StatML (Chapter 7): Language Models

Chunqi Shi

Brandeis University

shicq@brandeis.edu

October 3, 2014

Overview

Language Models

Why?

Language models answer the question:

How likely is a string of English words good English?

What?

A statistical language model assigns a probability to a sequence of m words $P(w_1, \ldots, w_m)$ by means of a probability distribution.

How?

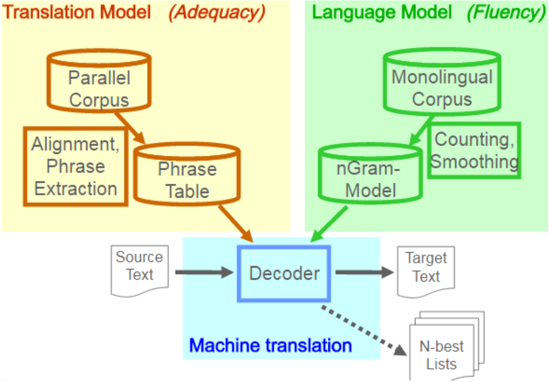
- Reordering:

  $P_{LM}(\text{the house is small}) > P_{LM}(\text{small the is house})$
- Word Choice:

  $P_{LM}(\text{I am going home}) > P_{LM}(\text{I am going house})$

Language Models & SMT Architecture

How language models work in a basic SMT architecture[1]?

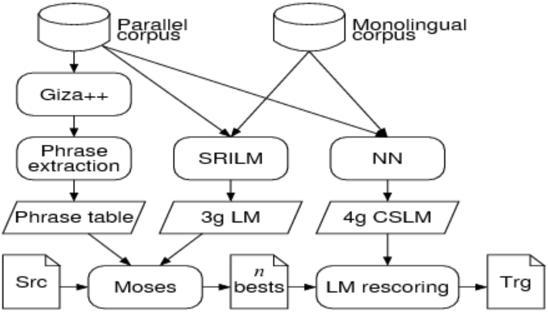


[1] http://slideplayer.us/slide/203403/

Open Source Language Models Example

Architecture of the LIMSI SMT system[2] and open language models:

- SRILM[3] (N-Gram) & NN[Neural Networks] (Continuous Space LM).
- Giza++: Translation Model.
- Moses: Decoder



[2] http://www.limsi.fr/tlp/mt/

[3] http://sourceforge.net/projects/irstlm/

Other Language Models Applications

Speech Recognition

$P_{LM}(\text{I saw a van}) > P_{LM}(\text{eyes awe of an})$

Spell Correction

The office is about fifteen minuets from my house.

$P_{LM}(\text{about fifteen minutes from}) > P_{LM}(\text{about fifteen minuets from})$

Information Retrieval

No results found for "University of Brandeis" (Query likelihood model).

$P_{LM}(\text{University of Brandeis}) > P_{LM}(\text{Brandeis University})$

More !!

Part-of-speech Tagging, Parsing, Summarization, Question-Answering, etc.

Probabilistic Language Modeling

How to Compute $P(W)$

$$P(W) = P(w_1, \ldots, w_m)$$

Probability of an upcoming word

$$P(w_k | w_1, w_2, \ldots, w_{k-1})$$

Decomposing using Chain Rule

$$P(w_1, \ldots, w_m) =$$
$$P(w_1)P(w_2|w_1)P(w_2|w_1, w_2) \ldots P(w_m|w_1, w_2, \ldots, w_{m-1})$$

Example

$$P(\text{its water is so transparent}) =$$
$$P(\text{its}) \times P(\text{water}|\text{its}) \times P(\text{is}|\text{its water}) \times P(\text{so}|\text{its water is}) \times$$
$$P(\text{transparent}|\text{its water is so})$$

Chain Rule Estimation

Joint Probability

$$P(w_1 w_2 \dots w_m) = \prod P(w_i | w_1 w_2 \dots w_i - 1)$$

How to estimate?

Maximum likelihood estimation:

$P(\text{transparent}|\text{its water is so}) =$

$$\frac{Count(\text{its water is so transparent})}{Count(\text{its water is so})}$$

Problems?

- Sparse data: NO enough data for estimating.
- Large space: HUGE possible sentences.

