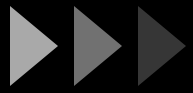
NLP:
Language Modeling

NLP Seminar, TAU, 2010, Semester A



Basic Idea

- Examine short sequence of words.
- How likely is each sequence?

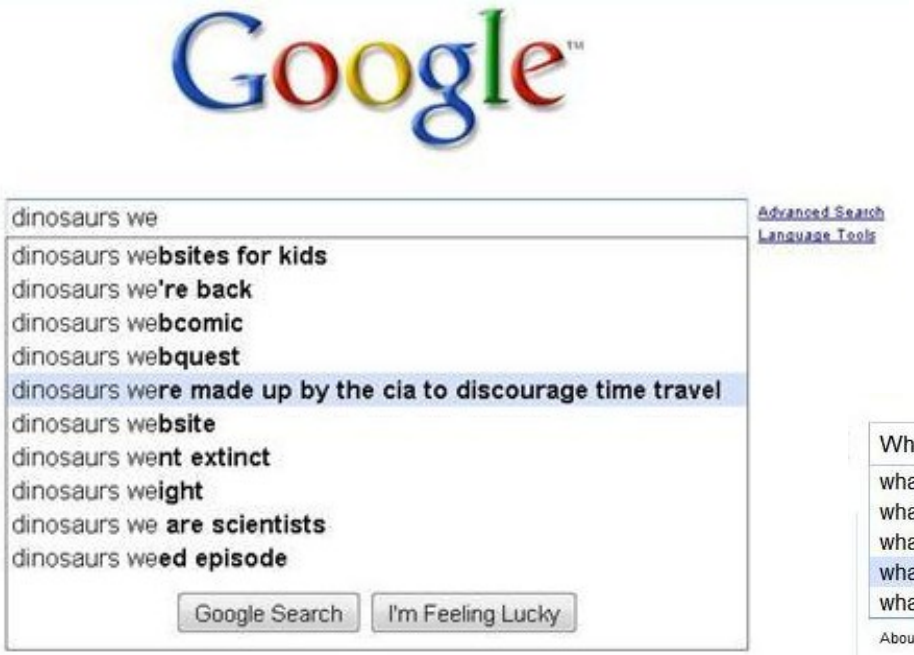
Statistical Language Models

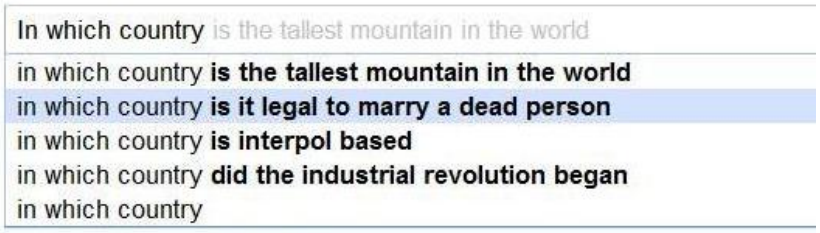
- The original (and is still the most important) application of SLMs is speech recognition.

("I ate a cherry" is a more likely sentence than "Eye eight uh Jerry")

- SLMs also play a vital role in other natural language applications: machine translation, POS tagging, intelligent input method (autocomplete) and Text To Speech system.
- What we want to accomplish:
  - Given a short sequence of words – how likely is it?
  - Markov assumption in effect
- Suppose we have a model – how do we determine if it's "good"?

When autocomplete goes wrong…





Shannon's Game

- *Claude E. Shannon. "Prediction and Entropy of Printed English", Bell System Technical Journal 30:50-64. 1951.*
- Predict the next word, given (*n-1)* previous words
  - Please turn off your cell _____
  - I want to go to _____

- **Woody Allen - Take the money and run:** http://www.youtube.com/watch?v=-UHOgkDbVqc

Forming Equivalence Classes (Bins)

- "*n-gram*" = sequence of n words
  - bigram
  - trigram
  - four-gram

- Estimate probability of each word given prior context.
  - P(phone | Please turn off your cell)
- Number of parameters required grows exponentially with the number of words of prior context.
- An N-gram model uses only N-1 words of prior context.
  - Unigram: P(phone)
  - Bigram: P(phone | cell)
  - Trigram: P(phone | your cell)

Reliability vs. Discrimination

- **Reliability vs. Discrimination**

  “large green _____”

  tree? mountain? frog? car?

