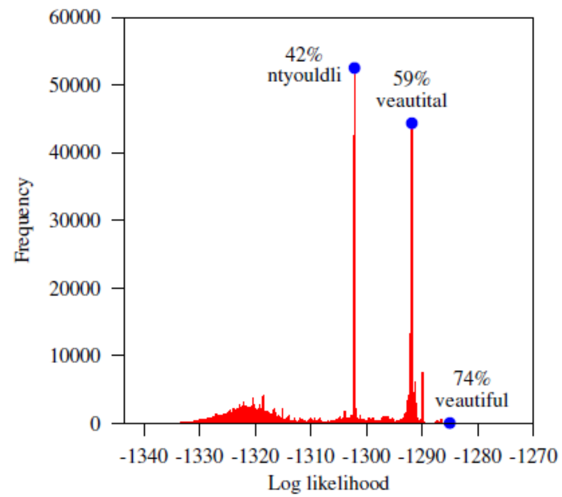
Answer something like this??

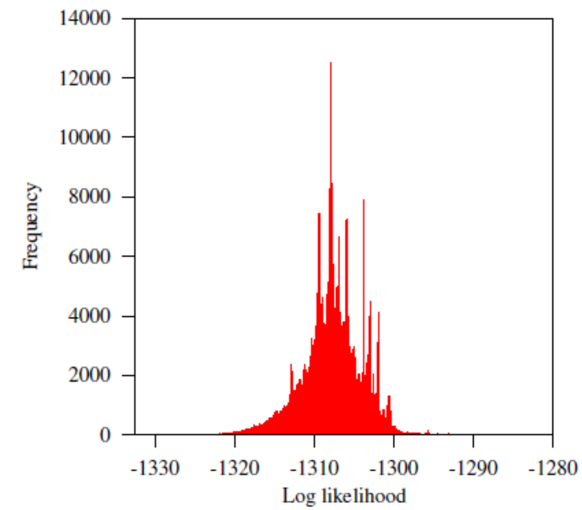


[Berg-Kirkpatrick & Klein, 2013)

Zodiac 340 Cipher







Synthetic cipher
(answer is known)

Actual cipher
(answer unknown)

(Berg-Kirkpatrick & Klein, 2013)

Bayesian Inference

knight

everything i say

is said in a workbook called
“Bayesian Inference with Tears”

available on my web page

Generative Modeling is Great, EM is Great

- They let us do unsupervised training when observed data is incomplete
  - translation dictionaries from bilingual data
  - grammar from raw text
  - plaintext from intercepted ciphertext
  - linguistic discovery … ?
- "How did this observed data get here?"
- Build a model, train parameters with EM

- But it's not the last word…

Unsupervised Word Segmentation

- Observed data:
  - shakespeareoncesaidtobeornottobethatisthequestion + 100k more characters
- Desired data completion:
  - shakespeare once said to be or not to be that is the question + 100k more
- Generative story:
  1. generate word w with probability P(w), print out characters in w (no space)
  2. with probability P(STOP), quit ... otherwise repeat
- Model:
  - $P(\text{observed}) = \text{sum}_w\, P(w) * P(\text{observed} \mid w)$
    $= \text{sum}_w\, P(w)$ for all w producing observed
    $P(w) = P(w_1) * 0.9 * P(w_2) * 0.9\ \ldots * P(w_n) * 0.1$ supposing P(STOP) = 0.1
- Objective:
  - maximize P(observed)

Smaller Problem

- Observed data:
  - “abcabc”
- Desired data completion (w):
  - what do you think?
- Model:
  - $P(\text{observed}) = \text{sum}_w\, P(w) * P(\text{observed} \mid w)$

    $= \text{sum}_w\, P(w)$ for all w producing observed

  $P(w) = P(w_1) * 0.9 * P(w_2) * 0.9 \ldots * P(w_n) * 0.1$
- Objective:
  - maximize P(observed)
  - parameters to play with: P(a), P(b), P(c), P(ab), P(abca), ... P(abcabc)

Observed data: “abcabc”

Data completion	Fitted word probabilities	Score(Data completion)	
a b c a b c	P(a) = 1/3, P(b) = 1/3, P(c) = 1/3	1/3 * 0.9 * 1/3 * 0.9 * 1/3 * 0.9 * 1/3 * 0.9 * 1/3 * 0.9 * 1/3 * 0.1	0.00008
a bca bc	P(a) = 1/3, P(bca) = 1/3, P(bc) = 1/3	1/3 * 0.9 * 1/3 * 0.9 * 1/3 * 0.1	0.003
a bc a bc	P(a) = 1/2, P(bc) = 1/2	1/2 * 0.9 * 1/2 * 0.9 * 1/2 * 0.9 * 1/2 * 0.1	0.004
abc abc	P(abc) = 1	1 * 0.9 * 1 * 0.1	0.090
abcabc	P(abcabc) = 1	1 * 0.1	**0.100**