Markov Chain

Markov Assumption

Only previous history matters:

$P(\text{transparent}|\text{its water is so}) = (\text{transparent}|\text{so})$ or maybe

$P(\text{transparent}|\text{its water is so}) = (\text{transparent}|\text{so})$

k_{th} Order Markov Model

$$P(w_1 w_2 \ldots w_m) = \prod P(w_i | w_{i-k} w_2 \ldots w_i - 1)$$

Simple Cases

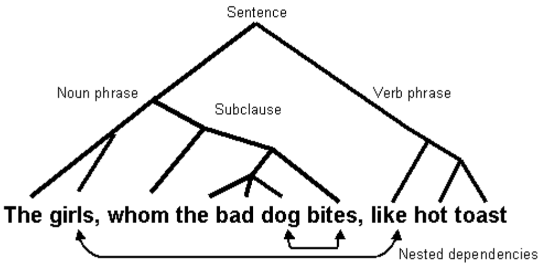
Unigram model: $P(w_1 w_2 \ldots w_m) = \prod P(w_i)$

Bigram model: $P(w_1 w_2 \ldots w_m) = \prod P(w_i | w_{i-1})$

N-gram Models

Is Markov assumption sufficient? NO!

Language has long-distance dependencies:



Or:

"The computer which I had just put into the machine room on the fifth floor crashed."

Bigram Example

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

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

$$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67 \qquad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33 \qquad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$$

$$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5 \qquad P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5 \qquad P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$$

Trigram Example

- Counts for trigrams and estimated word probabilities

the green (total: 1748)

word	c.	prob.
paper	801	0.458
group	640	0.367
light	110	0.063
party	27	0.015
ecu	21	0.012

the red (total: 225)

word	c.	prob.
cross	123	0.547
tape	31	0.138
army	9	0.040
card	7	0.031
,	5	0.022

the blue (total: 54)

word	c.	prob.
box	16	0.296
.	6	0.111
flag	6	0.111
,	3	0.056
angel	3	0.056

- 225 trigrams in the Europarl corpus start with the red
- 123 of them end with cross

→ maximum likelihood probability is $\frac{123}{225} = 0.547$.

"The red cross" and "The green party" are frequent trigrams in the Europarl corpus.

Evaluation of N-gram Models

How good is our model?

Extrinsic Evaluation: training A & B, testing, comparing accuracy of A & B by evaluation metric.
But it is time-consuming.

Intrinsic Evaluation

Perplexity: How well can we predict the next word?
Intrinsic evaluation is Bad approximation! Unless the test data looks just like the training data.
But is helpful to think about.

Intuition of Perplexity

How hard is the task of recognizing digits "0, 1, 2, 3, 4, 5, 6, 7, 8, 9"?
Perplexity 10.

Perplexity

Cross Entropy

$$H(W) = -\frac{1}{n} \log P(w_1 w_2 \ldots w_n)$$
$$= -\frac{1}{n} \sum_{i}^{n} \log P(w_i | w_1 \ldots, w_{i-1})$$

Perplexity:

$$PP(W) = 2^{H(W)} = P(W)^{-\frac{1}{n}}$$

Perplexity as branching factor

$$PP(W) = P(1,\ 2,\ \ldots,\ 10)^{-\frac{1}{10}}$$
$$= (\frac{1}{10})^{10 \times -\frac{1}{10}} = (\frac{1}{10})^{-1} = 10$$

Comparison N-gram Models

Minimizing perplexity is the same as maximizing probability, thus better model.

word	unigram	bigram	trigram	4-gram
i	6.684	3.197	3.197	3.197
would	8.342	2.884	2.791	2.791
like	9.129	2.026	1.031	1.290
to	5.081	0.402	0.144	0.113
commend	15.487	12.335	8.794	8.633
the	3.885	1.402	1.084	0.880
rapporteur	10.840	7.319	2.763	2.350
on	6.765	4.140	4.150	1.862
his	10.678	7.316	2.367	1.978
work	9.993	4.816	3.498	2.394
.	4.896	3.020	1.785	1.510
</s>	4.828	0.005	0.000	0.000
average	8.051	4.072	2.634	2.251
perplexity	265.136	16.817	6.206	4.758

Generalization and Zeros

Unseen N-grams

Things that NOT ever occur in the training set. But occur in the test set.