  “swallowed the large green _____”

  pill? broccoli?

- **larger n:** more information about the context of the specific instance (greater discrimination). Also, Using larger n sizes exponentially increases the computational complexity.
- **smaller n:** more instances in training data, better statistical estimates (more reliability)

Formulas

- Word sequences (n-gram)

$$w_1^n = w_1 \ldots w_n$$

- Chain rule of probability

$$P(w_1^n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2)\ldots P(w_n \mid w_1^{n-1}) = \prod_{k=1}^{n} P(w_k \mid w_1^{k-1})$$

- Bigram approximation

$$P(w_1^n) = \prod_{k=1}^{n} P(w_k \mid w_{k-1})$$

- N-gram approximation

$$P(w_1^n) = \prod_{k=1}^{n} P(w_k \mid w_{k-N+1}^{k-1})$$

Formulas

- Let us focus on the bin ***comes across***. In a certain corpus the authors found 10 training instances of the words ***comes across***. 8 times they were followed by ***as***, once by ***more*** and once by ***a***.
- A naïve approach:

$$P_{MLE}(w_1 \cdots w_n) = \frac{C(w_1 \cdots w_n)}{B}$$

$$P_{MLE}(w_n \mid w_1 \cdots w_{n-1}) = \frac{C(w_1 \cdots w_n)}{C(w_1 \cdots w_{n-1})}$$

$$P(as) = 0.8$$
$$P(more) = 0.1$$
$$P(a) = 0.1$$
$$P(ANY\_OTHER\_WORD) = 0$$

C - frequency of n-gram
B - number of training instances (actual *n* word sequences we found in the text), aka **bins**

if bigram: B = text.length – 1
if trigram: B = text.length – 2

The problem

- Obviously flawed – we'll never accept sequences we didn't see in our training corpus (very sparse data), for instance: ***comes across seven dwarves*.**
- Suppose we want to estimate the likelihood of the sentence **he likes apple cake**:

$$P_{MLE}(he, likes, apple, cake)$$

- According to the chain rule:

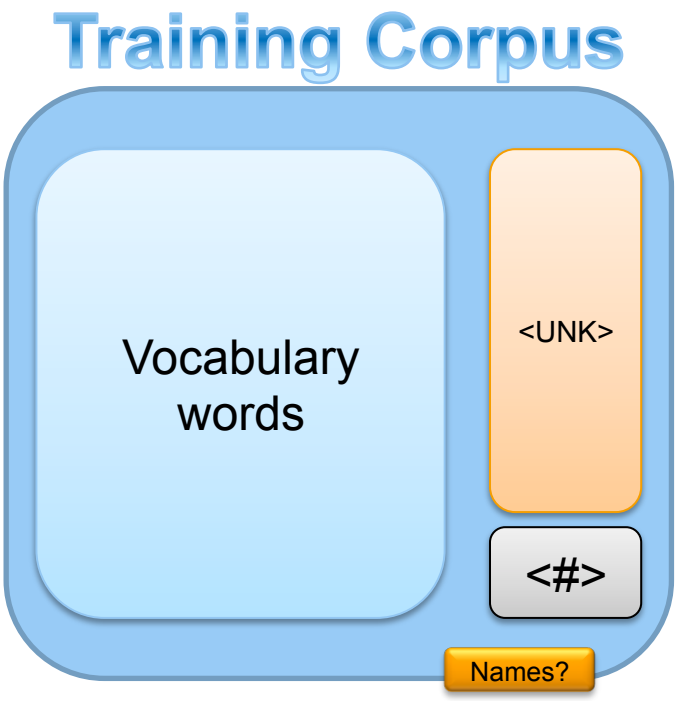
$$= P(he) \cdot P(likes \mid he) \cdot P(apple \mid he, likes) \cdot P(cake \mid he, likes, apple)$$

- Assuming a 2nd order Markov Chain model:

$$= P(he) \cdot P(likes \mid he) \cdot P(apple \mid likes) \cdot P(cake \mid apple)$$

- **If one of them equals zero it negates the entire probability.**

How to count bigrams?



