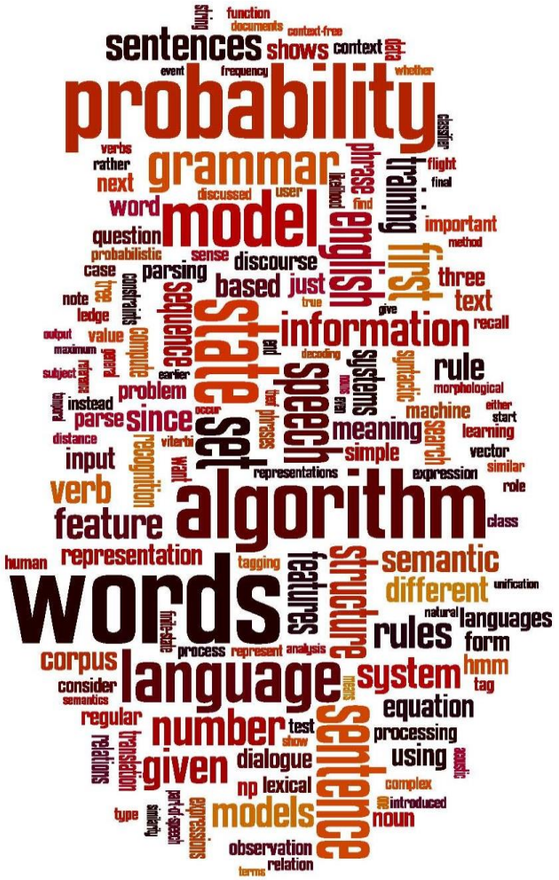
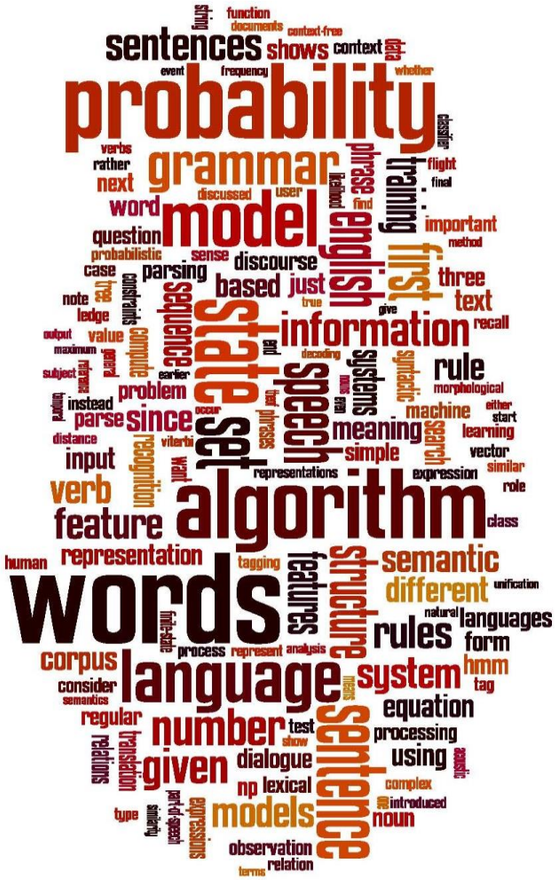


Language Modeling

Introduction to N–grams



Language Modeling

Smoothing:

