CMSC 723: Computational Linguistics I — Session #9

N-Gram Language Models

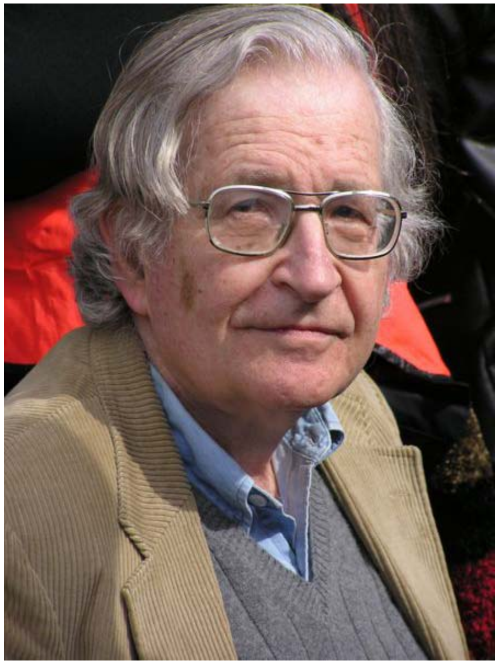
Jimmy Lin
The iSchool
University of Maryland

Wednesday, October 28, 2009

N-Gram Language Models

- What?
  - LMs assign probabilities to sequences of tokens
- Why?
  - Statistical machine translation
  - Speech recognition
  - Handwriting recognition
  - Predictive text input
- How?
  - Based on previous word histories
  - n-gram = consecutive sequences of tokens

Huh?



Noam Chomsky

But it must be recognized that the notion "probability of a sentence" is an entirely useless one, under any known interpretation of this term. (1969, p. 57)

Fred Jelinek

Anytime a linguist leaves the group the recognition rate goes up. (1988)

Every time I fire a linguist…

N-Gram Language Models

N=1 (unigrams)

This is a sentence

Unigrams:
This,
is,
a,
sentence

Sentence of length *s*, how many unigrams?

N-Gram Language Models

N=2 (bigrams)

This is a sentence

Bigrams:
This is,
is a,
a sentence

Sentence of length *s*, how many bigrams?

N-Gram Language Models

N=3 (trigrams)

This is a sentence

Trigrams:
This is a,
is a sentence

Sentence of length *s*, how many trigrams?

Computing Probabilities

$$P(w_1, w_2, \ldots, w_T)$$

$$= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\ldots P(w_T|w_1, \ldots, w_{T-1})$$

[chain rule]

Is this practical?

No! Can't keep track of all possible histories of all words!

Approximating Probabilities

Basic idea: limit history to fixed number of words N (Markov Assumption)

$$P(w_k|w_1, \ldots, w_{k-1}) \approx P(w_k|w_{k-N+1}, \ldots, w_{k-1})$$

N=1: Unigram Language Model

$$P(w_k|w_1, \ldots, w_{k-1}) \approx P(w_k)$$

$$\Rightarrow P(w_1, w_2, \ldots, w_T) \approx P(w_1)P(w_2)\ldots P(w_T)$$

Relation to HMMs?

Approximating Probabilities

Basic idea: limit history to fixed number of words N (Markov Assumption)

$$P(w_k|w_1,\ldots,w_{k-1}) \approx P(w_k|w_{k-N+1},\ldots,w_{k-1})$$

N=2: Bigram Language Model

$$P(w_k|w_1,\ldots,w_{k-1}) \approx P(w_k|w_{k-1})$$

$$\Rightarrow P(w_1,w_2,\ldots,w_T) \approx P(w_1|<\mathrm{S}>)P(w_2|w_1)\ldots P(w_T|w_{T-1})$$

Relation to HMMs?

Approximating Probabilities

Basic idea: limit history to fixed number of words N (Markov Assumption)

$$P(w_k|w_1,\ldots,w_{k-1}) \approx P(w_k|w_{k-N+1},\ldots,w_{k-1})$$

N=3: Trigram Language Model

$$P(w_k|w_1,\ldots,w_{k-1}) \approx P(w_k|w_{k-2},w_{k-1})$$

$$\Rightarrow P(w_1,w_2,\ldots,w_T) \approx P(w_1|<\mathrm{S}><\mathrm{S}>)\ldots P(w_T|w_{T-2}w_{T-1})$$

Relation to HMMs?