Count	Bigram
17	(I, first)
11	(When, we)
1	(Raspberry, cake)
…	
5	<UNK, county>
3	<year, #>

<UNK> := words not in our vocabulary (OOV), or all words which appear only once in the text (*Hapax Legomenon*)

Selecting an N

Vocabulary (V) = 20,000 words

N	Number of bins (V^2)
2 (bigrams)	400,000,000
3 (trigrams)	8,000,000,000,000
4 (4-grams)	1.6×10^{17}

Smoothing

But what happens if we try to calculate

$$P_{MLE}(w_n \mid w_1 \cdots w_{n-1}) = \frac{C(w_1 \cdots w_n)}{C(w_1 \cdots w_{n-1})}$$

but w_n is a new word we never saw in the training corpus? P will equal 0…
We can "back off" to lower n-gram models, and we will use this method later (combined with other methods). But it's not good enough.
What about single words we never saw?

So….
Smoothing methods to the rescue!

In	*person*	*she*		*was*		inferior		*to*		*both*		*sisters*	
1-gram		$P(\cdot)$		$P(\cdot)$		$P(\cdot)$		$P(\cdot)$		$P(\cdot)$		$P(\cdot)$	
		the*	0.034	the	0.034	the	0.034	the	*0.034*	the	*0.034*	the	0.034
		to	0.032	to	0.032	to	0.032	to	*0.032*	to	*0.032*	to	0.032
		and	0.030	and	0.030	and	0.030			and	*0.030*	and	0.030
		of	0.029	of	0.029	of	0.029			of	0.029	of	0.029
	8	was	0.015	was	0.015	was	0.015			was	0.015	was	0.015
	13	she	0.011			she	0.011			she	0.011	she	0.01 I
	264					both	0.0005			both	0.0005	both	0.0005
	435					sisters	0.0003					sisters	0.0003
	1701					inferior	0.00005						
Z-gram		$P(\cdot \mid person)$		$P(\cdot \mid she)$		$P(\cdot \mid was)$		$P(\cdot \mid inferior)$		$P(\cdot \mid to)$		$P(\cdot \mid both)$	
	1	and	0.099	had	0.141	not	0.065	to	*0.2 12*	be	0.111	of	0.066
	2	who	0.099	was	0.122	a	0.052			the	0.057	to	0.041
	3	to	0.076			the	0.033			her	0.048	in	0.038
	4	in	0.045			to	0.031			have	0.027	and	0.025
	23	she	0.009							Mrs	0.006	she	0.009
	41									what	0.004	sisters	0.006
	293									**both**	0.0004		
	∞					inferior	0						
3-gram		$P(\cdot \mid In,person)$		$P(\cdot \mid person,she)$		$P(\cdot \mid she,was)$		$P(\cdot \mid was,inf.)$		$P(\cdot \mid inferior,to)$		$P(\cdot \mid to,both)$	
		UNSEEN		did	0.5	not	*0.057*	UNSEEN		the	*0.286*	to	0.222
	2			was	0.5	very	*0.038*			Maria	0.143	Chapter	0.111
						in	0.030			cherries	0.143	Hour	0.111
	4					to	0.026			her	0.143	Twice	0.111
	∞					inferior	0			both	0	sisters	0
4-gram		$P(\cdot \mid u,l,p)$		$P(\cdot \mid l,p,s)$		$P(\cdot \mid p,s,w)$		$P(\cdot \mid s,w,i)$		$P(\cdot \mid w,i,t)$		$P(\cdot \mid i,t,b)$	
		UNSEEN		UNSEEN		in	1.0	UNSEEN		UNSEEN		UNSEEN	
	∞					inferior	0						

Smoothing (cont.)

- parameters are ***smoothed*** (a.k.a. ***regularized***) to reassign some probability mass to unseen events.
- Adding probability mass to unseen events requires removing it from seen ones (***discounting***) in order to maintain a joint distribution that sums to 1.
- Various smoothing methods:
  - Discounting methods – Laplace, Lidstone
  - Validation - Smoothing methods which utilize a second batch of test data.

Smoothing (cont.)

- What are the chances that the sun won't shine tomorrow?

- Homo sapiens have existed for approx. 250,000 years (almost 100 million days). Laplace says we have to take into account a single unknown day the sun didn't shine.

Laplace (Add-One) Smoothing

- "Hallucinate" additional training data in which each word occurs exactly once in every possible (N-1)-gram context and adjust estimates accordingly.