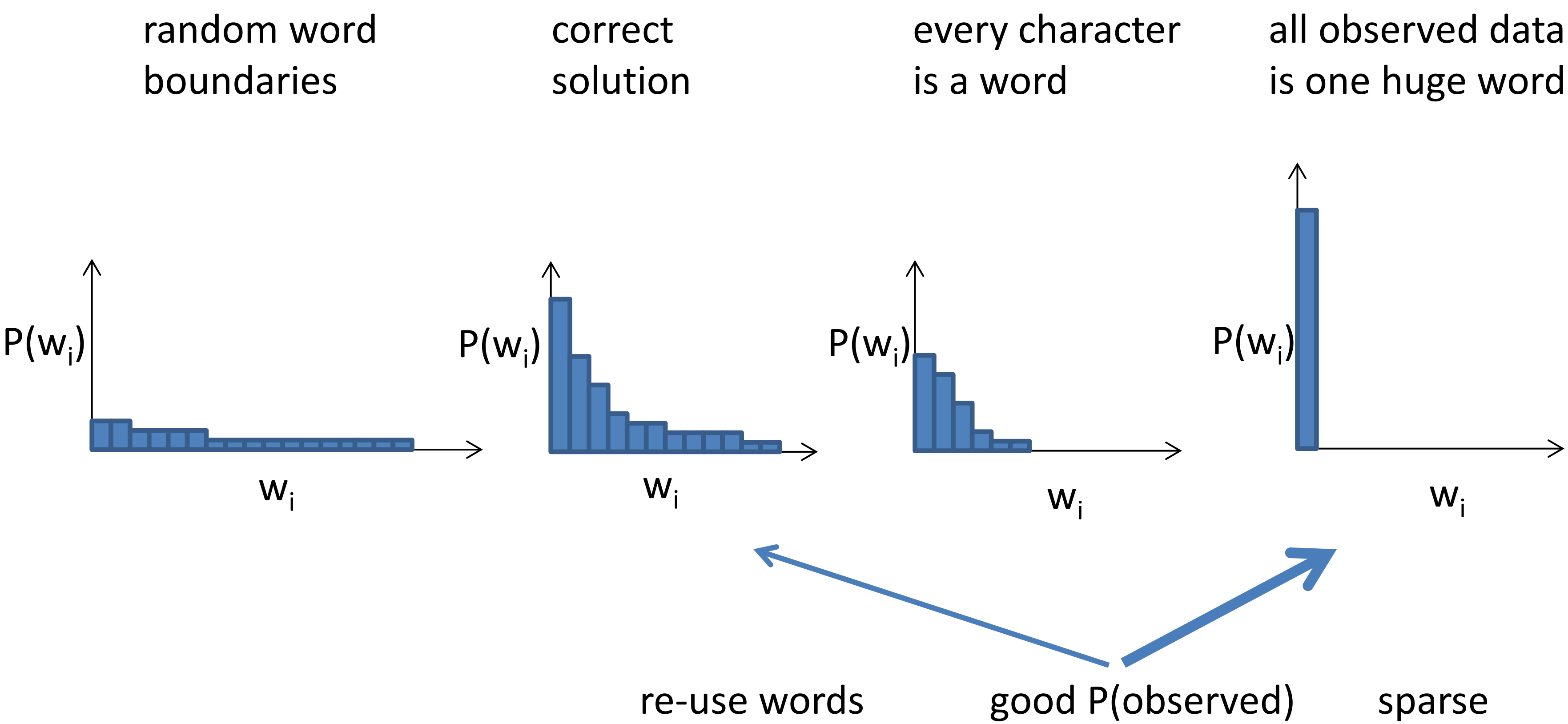
EM will probably pick P(abcabc) = 1, to maximize P(observed)