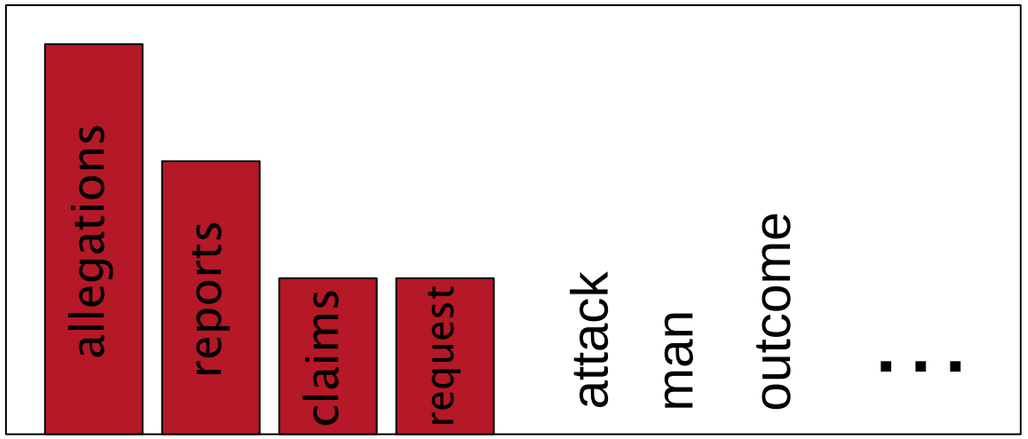
Add–one (Laplace) smoothing

The intuition of smoothing (from Dan Klein)

When we have sparse statistics:

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

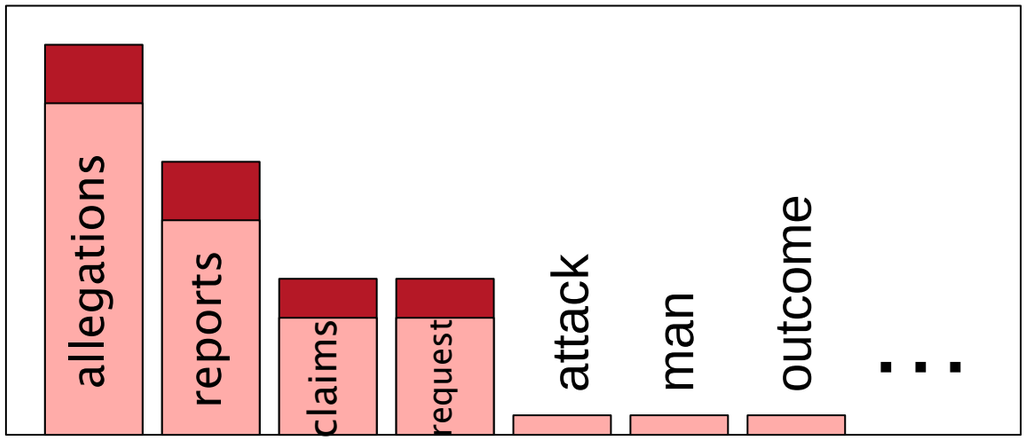
7 total



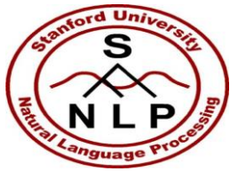
- Steal probability mass to generalize better

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

7 total



Add-one estimation

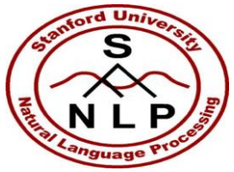


- Also called Laplace smoothing
- Pretend we saw each word one more time than we did
- Just add one to all the counts!
- MLE estimate:

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

- Add-1 estimate:

$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)+1}{c(w_{i-1})+V}$$



Maximum Likelihood Estimates

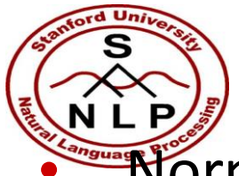
- The maximum likelihood estimate
  - of some parameter of a model M from a training set T
  - maximizes the likelihood of the training set T given the model M
- Suppose the word “bagel” occurs 400 times in a corpus of a million words
- What is the probability that a random word from some other text will be “bagel”?
- MLE estimate is 400/1,000,000 = .0004
- This may be a bad estimate for some other corpus
  - But it is the **estimate** that makes it **most likely** that “bagel” will occur 400 times in a million word corpus.