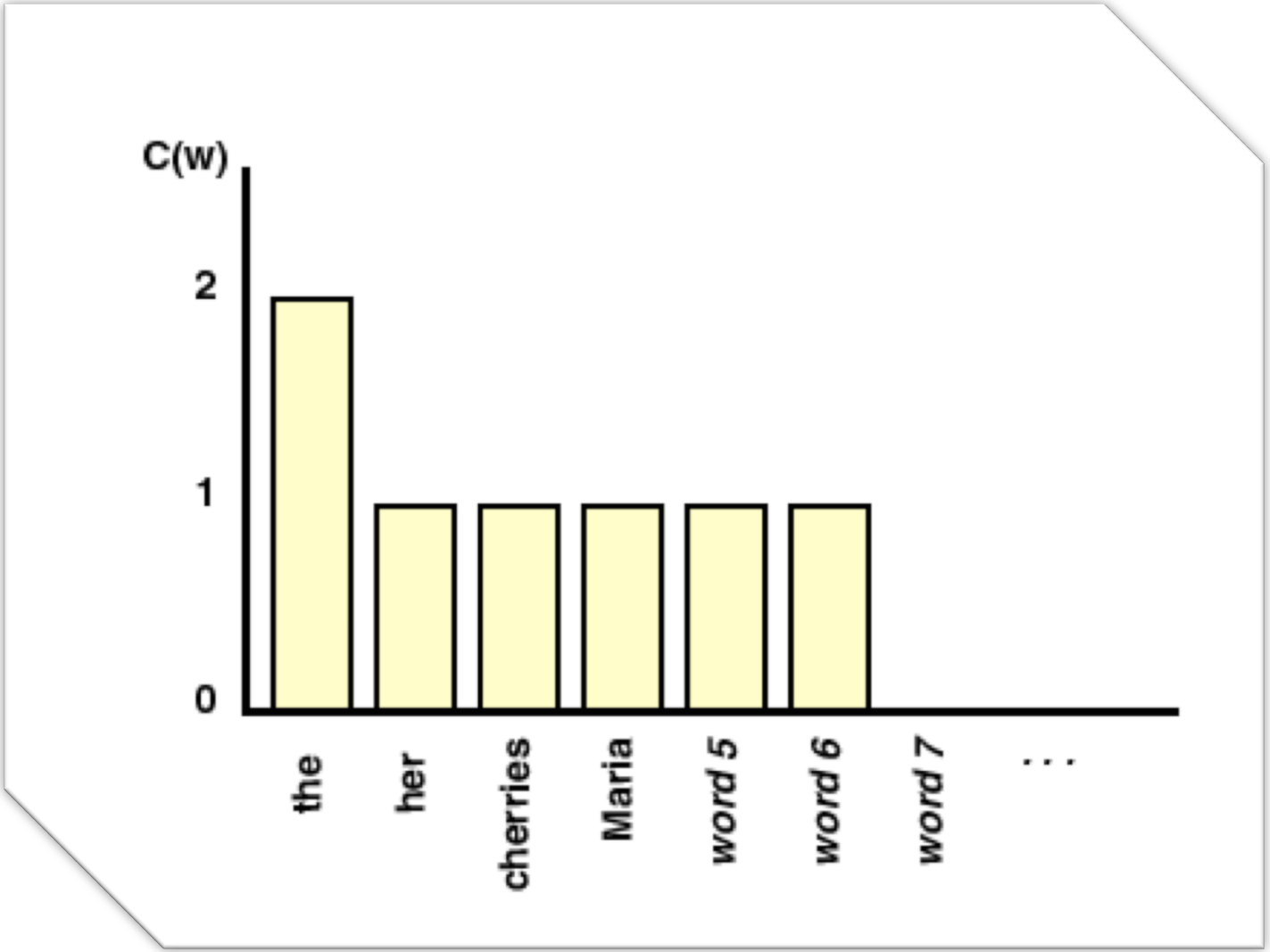
Bigram: $$P(w_1 \cdots w_n) = \frac{C(w_1 \cdots w_n) + 1}{N + B}$$