Training Set:

1. ... denied the allegations
2. ... denied the reports
3. ... denied the claims
4. ... denied the request

Test Set:

1. ... denied the offer
2. ... denied the loan

$$P(\text{``offer''} \,|\, \text{''denied the''}) = 0$$

Smoothing

Sparse statistics, smoothing to generalize better.

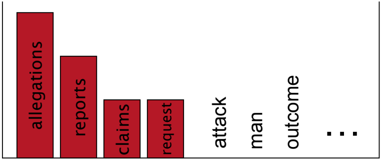
Smoothing

How to smooth all words non-zeros

- When we have sparse statistics:

  P(w | denied the)
  3 allegations
  2 reports
  1 claims
  1 request

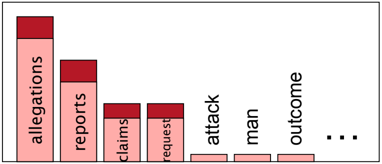
  7 total



- Steal probability mass to generalize better

  P(w | denied the)
  2.5 allegations
  1.5 reports
  0.5 claims
  0.5 request
  2 other

  7 total



Add-One Smoothing

Laplace smoothing

Pretend we saw each word one more time than we did.

- For all possible n-grams, add the count of one.

$$p = \frac{c+1}{n+v}$$

  - c = count of n-gram in corpus
  - n = count of history
  - v = vocabulary size

- But there are many more unseen n-grams than seen n-grams

- Example: Europarl 2-bigrams:
  - $86,700$ distinct words
  - $86,700^2 = 7,516,890,000$ possible bigrams
  - but only about $30,000,000$ words (and bigrams) in corpus

Bigram Add-One Smoothing

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Add-α Smoothing

Will α adjusted count a lot?

- Add $\alpha < 1$ to each count

$$p = \frac{c + \alpha}{n + \alpha v}$$

- What is a good value for α?
- Could be optimized on held-out set

Comparison of Add-α Smoothing

Bigram in Europarl corpus

Count	Adjusted count		Test count
c	$(c+1)\frac{n}{n+v^2}$	$(c+\alpha)\frac{n}{n+\alpha v^2}$	t_c
0	0.00378	0.00016	0.00016
1	0.00755	0.95725	0.46235
2	0.01133	1.91433	1.39946
3	0.01511	2.87141	2.34307
4	0.01888	3.82850	3.35202
5	0.02266	4.78558	4.35234
6	0.02644	5.74266	5.33762
8	0.03399	7.65683	7.15074
10	0.04155	9.57100	9.11927
20	0.07931	19.14183	18.95948

- Add-α smoothing with $\alpha = 0.00017$
- t_c are average counts of n-grams in test set that occurred c times in corpus

N-gram Models

Is Markov assumption sufficient? NO!

Language has long-distance dependencies:

- Add $\alpha < 1$ to each count

$$p = \frac{c + \alpha}{n + \alpha v}$$

- What is a good value for α?
- Could be optimized on held-out set

Comparison of Add-α Smoothing

Bigram in Europarl corpus

Count	Adjusted count		Test count
c	$(c+1)\frac{n}{n+v^2}$	$(c+\alpha)\frac{n}{n+\alpha v^2}$	t_c
0	0.00378	0.00016	0.00016
1	0.00755	0.95725	0.46235
2	0.01133	1.91433	1.39946
3	0.01511	2.87141	2.34307
4	0.01888	3.82850	3.35202
5	0.02266	4.78558	4.35234
6	0.02644	5.74266	5.33762
8	0.03399	7.65683	7.15074
10	0.04155	9.57100	9.11927
20	0.07931	19.14183	18.95948

- Add-α smoothing with $\alpha = 0.00017$
- t_c are average counts of n-grams in test set that occurred c times in corpus

Held-out Estimation

Long Tail of N-gram Counting

- 1,266,566 bigrams in this corpus, more than half, 753,777, occur only once.
- Zipf's Law: the frequency of any word is inversely proportional to its rank in the frequency table.

If we observe an n-gram c times in the training corpus, ow often do we expect it to see in the future (Held-out Estimation)?