Building N-Gram Language Models

- Use existing sentences to compute n-gram probability estimates (training)
- Terminology:
  - N = total number of words in training data (tokens)
  - V = vocabulary size or number of unique words (types)
  - $C(w_1,...,w_k)$ = frequency of n-gram $w_1, ..., w_k$ in training data
  - $P(w_1, ..., w_k)$ = probability estimate for n-gram $w_1 ... w_k$
  - $P(w_k|w_1, ..., w_{k-1})$ = conditional probability of producing w_k given the history $w_1, ... w_{k-1}$

What's the vocabulary size?

Vocabulary Size: Heaps' Law

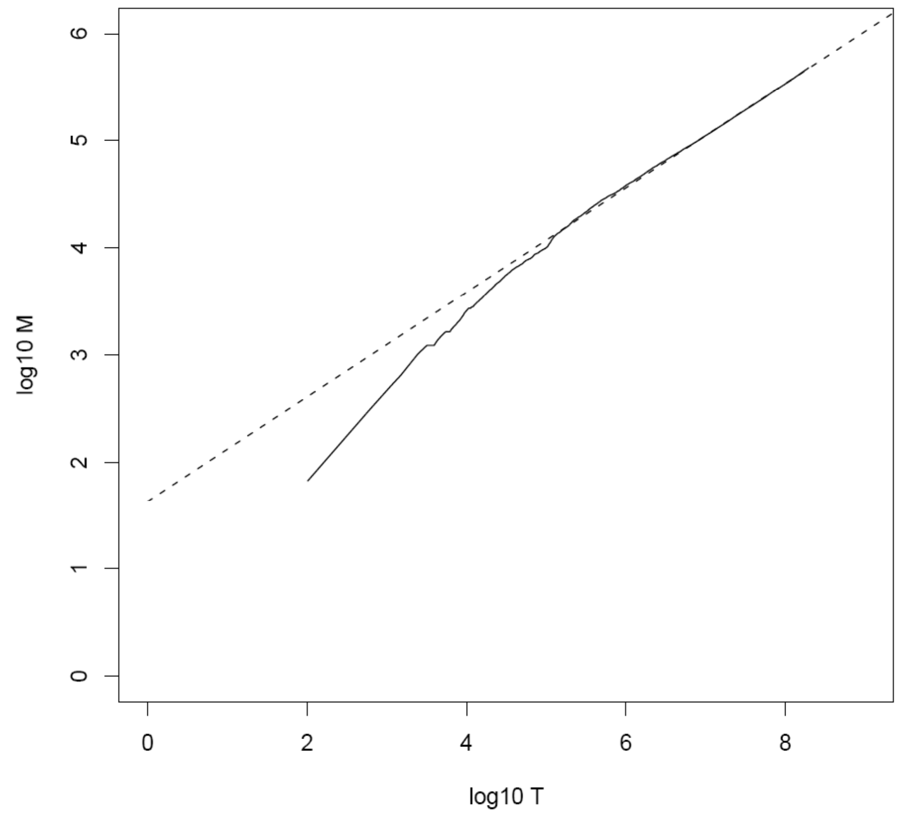
$$M = kT^b$$

M is vocabulary size
T is collection size (number of documents)
k and *b* are constants

Typically, *k* is between 30 and 100, *b* is between 0.4 and 0.6

- Heaps' Law: linear in log-log space
- Vocabulary size grows unbounded!

Heaps' Law for RCV1



k = 44
b = 0.49

First 1,000,020 terms:
Predicted = 38,323
Actual = 38,365

Reuters-RCV1 collection: 806,791 newswire documents (Aug 20, 1996-August 19, 1997)

Manning, Raghavan, Schütze, Introduction to Information Retrieval (2008)

Building N-Gram Models

- Start with what's easiest!
- Compute maximum likelihood estimates for individual n-gram probabilities
  - Unigram: $P(w_i) = \frac{C(w_i)}{N}$

    Why not just substitute *$P(w_i)$* ?

  - Bigram: $P(w_i, w_j) = \frac{C(w_i, w_j)}{N}$

    $$P(w_j | w_i) = \frac{P(w_i, w_j)}{P(w_i)} = \frac{C(w_i, w_j)}{\sum_w C(w_i, w)} = \frac{C(w_i, w_j)}{C(w_i)}$$

- Uses relative frequencies as estimates
- Maximizes the likelihood of the data given the model P(D|M)

Example: Bigram Language Model

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

Training Corpus

P(I | <s>) = 2/3 = 0.67
P(am | I) = 2/3 = 0.67
P(</s> | Sam)= 1/2 = 0.50
...