Remember that P(observed) is strictly the sum over all data completions

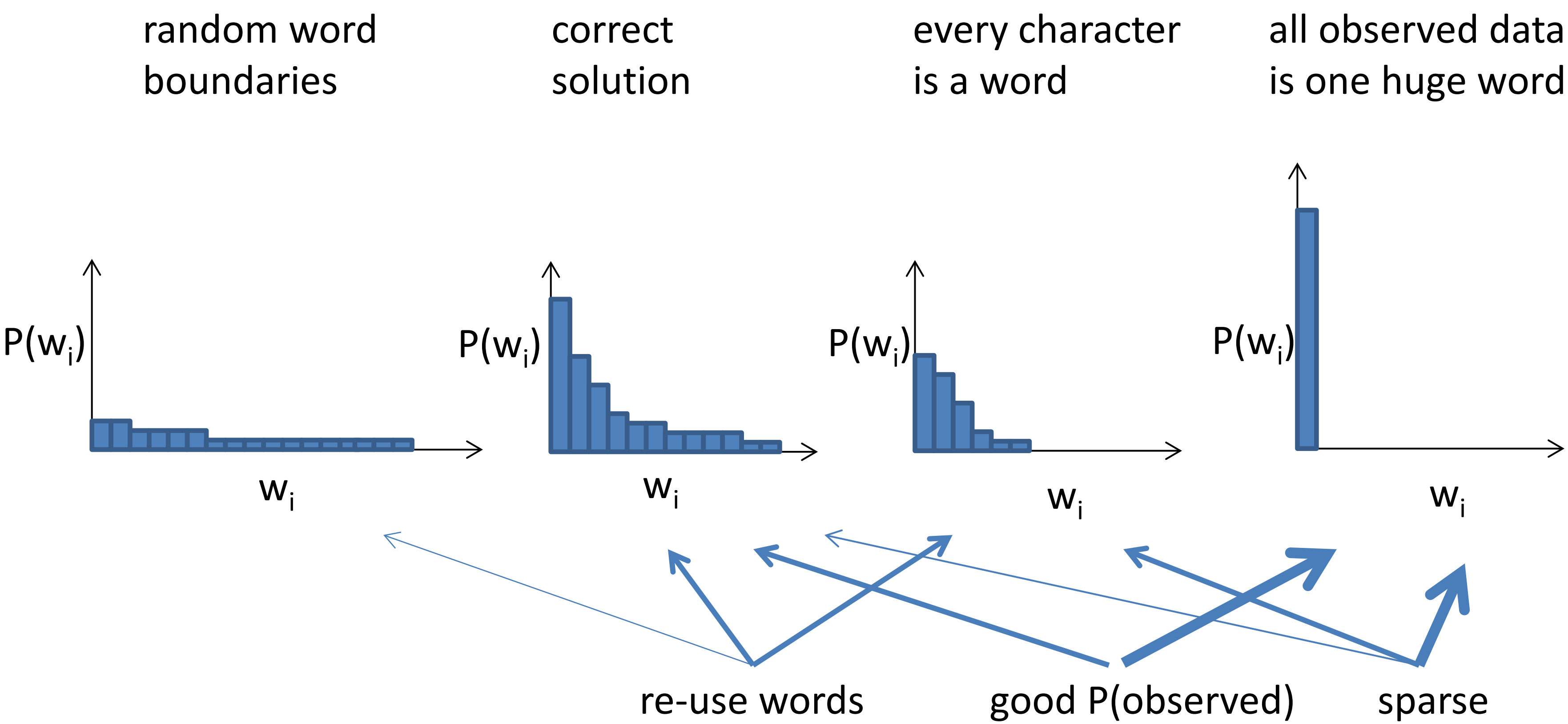
Big versus Small

- This happens whenever EM is comparing:
  - many small derivation steps
  - few large derivation steps
- EM prefers a few large steps:
  - fewer factors to multiply
- Frequent problem in practice
- We luckily didn’t have to worry about it with part-of-speech tagging, cryptanalysis, word alignment ...
- But EM still produces probabilities that are too flat compared to results of supervised training (not “sparse”)
- **Re-use** is important …

Criteria


random word boundaries
correct solution
every character is a word
all observed data is one huge word
P(w_i)
w_i
P(w_i)
w_i
P(w_i)
w_i
P(w_i)
w_i
re-use words
good P(observed)
sparse

Criteria


random word boundaries
correct solution
every character is a word
all observed data is one huge word
P(w_i)
w_i
P(w_i)
w_i
P(w_i)
w_i
P(w_i)
w_i
re-use words
good P(observed)
sparse

Solution: a New Generative Story

- Old story
  - generate word w with probability P(w), print out chars in w
  - with probability P(STOP), quit ... otherwise repeat

- New story
  - with probability β, generate word w with probability $P_0(w)$ ("base probability" -- let's say uniform)

    with probability 1-β, generate word w with probability proportional to how often we've picked it so far ("cache")
  - print out chars in w
  - with probability P(STOP), quit ... otherwise repeat

always use base probability for first word.
after that, encourage re-use = "rich get richer".

Solution: a New Generative Story

- Old story
  - generate word w with probability P(w), print out chars in w
  - with probability P(STOP), quit … otl… epeat

Everything else we do today will just be the logical consequence of the new story.

- New story
  - with p… with probability $P_0(w)$ ("base … let's say uniform)

    with probability 1-β, generate word w with probability proportional to how often we've picked it so far ("cache")
  - print out chars in w
  - with probability P(STOP), quit ... otherwise repeat

always use base probability for first word.
after that, encourage re-use = "rich get richer".

Base Probability is Fixed, Not Learned

- Could be uniform, e.g.:
  - P(a) = 1/516000
  - P(b) = 1/516000
  - P(abb) = 1/516000
  - P(abcabc) = 1/516000

    ....