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

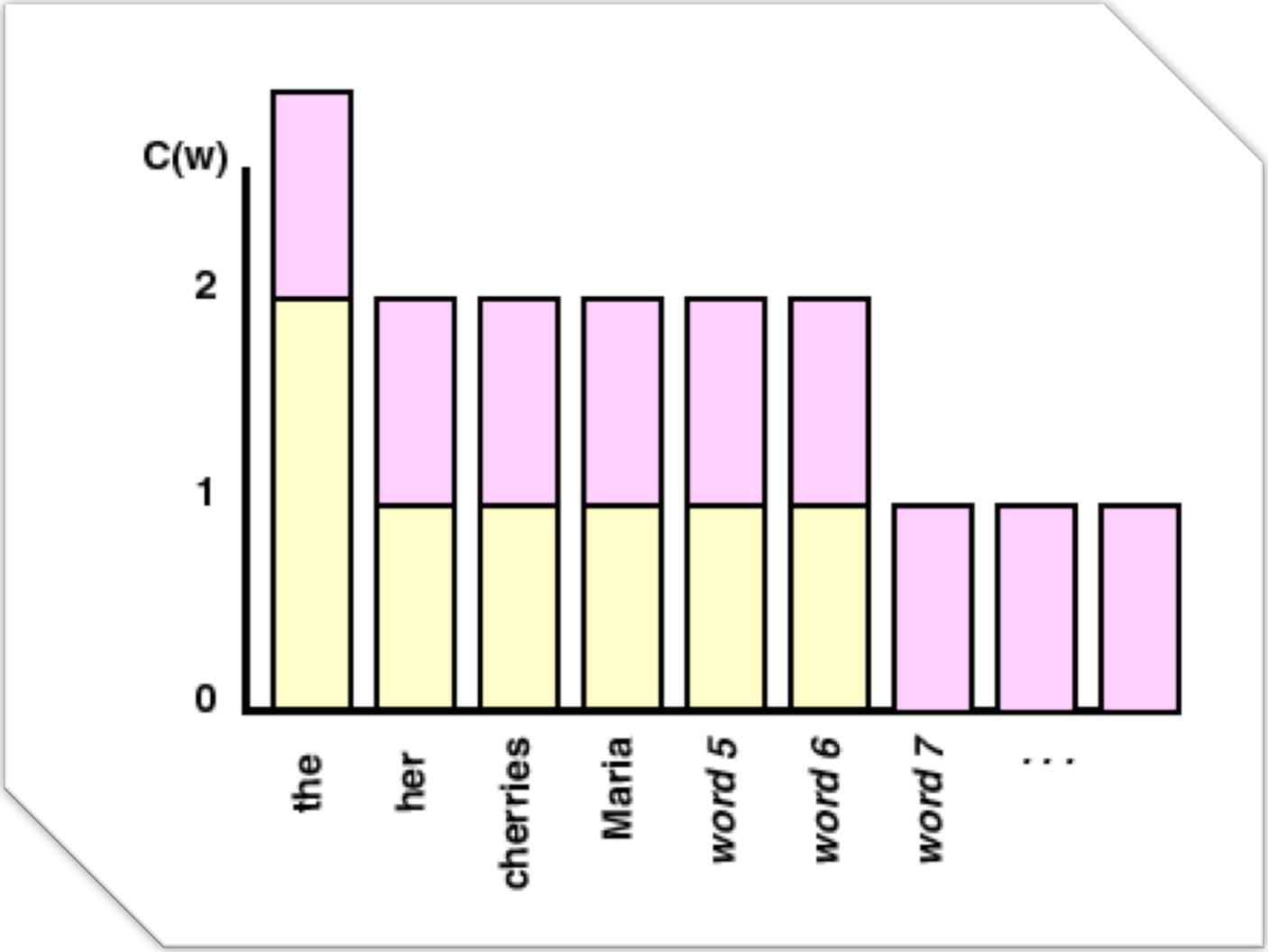
B is the number of possible bins

V is the total number of possible words (i.e. the vocabulary size).