P(Sam | <s>) = 1/3 = 0.33
P(do | I) = 1/3 = 0.33
P(Sam | am) = 1/2 = 0.50

Bigram Probability Estimates

Note: We don’t ever cross sentence boundaries

Building N-Gram Models

- Start with what's easiest!
- Compute maximum likelihood estimates for individual n-gram probabilities
  - Unigram: $P(w_i) = \frac{C(w_i)}{N}$
  - Bigram: $P(w_i, w_j) = \frac{C(w_i, w_j)}{N}$

$$P(w_j|w_i) = \frac{P(w_i, w_j)}{P(w_i)} = \frac{C(w_i, w_j)}{\sum_w C(w_i, w)} = \frac{C(w_i, w_j)}{C(w_i)}$$

Let's revisit this issue…

Why not just substitute *$P(w_i)$* ?

- Uses relative frequencies as estimates
- Maximizes the likelihood of the data given the model P(D|M)

More Context, More Work

- Larger N = more context
  - Lexical co-occurrences
  - Local syntactic relations
- More context is better?
- Larger N = more complex model
  - For example, assume a vocabulary of 100,000
  - How many parameters for unigram LM? Bigram? Trigram?
- Larger N has another more serious and familiar problem!

Data Sparsity

P(I | <s>) = 2/3 = 0.67
P(am | I) = 2/3 = 0.67
P(</s> | Sam)= 1/2 = 0.50
...

P(Sam | <s>) = 1/3 = 0.33
P(do | I) = 1/3 = 0.33
P(Sam | am) = 1/2 = 0.50

Bigram Probability Estimates

P(I like ham)

= P(I | <s>) P(like | I) P(ham | like) P(</s> | ham)

= 0

Why?
Why is this bad?

Data Sparsity

- Serious problem in language modeling!
- Becomes more severe as N increases
  - What's the tradeoff?
- Solution 1: Use larger training corpora
  - Can't always work... Blame Zipf's Law (Looong tail)
- Solution 2: Assign non-zero probability to unseen n-grams
  - Known as smoothing

Smoothing

- Zeros are bad for any statistical estimator
  - Need better estimators because MLEs give us a lot of zeros
  - A distribution without zeros is “smoother”
- The Robin Hood Philosophy: Take from the rich (seen n-grams) and give to the poor (unseen n-grams)
  - And thus also called discounting
  - Critical: make sure you still have a valid probability distribution!
- Language modeling: theory vs. practice

Laplace's Law

- Simplest and oldest smoothing technique
- Just add 1 to all n-gram counts including the unseen ones
- So, what do the revised estimates look like?

Laplace's Law: Probabilities

Unigrams

$$P_{MLE}(w_i) = \frac{C(w_i)}{N} \longrightarrow P_{LAP}(w_i) = \frac{C(w_i)+1}{N+V}$$

Bigrams

$$P_{MLE}(w_i, w_j) = \frac{C(w_i, w_j)}{N} \longrightarrow P_{LAP}(w_i, w_j) = \frac{C(w_i, w_j)+1}{N+V^2}$$

Careful, don't confuse the N's!

$$P_{LAP}(w_j|w_i) = \frac{P_{LAP}(w_i, w_j)}{P_{LAP}(w_i)} = \frac{C(w_i, w_j)+1}{C(w_i)+V}$$

What if we don't know V?

Laplace's Law: Frequencies

Expected Frequency Estimates

$$
\begin{aligned}
C_{LAP}(w_i) &= P_{LAP}(w_i)N \\
C_{LAP}(w_i, w_j) &= P_{LAP}(w_i, w_j)N
\end{aligned}
$$

Relative Discount

$$
\begin{aligned}
d_1 &= \frac{C_{LAP}(w_i)}{C(w_i)} \\
d_2 &= \frac{C_{LAP}(w_i, w_j)}{C(w_i, w_j)}
\end{aligned}
$$

Laplace's Law

- Bayesian estimator with uniform priors
- Moves too much mass over to unseen n-grams
- What if we added a fraction of 1 instead?

Lidstone’s Law of Succession

- Add $0 < \gamma < 1$ to each count instead
- The smaller γ is, the lower the mass moved to the unseen n-grams (0=no smoothing)
- The case of $\gamma = 0.5$ is known as Jeffery-Perks Law or Expected Likelihood Estimation
- How to find the right value of γ?