- Or could have a micro-generative story:
  1. generate character a, b, or c with probability 1/3
  2. with p = 0.1, quit ... otherwise repeat

- So probability of word “bc” is “1/3 * 0.9 * 1/3 * 0.1”

Sample Derivations of “abcabc”

a bc abc

Sample Derivations of “abcabc”

β *
1/3 * 0.1

a

$P_0(a)$

Sample Derivations of "abcabc"

β *
1/3 * 0.1

β *
1/3 * 0.9 *
1/3 * 0.1

a

bc

$P_0(a)$

$\beta * P_0(bc) +$
$(1-\beta) * P_{cache}(bc)$

0

Sample Derivations of "abcabc"

β *
1/3 * 0.1

a

$P_0(a)$

β *
1/3 * 0.9 *
1/3 * 0.1

bc

$\beta * P_0(bc) +$
$(1-\beta) * P_{cache}(bc)$

0

β *
1/3 * 0.9 *
1/3 * 0.9 *
1/3 * 0.1

abc

$\beta * P_0(abc) +$
$(1-\beta) * P_{cache}(abc)$

0

Sample Derivations of “abcabc”

abc

abc

Sample Derivations of “abcabc”

β *
1/3 * 0.9 *
1/3 * 0.9 *
1/3 * 0.1

abc

$P_0(abc)$

Sample Derivations of "abcabc"

β *
1/3 * 0.9 *
1/3 * 0.9 *
1/3 * 0.1

abc

$P_0(abc)$

abc

β *
1/3 * 0.9 *
1/3 * 0.9 *
1/3 * 0.1

$\beta * P_0(abc) +$
$(1-\beta) * P_{cache}(abc)$

$(1-\beta) * 1$

Sample Derivations of “abcabc”

β *
1/3 * 0.9 *
1/3 * 0.9 *
1/3 * 0.9 *
1/3 * 0.9 *
1/3 * 0.9 *
1/3 * 0.1

abcabc

P_0(abcabc)

no re-use
of cache

doesn’t exploit
any patterns

Sample Derivations of "abcabc"

β *
$1/3 * 0.1$
a
$P_0(a)$

β *
$1/3 * 0.1$
b
$\beta * P_0(b) +$
$(1-\beta) * P_{cache}(b)$
0

β *
$1/3 * 0.1$
c
$\beta * P_0(c) +$
$(1-\beta) * P_{cache}(c)$
0

β *
$1/3 * 0.1$
a
$\beta * P_0(a) +$
$(1-\beta) * P_{cache}(a)$
1/3

β *
$1/3 * 0.1$
b
$\beta * P_0(b) +$
$(1-\beta) * P_{cache}(b)$
1/4

β *
$1/3 * 0.1$
c
$\beta * P_0(c) +$
$(1-\beta) * P_{cache}(c)$
1/5

$= \beta * 1/3 * 0.1 * \beta * 1/3 * 0.1 * \beta * 1/3 * 0.1 *$
$(\beta * 1/3 * 0.1 + 1/3) *$
$(\beta * 1/3 * 0.1 + 1/4) *$
$(\beta * 1/3 * 0.1 + 1/5)$

if $\beta = 0.5$, then $\beta * 1/3 * 0.1 = 0.01665$
P(derivation) = 0.000000093

Observed data: “abcabc”

Data completion	Score(Data completion) Old model, with fitted parameters	Score(Data completion) New cache model β = 0.5	Score(Data completion) New cache model β = 0.1
a bc abc	0.003	0.000000125	0.000000001
abc abc	0.090	**0.00075**	**0.00027**
abcabc	**0.100**	0.00004	0.0000081
a b c a b c	0.00008	0.000000093	

β terms reward re-use

Slight Adjustment

- As cache increases, we trust it more
- H = cache size
- Adjusted story:
  - with probability $\alpha/(\alpha+H)$, use base probability

    with probability $1-(\alpha/(\alpha+H))$, use cache
  - print out characters in w
  - with probability P(STOP), quit ... otherwise repeat

Cache Model Formula

score(derivation) =

$$\prod_{w \text{ in derivation}} \alpha/(\alpha+H) * P_0(w) + (1-(\alpha/(\alpha+H))) * \text{cache-count}(w) / H \quad =$$

H's cancel out

$$\boxed{\prod_{w \text{ in derivation}} \frac{\alpha * P_0(w) + \text{cache-count}(w)}{\alpha + H}}$$

btw, if $P_0(w) = 1 / |W|$,
we could set B to $\alpha / |W|$

$$\prod_{w \text{ in derivation}} \frac{B + \text{cache-count}(w)}{B * |W| + H}$$

Done?!

- We can now Score(Data Completion) for any data completion.
  - enumerate all data completions (aka “derivations”)
  - pick the Viterbi derivation
  - collect counts from Viterbi derivation
  - or, assign P(derivation) to each derivation
  - collect fractional counts from all derivations
  - normalize counts into probabilities
- We don’t need EM, iterations, etc.
- Very weird ...