Raw bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Raw bigram probabilities



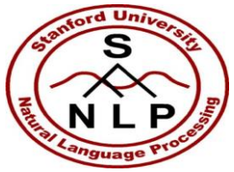
- Normalize by unigrams:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

- Result:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Berkeley Restaurant Corpus: Laplace smoothed bigram counts



	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Laplace-smoothed bigrams

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

P* (want | i)= count (i want) + 1/count (i) + V

827+1/ 2533 + 1446=0.2081=0.21 (round of)

	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

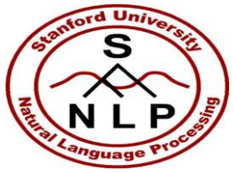
Reconstituted counts

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1] \times C(w_{n-1})}{C(w_{n-1})+V}$$

Prob(want | i) * count (i) = count (i want)
0.2081 * 2533 = **527**

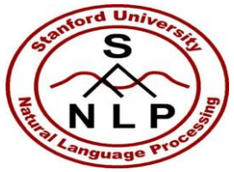
	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Compare with raw bigram counts



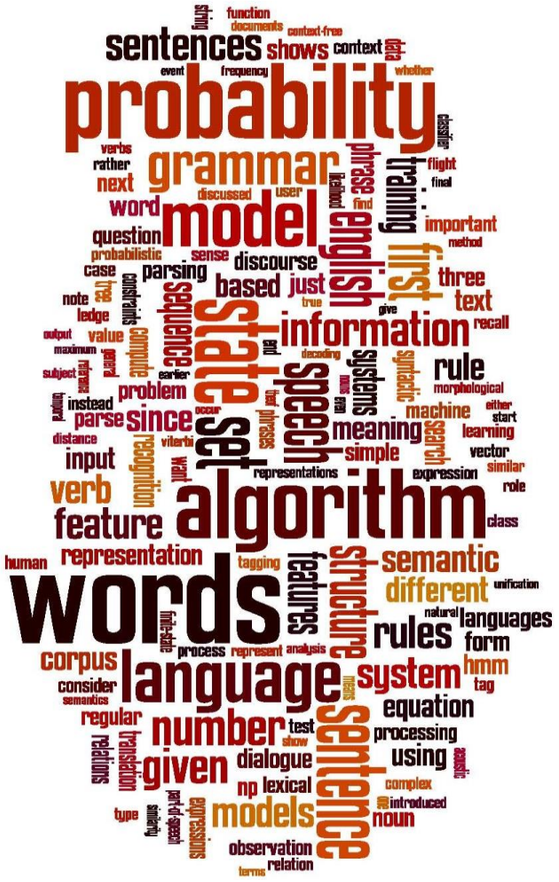
	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16



Add-1 estimation is a blunt instrument

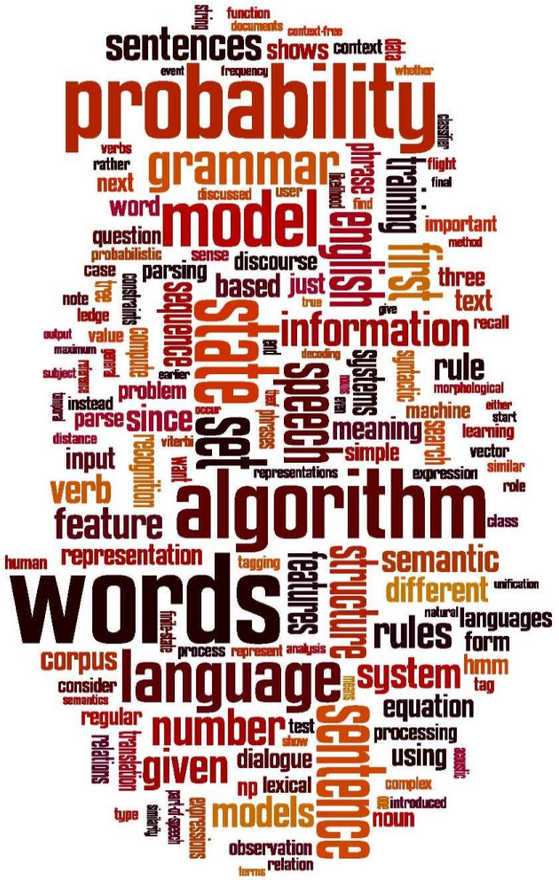
- So add-1 isnt used for N--grams:
  - We'll see better methods
- But add-1 is used to smooth other NLP models
  - For text classification
  - In domains where the number of zeros isn't so huge.