Good-Turing Estimator

- Intuition: Use n-grams seen once to estimate n-grams never seen and so on
- Compute N_r (frequency of frequency r)

$$N_r = \sum_{w_i w_j : C(w_i w_j)} 1$$

  - N_0 is the number of items with count 0
  - N_1 is the number of items with count 1
  - …

Good-Turing Estimator

- For each *r*, compute an expected frequency estimate (smoothed count)

$$r' = C_{GT}(w_i, w_j) = (r+1)\frac{N_{r+1}}{N_r}$$

- Replace MLE counts of seen bigrams with the expected frequency estimates and use those for probabilities

$$P_{GT}(w_i, w_j) = \frac{C_{GT}(w_i, w_j)}{N} \qquad P_{GT}(w_j|w_i) = \frac{C_{GT}(w_i, w_j)}{C(w_i)}$$

Good-Turing Estimator

- What about an unseen bigram?

$$r' = C_{GT} = (0+1)\frac{N_1}{N_0} = \frac{N_1}{N_0}$$

$$P_{GT} = \frac{C_{GT}}{N}$$

- Do we know N_0? Can we compute it for bigrams?

$$N_0 = V^2 - \text{bigrams we have seen}$$

Good-Turing Estimator: Example

r	N_r
1	138741
2	25413
3	10531
4	5997
5	3565
6	...

V = 14585

Seen bigrams =199252

$$N_0 = (14585)^2 - 199252$$

$$C_{unseen} = N_1 / N_0 = 0.00065$$

$$P_{unseen} = N_1 / (N_0 N) = 1.06 \times 10^{-9}$$

Note: Assumes mass is uniformly distributed

C(person she) = 2 $\quad$ C_{GT}(person she) = (2+1)(10531/25413) = 1.243

C(person) = 223 $\quad$ P(she|person) =C_{GT}(person she)/223 = 0.0056

Good-Turing Estimator

- For each *r*, compute an expected frequency estimate (smoothed count)

$$r' = C_{GT}(w_i, w_j) = (r+1)\frac{N_{r+1}}{N_r}$$

- Replace MLE counts of seen bigrams with the expected frequency estimates and use those for probabilities

$$P_{GT}(w_i, w_j) = \frac{C_{GT}(w_i, w_j)}{N} \qquad P_{GT}(w_j | w_i) = \frac{C_{GT}(w_i, w_j)}{C(w_i)}$$

What if w_i isn't observed?

Good-Turing Estimator

- Can't replace all MLE counts
- What about r_{max}?
  - $N_{r+1} = 0$ for $r = r_{max}$
- Solution 1: Only replace counts for r < k (~10)
- Solution 2: Fit a curve S through the observed (r, N_r) values and use $S(r)$ instead
- For both solutions, remember to do what?
- Bottom line: the Good-Turing estimator is not used by itself but in combination with other techniques

Combining Estimators

- Better models come from:
  - Combining n-gram probability estimates from different models
  - Leveraging different sources of information for prediction
- Three major combination techniques:
  - Simple Linear Interpolation of MLEs
  - Katz Backoff
  - Kneser-Ney Smoothing

Linear MLE Interpolation

- Mix a trigram model with bigram and unigram models to offset sparsity
- Mix = Weighted Linear Combination

$$P(w_k|w_{k-2}w_{k-1}) =$$

$$\lambda_1 P(w_k|w_{k-2}w_{k-1}) + \lambda_2 P(w_k|w_{k-1}) + \lambda_3 P(w_k)$$

$$0 <= \lambda_i <= 1 \qquad \sum_i \lambda_i = 1$$

Linear MLE Interpolation

- λ_i are estimated on some held-out data set (not training, not test)
- Estimation is usually done via an EM variant or other numerical algorithms (e.g. Powell)

Backoff Models

- Consult different models in order depending on specificity (instead of all at the same time)
- The most detailed model for current context first and, if that doesn’t work, back off to a lower model
- Continue backing off until you reach a model that has some counts

Backoff Models

- Important: need to incorporate discounting as an integral part of the algorithm… Why?
- MLE estimates are well-formed…
- But, if we back off to a lower order model without taking something from the higher order MLEs, we are adding extra mass!
- Katz backoff
  - Starting point: GT estimator assumes uniform distribution over unseen events… can we do better?
  - Use lower order models!