An Aside: Add-one Smoothing

- Given a POS tag sequence of length N:
  - DT NN NN VBZ DT JJ NN IN DT NN …
- What's the chance the next tag will be VBG, if we haven't seen it yet? (Assume 42 total tags)
- Old story:
  - $P(VBG) = count(VBG) / N = \boxed{0 / N}$
- New story:

$$P(VBG) = \frac{\alpha * P_0(VBG) + \text{cache-count}(VBG)}{\alpha + N}$$

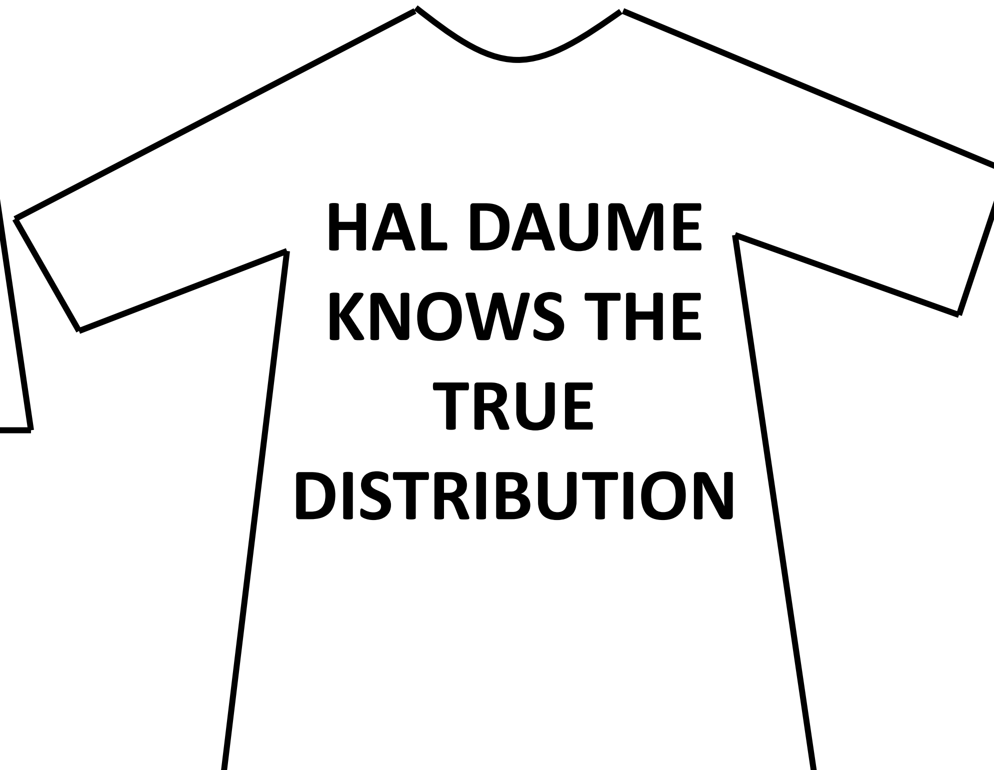
Suppose 42 distinct tags, $P_0(tag) = 1/42$, and $\alpha=42$.
Then:

$$P(VBG) = (42 * 1/42 + 0) / (42 + N) = \boxed{1 / (N+42)}$$

T-Shirt

T-Shirt


HAL DAUME
DOES NOT
SMOOTH


HAL DAUME
KNOWS THE
TRUE
DISTRIBUTION

Part-of-Speech Tagging

- Given word sequence w and word/tag dictionary, learn P(t | t) and P(w | t), then use Viterbi
- EM with bigram tag model gets ~81% accuracy
- EM's learned model doesn't **re-use** as much as human taggers do
  – there are more distinct bigrams in the Viterbi tagging than you see in human tagging
  – EM has more non-zero parameters than supervised learning
- EM with trigram tag model only gets ~75%!
  – even more non-zero parameters ...
- If we encourage **re-use**, we can get 81% → 85%

Scale is a Problem

- enumerate all derivations
- assign a score to each derivation, using cache model
- compute P(derivation)
- collect fractional counts from all derivations, weighted P(derivation)
- normalize counts into probabilities
- Viterbi tag

… if we can do this efficiently, we are done!

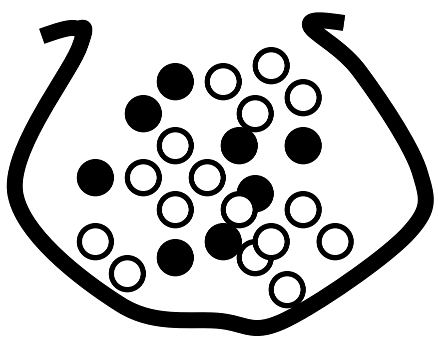
Sampling

Suppose:

P(derivation1) = 1/3 ●

P(derivation2) = 2/3 ○

Instead of fractional counting:



pull derivation

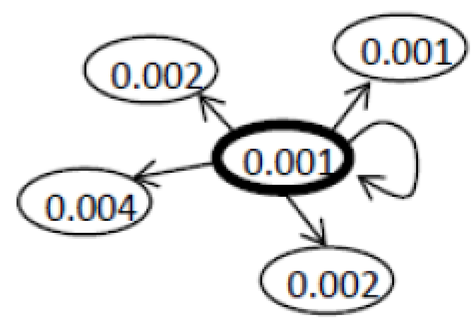
collect whole counts

put it back

Gibbs sampling

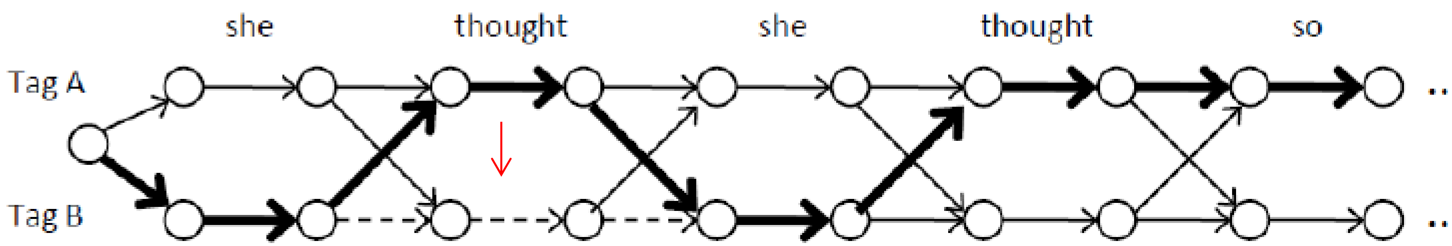
start with random derivation

change current sample in some small way. select which “way” in proportion to P(derivation):



repeat 100,000 times

Small Change Operator



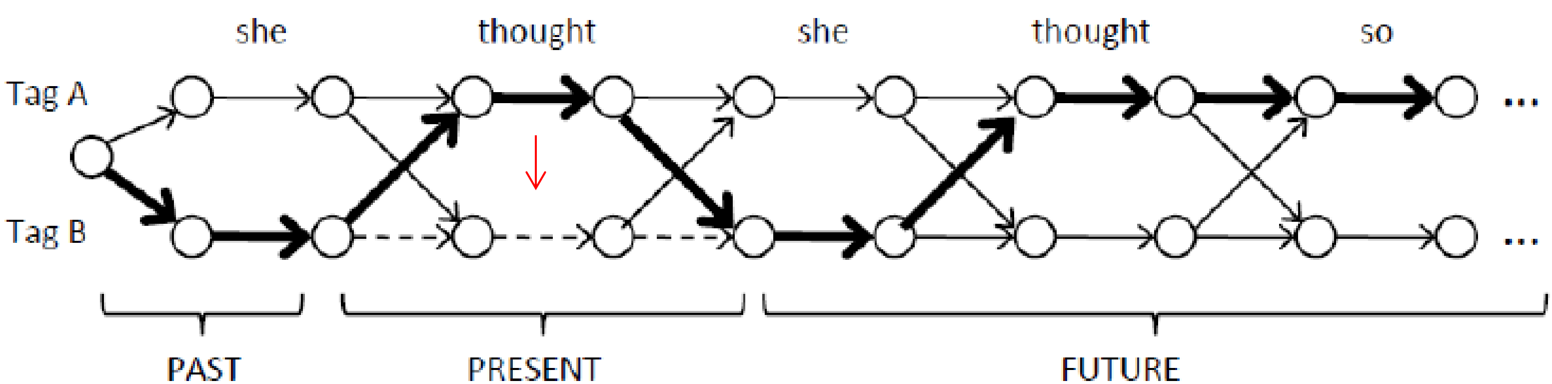
P(derivation) = P(B | START) * P(she | B) * P(A | B) * P(thought | A) * P(B | A) * P(she | B) *
P(A | B) * P(thought | A) * P(A | A) * P(so | A) ...

Incremental scoring allows us to evaluate small changes quickly.
Old (non-cache) model example:

$$P(\text{new-derivation}) = P(\text{current-derivation}) * \frac{P(B \mid B)}{P(A \mid B)} * \frac{P(\text{thought} \mid B)}{P(\text{thought} \mid A)} * \frac{P(B \mid B)}{P(B \mid A)}$$