Language Modeling

Smoothing:

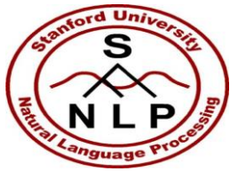
Add–one (Laplace) smoothing



Language Modeling

Interpolation, Backoff, and Web-Scale LMs

Backoff and Interpolation



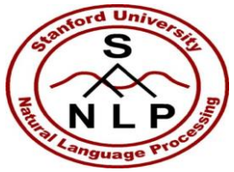
- Sometimes it helps to use **less** context
  - Condition on less context for contexts you haven't learned much about
- **Backoff:**
  - Go to a lower order n-gram if the higher order n-gram count is unreliable or zero
  - use trigram if you have good evidence, otherwise bigram, otherwise unigram

The quick brown fox jumped over ***the lazy dog***

P(dog| the lazy) if this trigram has an unreliable or zero value

P(dog|lazy) back off to a lower order n-gram

Interpolation

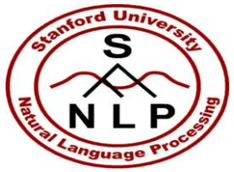


- mix unigram, bigram, trigram
- Add lambda with lower order n-gram and use it for prob estimate
- Add lower order n-grams to the estimate of higher order n-grams

*The quick brown fox jumped over **the lazy dog***

P(dog| the lazy)= lambda P(dog| the lazy) + lambda P(dog|lazy) + lambda P(dog)

- Lambda estimated on training corpus, learn with diff lambdas on training set
- Evaluate on validation set (for which lambda perplexity is minimal: intrinsic evaluation)
- An interpolated model combines the probabilities from unigram, bigram, and trigram models to predict the next word.



Interpolation

The basic idea is to interpolate between *higher-order n-gram models that have lower frequency* and lower-order n-gram models that have higher frequency.

Linear Interpolation

- Simple interpolation

$$P(w_i \mid w_{i\_1}) = lambda \frac{C(wi_{\_1}\, wi)}{C(wi_{\_1})} + lambda * P(wi)$$

$$P(w_i|w_{i-1}, w_{i-2}) = \lambda_3 \cdot P_{\text{trigram}}(w_i|w_{i-1}, w_{i-2}) + \lambda_2 \cdot P_{\text{bigram}}(w_i|w_{i-1}) + \lambda_1 P_{\text{unigram}}(w_i)$$

- Add a weighted estimate of a lower order n-gram
- Can be done recursively (trigram, bigram, unigram)
- Lambda estimated on training corpus
- Stupid Backoff

Linear Interpolation

- Simple interpolation

$$\hat{P}(w_n|w_{n-1}w_{n-2}) = \lambda_1 P(w_n|w_{n-1}w_{n-2}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n)$$

$$\sum_i \lambda_i = 1$$

- Lambdas conditional on context:

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1(w_{n-2}^{n-1})P(w_n|w_{n-2}w_{n-1}) + \lambda_2(w_{n-2}^{n-1})P(w_n|w_{n-1}) + \lambda_3(w_{n-2}^{n-1})P(w_n)$$

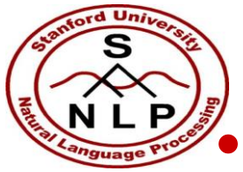
How to set the lambdas?