Katz Backoff

Given a trigram "*x y z*"

$$P_{katz}(z|x,y) = \begin{cases} P_{GT}(z|x,y), & \text{if } C(x,y,z) > 0 \\ \alpha(x,y)P_{katz}(z|y), & \text{otherwise} \end{cases}$$

$$P_{katz}(z|y) = \begin{cases} P_{GT}(z|y), & \text{if } C(y,z) > 0 \\ \alpha(y)P_{GT}(z), & \text{otherwise} \end{cases}$$

Katz Backoff (from textbook)

Given a trigram "*x y z*"

$$P_{katz}(z|x,y) = \begin{cases} P_{GT}(z|x,y), & \text{if } C(x,y,z) > 0 \\ \boxed{\alpha(x,y)P_{katz}(z|y), \quad \text{else if } C(x,y) > 0 \\ P_{GT}(z), \quad \text{otherwise}} \end{cases}$$

Typo?

$$P_{katz}(z|y) = \begin{cases} P_{GT}(z|y), & \text{if } C(y,z) > 0 \\ \alpha(y)P_{GT}(z), & \text{otherwise} \end{cases}$$

Katz Backoff

- Why use P_{GT} and not P_{MLE} directly ?
  - If we use P_{MLE} then we are adding extra probability mass when backing off!
  - Another way: Can’t save any probability mass for lower order models without discounting
- Why the α’s?
  - To ensure that total mass from all lower order models sums exactly to what we got from the discounting

Kneser-Ney Smoothing

- Observation:
  - Average Good-Turing discount for $r \geq 3$ is largely constant over r
  - So, why not simply subtract a fixed discount D (≤ 1) from non-zero counts?
- Absolute Discounting: discounted bigram model, back off to MLE unigram model
- Kneser-Ney: Interpolate discounted model with a special "continuation" unigram model

Kneser-Ney Smoothing

- Intuition
  - Lower order model important only when higher order model is sparse
  - Should be optimized to perform in such situations
- Example
  - C(Los Angeles) = C(Angeles) = M; M is very large
  - “Angeles” always and only occurs after “Los”
  - Unigram MLE for “Angeles” will be high and a normal backoff algorithm will likely pick it in any context
  - It shouldn’t, because “Angeles” occurs with only a single context in the entire training data

Kneser-Ney Smoothing

- Kneser-Ney: Interpolate discounted model with a special "continuation" unigram model
  - Based on appearance of unigrams in different contexts
  - Excellent performance, state of the art

$$P_{KN}(w_k|w_{k-1}) = \frac{C(w_{k-1}w_k) - D}{C(w_{k-1})} + \beta(w_k)P_{CONT}(w_k)$$

$$P_{CONT}(w_i) = \frac{N(\bullet\ w_i)}{\sum_{w'} N(\bullet\ w')}$$

$N(\bullet\ w_i)$ = number of different contexts w_i has appeared in

- Why interpolation, not backoff?

Explicitly Modeling OOV

- Fix vocabulary at some reasonable number of words
- During training:
  - Consider any words that don't occur in this list as unknown or out of vocabulary (OOV) words
  - Replace all OOVs with the special word <UNK>
  - Treat <UNK> as any other word and count and estimate probabilities
- During testing:
  - Replace unknown words with <UNK> and use LM
  - Test set characterized by OOV rate (percentage of OOVs)

Evaluating Language Models

- Information theoretic criteria used
- Most common: Perplexity assigned by the trained LM to a test set
- Perplexity: How surprised are you on average by what comes next ?
  - If the LM is good at knowing what comes next in a sentence ⇒ Low perplexity (lower is better)
  - Relation to weighted average branching factor

Computing Perplexity

- Given testset *W* with words $w_1, \ldots, w_N$
- Treat entire test set as one word sequence
- Perplexity is defined as the probability of the entire test set normalized by the number of words

$$PP(T) = P(w_1, \ldots, w_N)^{-1/N}$$

- Using the probability chain rule and (say) a bigram LM, we can write this as

$$PP(T) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

- A lot easer to do with log probs!

Practical Evaluation

- Use <s> and </s> both in probability computation
- Count </s> but not <s> in *N*
- Typical range of perplexities on English text is 50-1000
- Closed vocabulary testing yields much lower perplexities
- Testing across genres yields higher perplexities
- Can only compare perplexities if the LMs use the same vocabulary

Order	Unigram	Bigram	Trigram
PP	962	170	109

Training: N=38 million, V~20000, open vocabulary, Katz backoff where applicable
Test: 1.5 million words, same genre as training

Typical "State of the Art" LMs

- Training
  - N = 10 billion words, V = 300k words
  - 4-gram model with Kneser-Ney smoothing
- Testing
  - 25 million words, OOV rate 3.8%
  - Perplexity ~50

Take-Away Messages

- LMs assign probabilities to sequences of tokens
- N-gram language models: consider only limited histories
- Data sparsity is an issue: smoothing to the rescue
  - Variations on a theme: different techniques for redistributing probability mass
  - Important: make sure you still have a valid probability distribution!