$p_h(w_1 w_2 \dots w_n) = \frac{E[r]}{N_h} \sim \frac{T_r}{N_r \times N_h}$, $r = count_h(w_1 w_2 \dots w_n)$

Cross-Validation Estimation

$$E(r) = \frac{T_r^1 + T_r^2}{N_r^1 + N_r^2}$$

Deleted Estimation

Deleted Estimation: leave one part of the training corpus out for validation.

Count r	Count of counts N_r	Count in held-out T_r	Exp. count $E[r] = T_r/N_r$
0	7,515,623,434	938,504	0.00012
1	753,777	353,383	0.46900
2	170,913	239,736	1.40322
3	78,614	189,686	2.41381
4	46,769	157,485	3.36860
5	31,413	134,653	4.28820
6	22,520	122,079	5.42301
8	13,586	99,668	7.33892
10	9,106	85,666	9.41129
20	2,797	53,262	19.04992

	Adjusted count		
Count c	$(c+1)\dfrac{n}{n+v^2}$	$(c+\alpha)\dfrac{n}{n+\alpha v^2}$	Test count t_c
0	0.00378	0.00016	0.00016
1	0.00755	0.95725	0.46235
2	0.01133	1.91433	1.39946
3	0.01511	2.87141	2.34307
4	0.01888	3.82850	3.35202
5	0.02266	4.78558	4.35234
6	0.02644	5.74266	5.33762
8	0.03399	7.65683	7.15074
10	0.04155	9.57100	9.11927
20	0.07931	19.14183	18.95948

Good-Turing Smoothing Intuition

- You are fishing (a scenario from Josh Goodman), and caught:
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish
- How likely is it that next species is trout?
  - 1/18
- How likely is it that next species is new (i.e. catfish or bass)
  - Let's use our estimate of things-we-saw-once to estimate the new things.
  - 3/18 (because N_1=3)
- Assuming so, how likely is it that next species is trout?
  - Must be less than 1/18
  - How to estimate?

Good-Turing Smoothing Calculation

$$P^*_{GT}(\text{things with zero frequency}) = \frac{N_1}{N} \qquad c^* = \frac{(c+1)N_{c+1}}{N_c}$$

- Unseen (bass or catfish)
  - c = 0:
  - MLE p = 0/18 = 0
  - P^*_{GT} (unseen) = N_1/N = 3/18
- Seen once (trout)
  - c = 1
  - MLE p = 1/18
  - C*(trout) = 2 * N2/N1
    = 2 * 1/3
    = 2/3
  - P^*_{GT}(trout) = 2/3 / 18 = 1/27

Leave One Out Intuition

- Intuition from leave-one-out validation
  - Take each of the c training words out in turn
  - c training sets of size c–1, held-out of size 1
  - What fraction of held-out words are unseen in training?
    - N_1/c
  - What fraction of held-out words are seen k times in training?
    - $(k+1)N_{k+1}/c$
  - So in the future we expect $(k+1)N_{k+1}/c$ of the words to be those with training count k
  - There are N_k words with training count k
  - Each should occur with probability:
    - $(k+1)N_{k+1}/c/N_k$
  - …or expected count: $$k^* = \frac{(k+1)N_{k+1}}{N_k}$$

Good-Turing for 2-Grams in Europarl

Count	Count of counts	Adjusted count	Test count
r	N_r	r^*	t
0	7,514,941,065	0.00015	0.00016
1	1,132,844	0.46539	0.46235
2	263,611	1.40679	1.39946
3	123,615	2.38767	2.34307
4	73,788	3.33753	3.35202
5	49,254	4.36967	4.35234
6	35,869	5.32928	5.33762
8	21,693	7.43798	7.15074
10	14,880	9.31304	9.11927
20	4,546	19.54487	18.95948

adjusted count fairly accurate when compared against the test count

Derivation of Good-Turing

- A specific n-gram α occurs with (unknown) probability p in the corpus
- Assumption: all occurrences of an n-gram α are independent of each other
- Number of times α occurs in corpus follows binomial distribution

$$p(c(\alpha) = r) = b(r; N, p_i) = \binom{N}{r} p^r (1-p)^{N-r}$$

Derivation of Good-Turing (2)