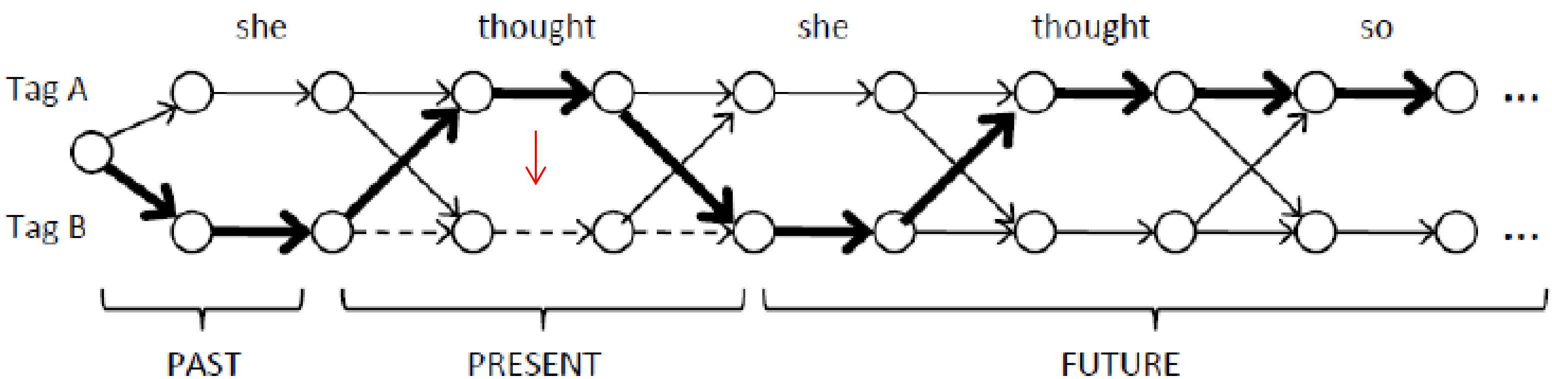
Cache Model

$$P(t, w) = \prod_{i=1}^{n} \frac{\beta_1 + \text{count}(\text{“}t_{i-1}\, t_i\text{” in cache})}{\beta_1 T + \text{count}(\text{“}t_{i-1}\text{” in cache})} * \prod_{i=1}^{n} \frac{\beta_2 + \text{count}(\text{“}t_i\ w_i\text{” in cache})}{\beta_2 W + \text{count}(\text{“}t_i\text{” in cache})}$$



Cache Model

$$P(t, w) = \prod_{i=1}^{n} \frac{\beta_1 + \text{count}(\text{“}t_{i-1}\, t_i\text{” in cache})}{\beta_1 T + \text{count}(\text{“}t_{i-1}\text{” in cache})} * \boxed{\prod_{i=1}^{n} \frac{\beta_2 + \text{count}(\text{“}t_i\ w_i\text{” in cache})}{\beta_2 W + \text{count}(\text{“}t_i\text{” in cache})}}$$

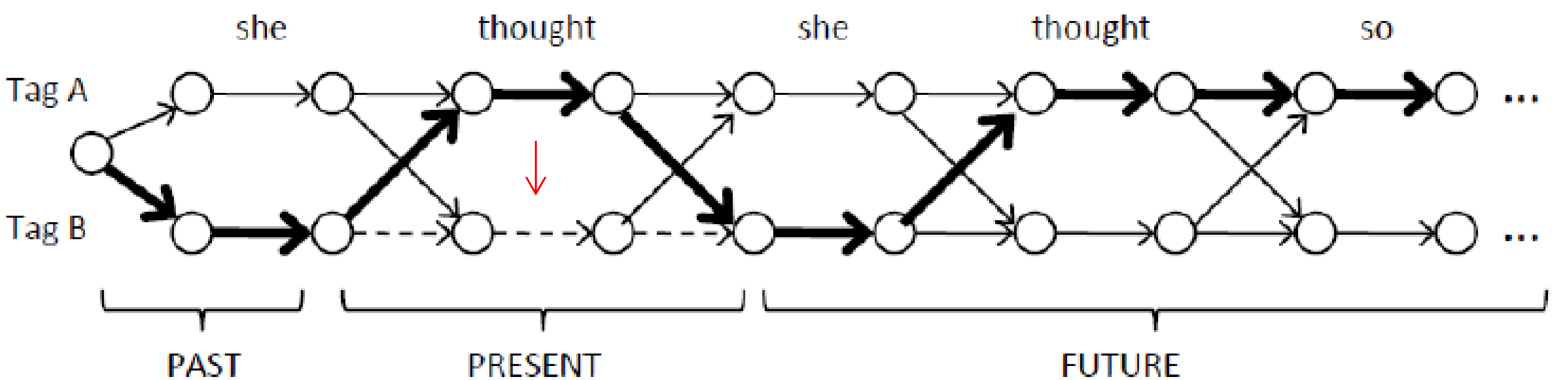


PAST	PRESENT	FUTURE
(B→she)	(A→thought)	(B→she) (A→thought) (A→so)

The portion of P(derivation) contributed by these events is:

$$\prod_{i=1}^{n} \frac{\beta_2 + \text{count}(\text{“}t_i/w_i\text{” in cache})}{\beta_2 W + \text{count}(\text{“}t_i\text{” in cache})} = \frac{\beta_2 + 0}{\beta_2 W + 0} * \frac{\beta_2 + 0}{\beta_2 W + 0} * \frac{\beta_2 + 1}{\beta_2 W + 1} * \frac{\beta_2 + 1}{\beta_2 W + 1} * \frac{\beta_2 + 0}{\beta_2 W + 2}$$

Key Incremental Scoring Trick



Now for the big trick. Let's move (exchange) the PRESENT to the very end of the sequence:

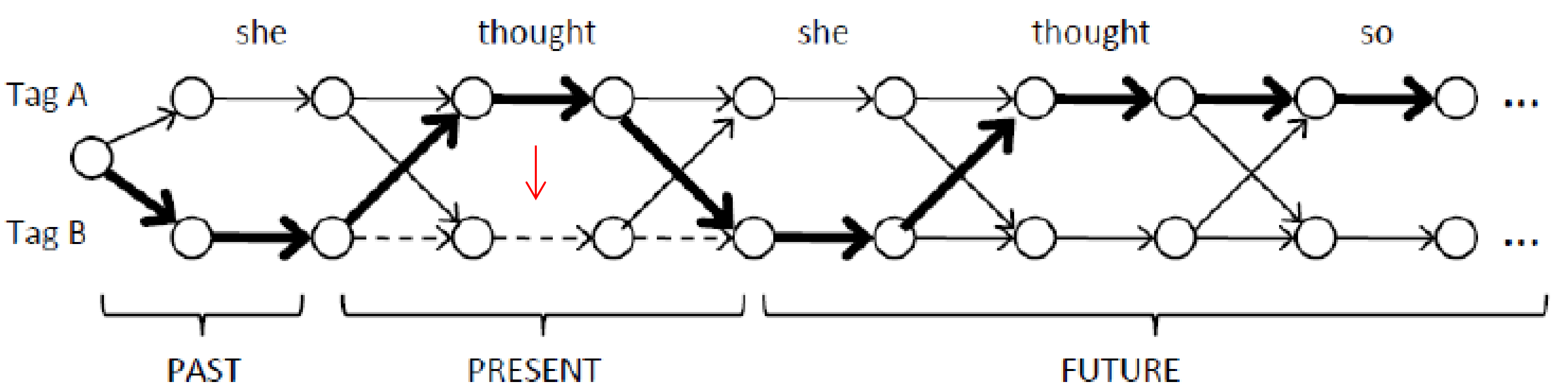
PAST	+	FUTURE			|	PRESENT
(B→she)		(B→she)	(A→thought)	(A→so)	|	(A→thought)

same numerators and denominators!

It turns out that this doesn't change the score. The new score is:

$$\prod_{i=1}^{n} \frac{\beta_2 + \text{count}(\text{"}t_i / w_i\text{" in cache})}{\beta_2 W + \text{count}(\text{"}t_i\text{" in cache})} = \frac{\beta_2 + 0}{\beta_2 W + 0} * \frac{\beta_2 + 1}{\beta_2 W + 1} * \frac{\beta_2 + 0}{\beta_2 W + 0} * \frac{\beta_2 + 0}{\beta_2 W + 1} * \frac{\beta_2 + 1}{\beta_2 W + 2}$$

Key Incremental Scoring Trick



Now for the big trick. Let's move (exchange) the PRESENT to the very end of the sequence:

PAST	+	FUTURE			|	PRESENT
(B→she)		(B→she)	(A→thought)	(A→so)	|	(A→thought)

to re-tag,
replace "+ 1"
with "+ 0"

It turns out that this doesn't change the score. The new score is:

$$\prod_{i=1}^{n} \frac{\beta_2 + \text{count}(\text{“}t_i/w_i\text{” in cache})}{\beta_2 W + \text{count}(\text{“}t_i\text{” in cache})} = \frac{\beta_2+0}{\beta_2 W+0} * \frac{\beta_2+1}{\beta_2 W+1} * \frac{\beta_2+0}{\beta_2 W+0} * \frac{\beta_2+0}{\beta_2 W+1} * \frac{\beta_2+1}{\beta_2 W+2}$$

Does It Work?

Recall plain model + EM training gets ~81%

Cache model + Gibbs sampling:

β_1 β_2 →	.001	.003	.01	.03	.1	.3	1.0	3.0	10.0
.001	84.71%	84.62%	84.54%	84.73%	84.54%	84.25%	83.89%	83.40%	84.64%
.003	84.90%	84.67%	84.75%	84.64%	84.51%	84.09%	83.80%	83.38%	84.67%
.01	84.70%	84.69%	84.36%	84.46%	84.46%	84.07%	83.84%	83.34%	84.68%
.03	84.99%	84.93%	84.71%	84.69%	84.79%	84.42%	84.14%	83.25%	84.30%
.1	84.68%	84.67%	84.85%	84.79%	84.99%	84.93%	84.55%	83.55%	82.72%
.3	84.19%	84.20%	84.28%	84.13%	**85.88%**	85.72%	85.09%	84.73%	83.08%
1.0	83.59%	83.70%	83.89%	83.72%	83.71%	83.52%	82.68%	84.41%	84.02%
3.0	83.27%	82.67%	82.05%	82.36%	82.59%	82.70%	82.45%	82.28%	84.76%
10.0	81.60%	81.73%	81.96%	82.39%	82.37%	82.43%	82.13%	83.12%	83.92%

Summary

- Mediates big step vs. small step problem
- Keeps system from memorizing data by setting P(observed) = 1.0
- Encourages sparser models
- Higher accuracy on lots of unsupervised tasks

- Cache model is called “Chinese restaurant process” in the literature, which uses lots of calculus to arrive at the same algorithm

- Yes, Carmel can do it:
  – see "Bayesian Inference for Finite-State Transducers", (D. Chiang, J. Graehl, K. Knight, A. Pauls, and S. Ravi), Proc. NAACL, 2010.

end