Use a **held--out** corpus

- Choose λs to maximize the probability of held--out data:
  - Fix the N-gram probabilities (on the training data)
  - Then search for λs that give largest probability to held--out set:

$$\log P(w_1 \ldots w_n \mid M(\lambda_1 \ldots \lambda_k)) = \sum_i \log P_{M(\lambda_1 \ldots \lambda_k)}(w_i \mid w_{i-1})$$

Find the set of probabilities such that the log prob of the actual words that occurred in the held out data is highest



Huge web-scale n-grams

- How to deal with, e.g., Google N-gram corpus
- Pruning
  - Only store N-grams with count > threshold.
    - Remove singletons of higher--order n-grams
  - Entropy--based pruning
- Efficiency
  - Efficient data structures
  - Store words as indexes, not strings
    - Use Huffman coding to fit large numbers of words into two bytes
  - Quantize probabilities (4-8 bits instead of 8-byte float)

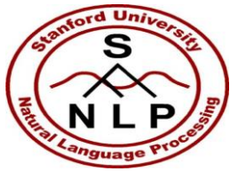
Smoothing for Web-scale N-grams

- "Stupid backoff" (Brants *et al*. 2007)
- No discounting, just use relative frequencies

$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-k+1}^{i})}{\text{count}(w_{i-k+1}^{i-1})} & \text{if} \quad \text{count}(w_{i-k+1}^{i}) > 0 \\ 0.4S(w_i \mid w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

Stupid Backoff



*The quick brown fox jumped over **the lazy dog***

P("dog" | "the lazy") = lambda * P("dog" | "lazy")

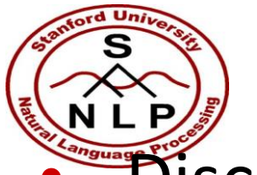
where lambda is a smoothing parameter, typically set to 0.4 or 0.5.

- Lambda is fixed, based on empirical analysis
- Reduce the estimate of a lower n-gram to get a better estimate of a higher order n-gram
- **Stupid backoff** produces scores rather than probabilities. And this works quite well for large scale N-Grams.

N-gram Smoothing Summary

- Add-1 smoothing:
  - OK for text categorization, not for language modeling
- The most commonly used method:
  - Extended Interpolated Kneser--Ney
- For very large N-grams like the Web:
  - Stupid backoff

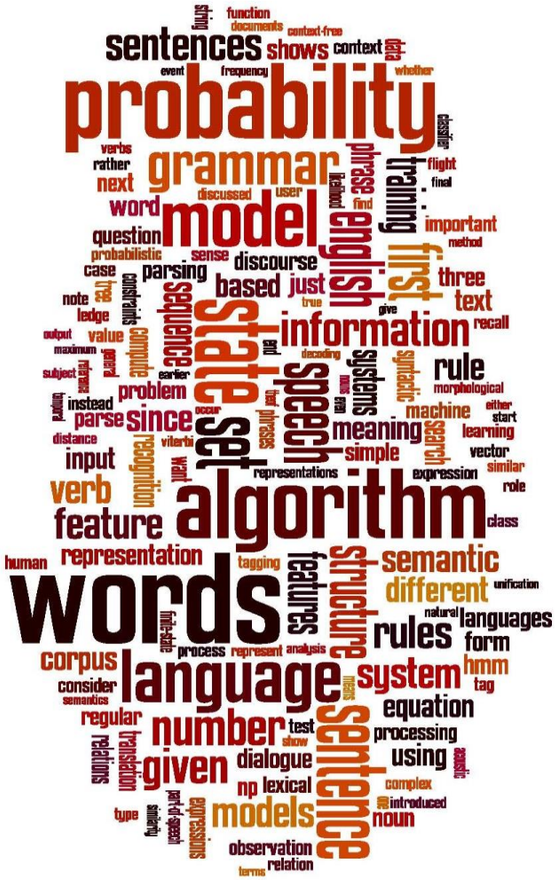
Advanced Language Modeling



- Discriminative models:
  - choose n-gram weights to improve a task, not to fit the training set
- Caching Models
  - Recently used words are more likely to appear