- Goal of Good-Turing smoothing: compute *expected count* c^*
- Expected count can be computed with help from binomial distribution:

$$\begin{aligned} E(c^*(\alpha)) &= \sum_{r=0}^{N} r\, p(c(\alpha) = r) \\ &= \sum_{r=0}^{N} r \binom{N}{r} p^r (1-p)^{N-r} \end{aligned}$$

- Note again: p is unknown, we cannot actually compute this

Derivation of Good-Turing (3)

- Definition: expected number of n-grams that occur r times: $E_N(N_r)$
- We have s different n-grams in corpus
  - let us call them $\alpha_1, ..., \alpha_s$
  - each occurs with probability $p_1, ..., p_s$, respectively
- Given the previous formulae, we can compute

$$\begin{aligned} E_N(N_r) &= \sum_{i=1}^{s} p(c(\alpha_i) = r) \\ &= \sum_{i=1}^{s} \binom{N}{r} p_i^r (1 - p_i)^{N-r} \end{aligned}$$

- Note again: p_i is unknown, we cannot actually compute this

Derivation of Good-Turing (4)

- Reflection
  - we derived a formula to compute $E_N(N_r)$
  - we have N_r
  - for small r: $E_N(N_r) \simeq N_r$
- Ultimate goal compute expected counts c^*, given actual counts c

$$E(c^*(\alpha)|c(\alpha) = r)$$

Derivation of Good-Turing (5)

- For a particular n-gram α, we know its actual count r
- Any of the n-grams α_i may occur r times
- Probability that α is one specific α_i

$$p(\alpha = \alpha_i | c(\alpha) = r) = \frac{p(c(\alpha_i) = r)}{\sum_{j=1}^{s} p(c(\alpha_j) = r)}$$

- Expected count of this n-gram α

$$E(c^*(\alpha) | c(\alpha) = r) = \sum_{i=1}^{s} N \, p_i \, p(\alpha = \alpha_i | c(\alpha) = r)$$

Derivation of Good-Turing (6)

- Combining the last two equations:

$$
\begin{aligned}
E(c^*(\alpha)|c(\alpha) = r) &= \sum_{i=1}^{s} N\, p_i \frac{p(c(\alpha_i) = r)}{\sum_{j=1}^{s} p(c(\alpha_j) = r)} \\
&= \frac{\sum_{i=1}^{s} N\, p_i\, p(c(\alpha_i) = r)}{\sum_{j=1}^{s} p(c(\alpha_j) = r)}
\end{aligned}
$$

- We will now transform this equation to derive Good-Turing smoothing

Derivation of Good-Turing (7)

- Repeat:

$$E(c^*(\alpha)|c(\alpha) = r) \; = \frac{\sum_{i=1}^{s} N \, p_i \, p(c(\alpha_i) = r)}{\sum_{j=1}^{s} p(c(\alpha_j) = r)}$$

- Denominator is our definition of expected counts $E_N(N_r)$

Derivation of Good-Turing (8)

- Numerator:

$$
\begin{aligned}
\sum_{i=1}^{s} N\, p_i\, p(c(\alpha_i) = r) &= \sum_{i=1}^{s} N\, p_i \binom{N}{r} p_i^r (1-p_i)^{N-r} \\
&= N \, \frac{N!}{N-r!r!} p_i^{r+1} (1-p_i)^{N-r} \\
&= N \, \frac{(r+1)}{N+1} \frac{N+1!}{N-r!r+1!} p_i^{r+1} (1-p_i)^{N-r} \\
&= (r+1) \, \frac{N}{N+1} E_{N+1}(N_{r+1}) \\
&\simeq (r+1) \, E_{N+1}(N_{r+1})
\end{aligned}
$$

Derivation of Good-Turing (9)

- Using the simplifications of numerator and denominator:

$$\begin{aligned} r^* &= E(c^*(\alpha)|c(\alpha) = r) \\ &= \frac{(r+1)\ E_{N+1}(N_{r+1})}{E_N(N_r)} \\ &\simeq (r+1)\frac{N_{r+1}}{N_r} \end{aligned}$$

- QED

Interpolation & Back-off

Sparseness in a Trigram Model

1. Back to bigrams, otherwise unigram: Back-off
2. Mix that model with bigram and unigram: Interpolation

Context

1. Back-off: some times it helps to use less context condition on less context for contexts you haven't learned much about.
2. Interpolation: higher and lower order n-gram models have different strengths and weaknesses:
   - high-order n-grams are sensitive to more context, but have sparse counts.
   - low-order n-grams consider only very limited context, but have robust counts