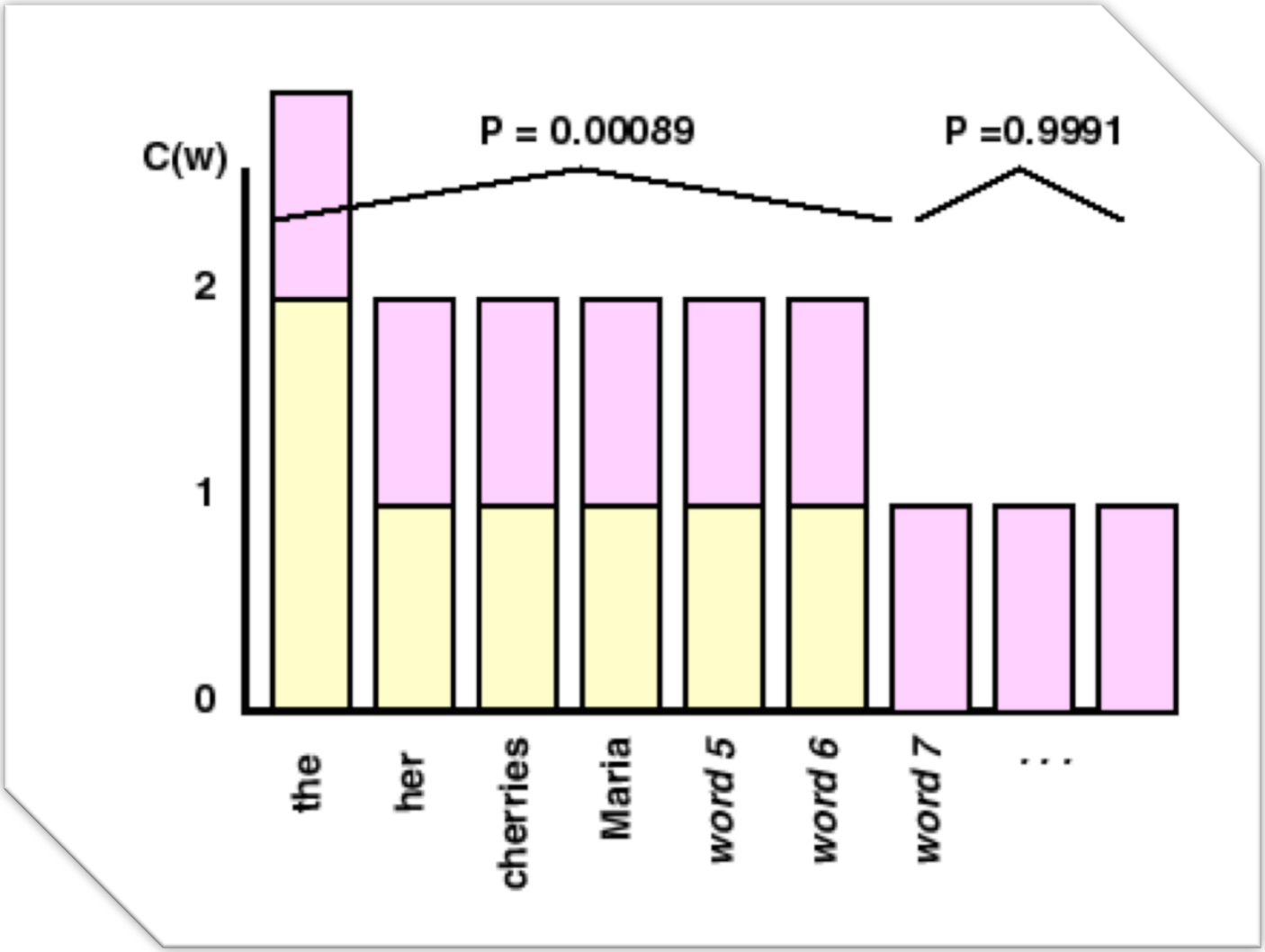
Laplace's Law (adding one)



Laplace's Law (adding one)



Laplace's Law (adding one)



Lidstone's Law

- Laplace tends to reassign too much mass to unseen events, so it can be adjusted to add 0<λ<1, normalized by *B*λ instead of *B* (*Lindstone)*

$$P_{Lid}(w_1 \cdots w_n) = \frac{C(w_1 \cdots w_n) + \lambda}{N + B\lambda}$$

P = probability of specific n-gram

C = count of that n-gram in training data

N = total n-grams in training data

B = number of "bins" (possible n-grams)