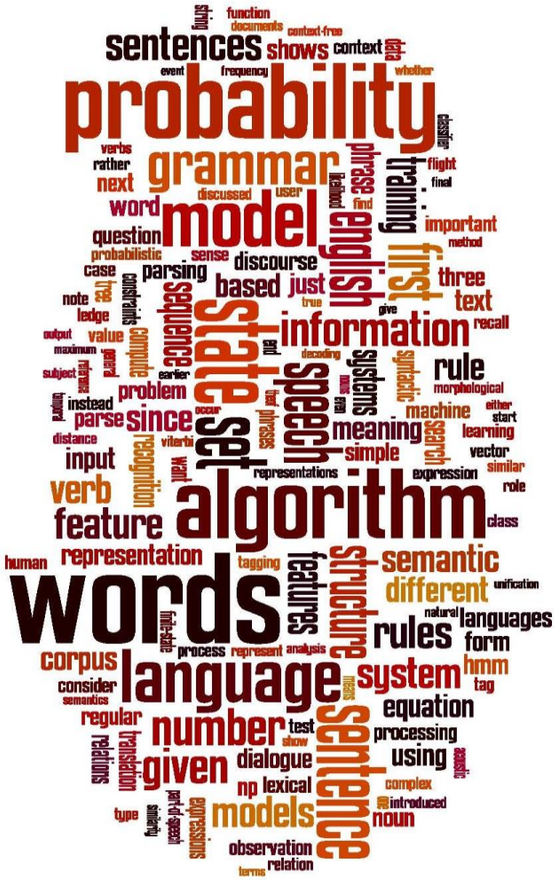
$$P_{CACHE}(w \mid history) = \lambda P(w_i \mid w_{i-2}w_{i-1}) + (1-\lambda)\frac{c(w \in history)}{|history|}$$

  - These perform very poorly for speech recognition (why?)



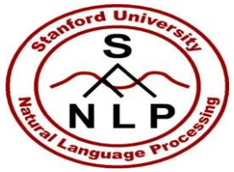
Language Modeling

Interpolation,
Backoff, and Web-Scale LMs



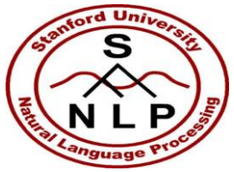
Language Modeling

Advanced:
Good Turing Smoothing



Reminder: Add-1 (Laplace) Smoothing

$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)+1}{c(w_{i-1})+V}$$

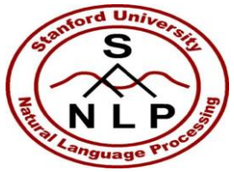


More general formulations: Add-k

$$P_{Add-k}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + k}{c(w_{i-1}) + kV}$$

$$P_{Add-k}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + m(\frac{1}{V})}{c(w_{i-1}) + m}$$

Can you use k>1?

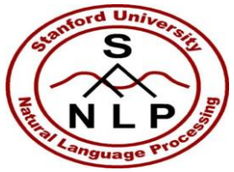


More general formulations: Add-k

$$P_{Add-k}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + m(\frac{1}{V})}{c(w_{i-1}) + m}$$

If all words occur uniformly, probability for words:
1/|v|
P(word)= 1/|V|

m>0
V=10000
1/|V|=0.0001
For good estimation m < |V|
If m=10000, it becomes add-1 smoothing



Unigram prior smoothing

$$P_{Add-k}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + m(\frac{1}{V})}{c(w_{i-1}) + m}$$

$$P_{\text{UnigramPrior}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + mP(w_i)}{c(w_{i-1}) + m}$$