Interpola1on

Simple interpolation

$$\hat{P}(w_n|w_{n-1}w_{n-2}) = \lambda_1 P(w_n|w_{n-1}w_{n-2}) + \lambda_2 P(w_n|w_{n-1}) + (1-\lambda_1-\lambda_2)P(w_n)$$

Recursive Interpolation

Lambdas conditional on context:

$$\hat{P}(w_n|w_{n-1}w_{n-2}) = \lambda_1(w_{n-2}^{n-1}) P(w_n|w_{n-1}w_{n-2}) + \lambda_2(w_{n-2}^{n-1}) P(w_n|w_{n-1}) + (1-\lambda_1(w_{n-2}^{n-1})-\lambda_2(w_{n-2}^{n-1}))P(w_n)$$

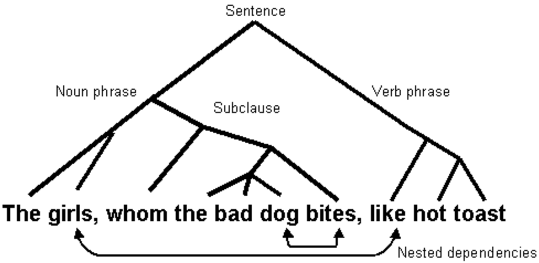
$$\sum_{w\in V}\hat{P}(w_n|w_{n-1}w_{n-2}) = \lambda_1(w_{n-2}^{n-1}) \sum_{w\in V} P(w_n|w_{n-1}w_{n-2}) + \lambda_2(w_{n-2}^{n-1}) \sum_{w\in V} P(w_n|w_{n-1}) + (1-\lambda_1(w_{n-2}^{n-1})-\lambda_2(w_{n-2}^{n-1}))\sum_{w\in V} P(w_n)$$

$$= \lambda_1(w_{n-2}^{n-1}) + \lambda_2(w_{n-2}^{n-1}) + (1-\lambda_1(w_{n-2}^{n-1})-\lambda_2(w_{n-2}^{n-1})) = 1$$

N-gram Models

Is Markov assumption sufficient? NO!

Language has long-distance dependencies:

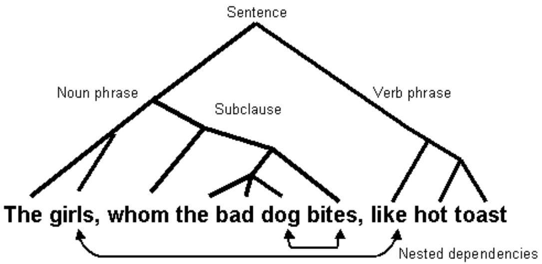


Or:
"The computer which I had just put into the machine room on the fifth floor crashed."

N-gram Models

Is Markov assumption sufficient? NO!

Language has long-distance dependencies:



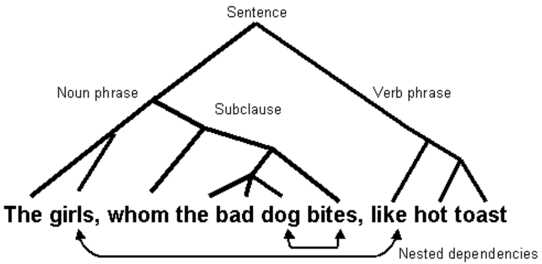
Or:

"The computer which I had just put into the machine room on the fifth floor crashed."

N-gram Models

Is Markov assumption sufficient? NO!

Language has long-distance dependencies:

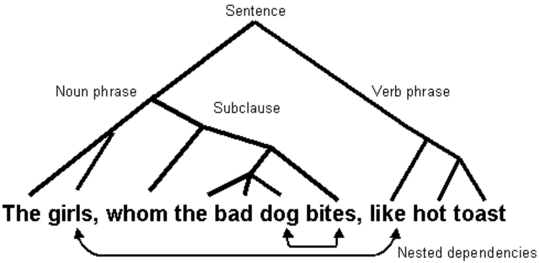


Or:
"The computer which I had just put into the machine room on the fifth floor crashed."