λ = small positive number

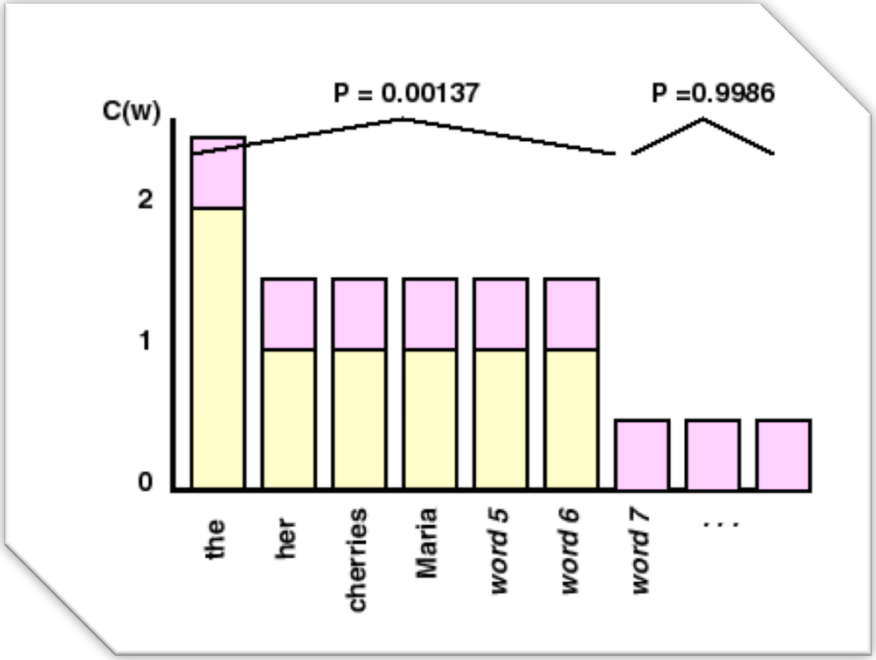
M.L.E: λ = 0
LaPlace's Law: λ = 1
Jeffreys-Perks Law: λ = ½

Objections to Lidstone's Law

- Need an *a priori* way to determine λ.
- Predicts all unseen events to be equally likely.
- Works really bad for large texts

Jeffreys-Perks Law: λ = ½ :

- So instead of modifying the *counts*, lets modify the *probabilities.*



A simple example

- consider the trigram estimates for a new language ABC with V={A, B, C}, and given the following corpus:
  A B C A B B C C A C B C A A C B C C B
- There are $3^3 = 27$ different possible trigram types, only 13 of which are observed outcomes (ABC, BCA, CAB, …,CBC) and thus 14 have not been seen. The raw counts are show in the following table, organized by the "context" (i.e., the previous two letters).

Trigram	Count	Trigram	Count	Trigram	Count	Context	Count
AAA	0	AAB	0	AAC	1	AA	1
ABA	0	ABB	1	ABC	1	AB	2
ACA	0	ACB	2	ACC	0	AC	2
BAA	0	BAB	0	BAC	0	BA	0
BBA	0	BBB	0	BBC	1	BB	1
BCA	2	BCB	0	BCC	2	BC	4
CAA	1	CAB	1	CAC	1	CA	1
CBA	0	CBB	0	CBC	3	CB	3
CCA	1	CCB	1	CCC	0	CC	2

A simple example (cont.)

- The smoothed counts using the add-λ Technique

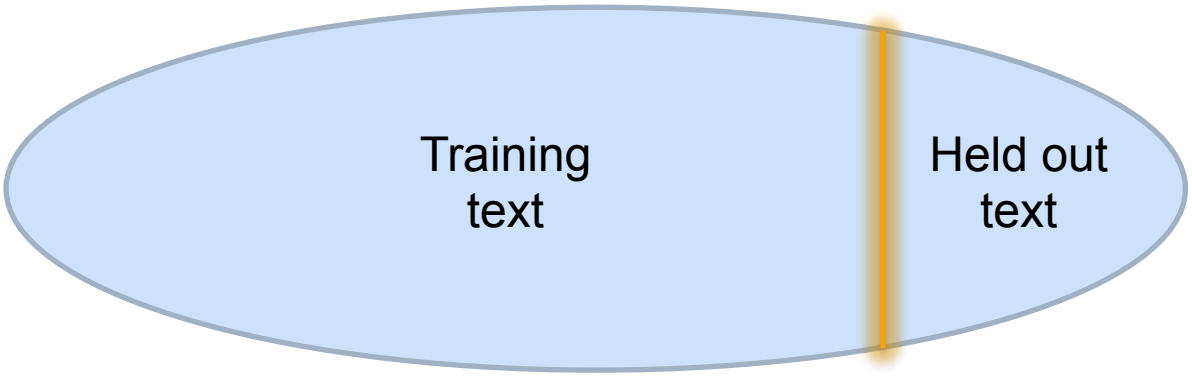
Trigram	Count	Trigram	Count	Trigram	Count	Context	Count
AAA	λ	AAB	λ	AAC	1+λ	AA	1+3λ
ABA	λ	ABB	1+λ	ABC	1+λ	AB	2+3λ
ACA	λ	ACB	2+λ	ACC	λ	AC	2+3λ
BAA	λ	BAB	λ	BAC	λ	BA	3λ
BBA	λ	BBB	λ	BBC	1+λ	BB	1+3λ
BCA	2+λ	BCB	λ	BCC	2+λ	BC	4+3λ
CAA	1+λ	CAB	1+λ	CAC	1+λ	CA	1+3λ
CBA	λ	CBB	λ	CBC	3+λ	CB	3+3λ
CCA	1+λ	CCB	1+λ	CCC	λ	CC	2+3λ