N-gram Models

Is Markov assumption sufficient? NO!

Language has long-distance dependencies:

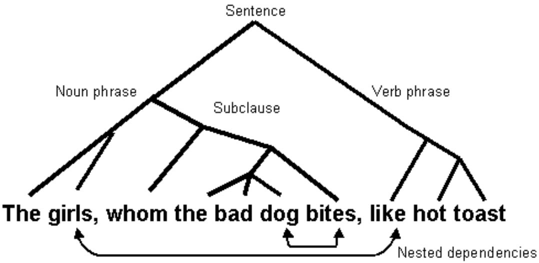


Or:
"The computer which I had just put into the machine room on the fifth floor crashed."

N-gram Models

Is Markov assumption sufficient? NO!

Language has long-distance dependencies:



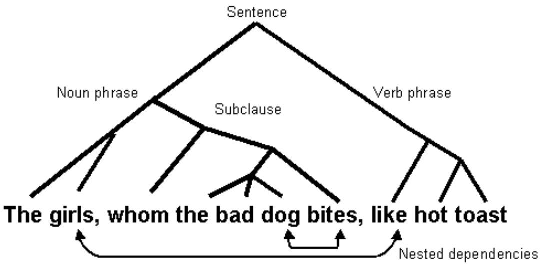
Or:

"The computer which I had just put into the machine room on the fifth floor crashed."

N-gram Models

Is Markov assumption sufficient? NO!

Language has long-distance dependencies:

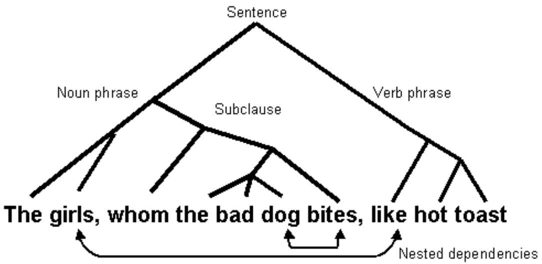


Or:
"The computer which I had just put into the machine room on the fifth floor crashed."

N-gram Models

Is Markov assumption sufficient? NO!

Language has long-distance dependencies:



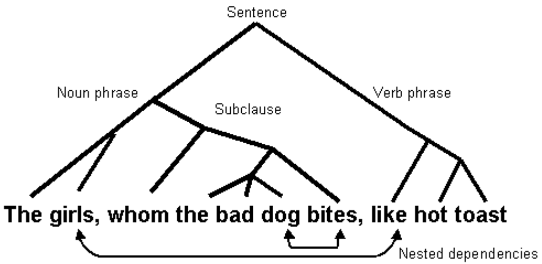
Or:

"The computer which I had just put into the machine room on the fifth floor crashed."

N-gram Models

Is Markov assumption sufficient? NO!

Language has long-distance dependencies:



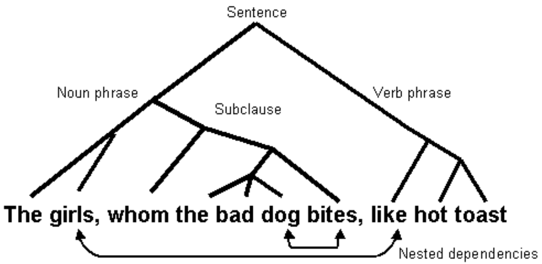
Or:

"The computer which I had just put into the machine room on the fifth floor crashed."

N-gram Models

Is Markov assumption sufficient? NO!

Language has long-distance dependencies:

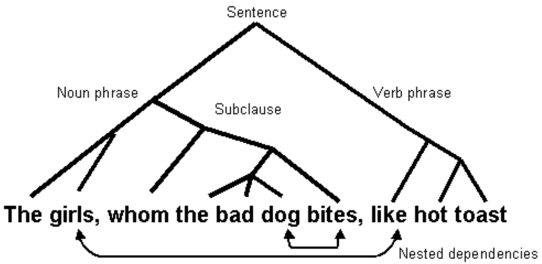


Or:
"The computer which I had just put into the machine room on the fifth floor crashed."

N-gram Models

Is Markov assumption sufficient? NO!

Language has long-distance dependencies:

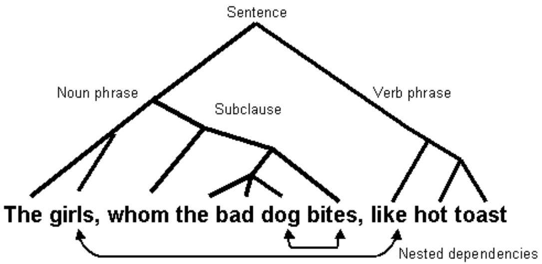


Or:
"The computer which I had just put into the machine room on the fifth floor crashed."

N-gram Models

Is Markov assumption sufficient? NO!

Language has long-distance dependencies:

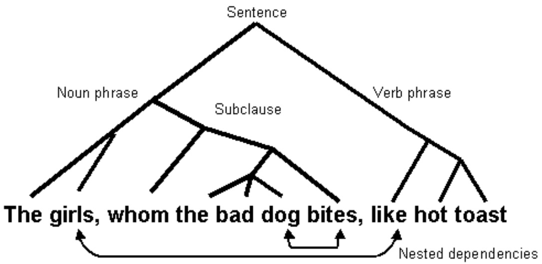


Or:
"The computer which I had just put into the machine room on the fifth floor crashed."

N-gram Models

Is Markov assumption sufficient? NO!

Language has long-distance dependencies:



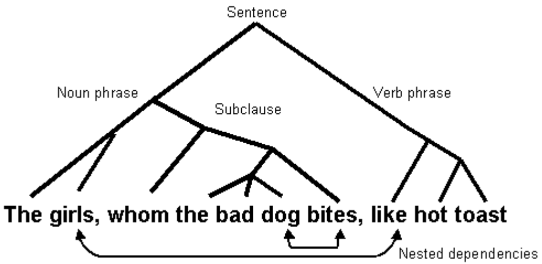
Or:

"The computer which I had just put into the machine room on the fifth floor crashed."

N-gram Models

Is Markov assumption sufficient? NO!

Language has long-distance dependencies:

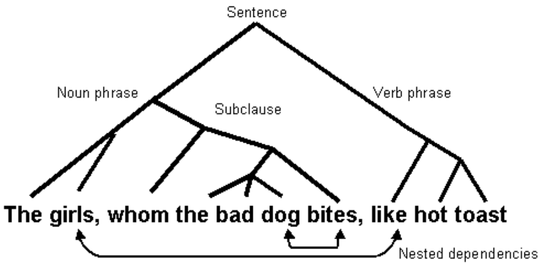


Or:
"The computer which I had just put into the machine room on the fifth floor crashed."

N-gram Models

Is Markov assumption sufficient? NO!

Language has long-distance dependencies:

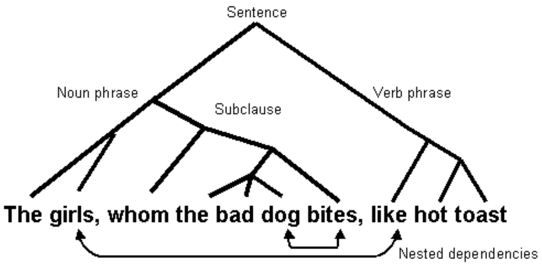


Or:
"The computer which I had just put into the machine room on the fifth floor crashed."

N-gram Models

Is Markov assumption sufficient? NO!

Language has long-distance dependencies:

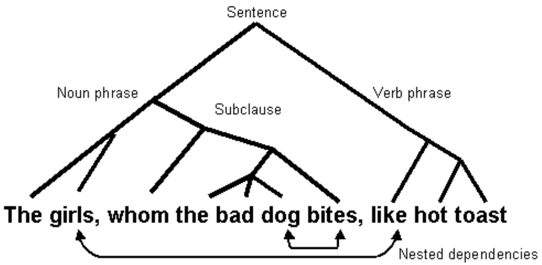


Or:
"The computer which I had just put into the machine room on the fifth floor crashed."

N-gram Models

Is Markov assumption sufficient? NO!

Language has long-distance dependencies:



Or:

"The computer which I had just put into the machine room on the fifth floor crashed."

References

Many slides are from:

- StatML book's Web site &
- Dan Jurafsky's "Language Modeling: Introduction to N-grams".

The End