- Since $P_{Lid}(w_n \mid w_1 \cdots w_{n-1}) = \frac{C(w_1 \cdots w_n) + \lambda}{C(w_1 \cdots w_{n-1}) + V \cdot \lambda}$: *(a sampling of trigrams)*

Probability	λ = 1	λ = .5	λ = .31	λ = .01
P(A | AA)	.25	.2	.16	.01
P(C | AA)	.5	.6	.68	.98
P(C | BA)	.333	.333	.333	.333
P(A | BC)	.428	.45	.47	.499
P(B | BC)	.143	.2	.06	.002
P(C | BC)	.428	.45	.47	.499

Held-out Estimator

- For this technique, we will call this additional corpus **the held out data,** for in essence it is a subset of the training data that is withheld from the initial estimation procedure.
- We could count how many times each observation occurred in both a training set and the held-out data, and use these number to explore how often observations that occur **r** times in the training data occur in the development data.
- This would not only give us an estimate of how often observations **not** in the training data occur in subsequent data but also how often observations that occur **r times** occur in subsequent data.



Held-out Estimator

- We first divide our training data into two corpora: T, still called the training data, and HO, called the held out data.
- For each sequence $w_1 \ldots w_n$, we count: $C_T(w_1 \cdots w_n)$ $C_{HO}(w_1 \cdots w_n)$
- We are interested in n-grams with the same number of occurrences.
- Let N_r be the set of all sequences that occurred exactly **r** times in the training corpus. If we used the standard MLE technique, we would estimate the probability of each item in N_r as having a probability **r/N**.
- Rather than use this, however, we look at how many times elements in each class occur in HO. Specifically, we count how many times any of these items occur in the development corpus:

$$T_r = \sum_{(w_1 \ldots w_n) \in N_r} Count_{HO}(w_1 \ldots w_n)$$

Held-out Estimator

- Now we can compute the average number of times an observation that occurred **r** times in the training corpus occurs in the HO corpus:
- AverageCnt(r) = T_r/N_r
- We now use this "adjusted count" for producing the probability estimate:

$$P_{HO}(w_1 \cdots w_n) = \frac{AverageCnt(r)}{N_{HO}}$$

(N_{HO} is the size of the held out data)

- The most clear case where this helps is seen by looking at the elements that do not occur in the training corpus. Let's use the previous trigram example. There were 14 trigrams that did not occur in the training corpus. Let's assume only 6 of these occur (or actually, don't occur ☺) in the HO text with size 40 (it doesn't matter if the same word occurred three times or three words occurred once). So we have T_0= 6
- Thus the average number of times we can expect one of these unseen bigrams to appear in the held out data is AverageCnt(0) = 6/40 = .15
- Thus, the held-out probability estimate for any one of these, say, AAA, is P_{HO}(AAA) = .15/40 = .00375

Different approaches for HO data

- As mentioned above, one disadvantage of the held out method appears to be that we need to reduce our training corpora in order to produce the held out data. So the smaller initial training set could hurt us.
- One way around this is two do the estimate with several different sets for held-out data . Say we divide the corpus into two parts: A and B. First we initially train on A and use B as the held out data, and then we create a new model by training on B and using A as the held-out data.
- We then have two estimates, P_{m1} and P_{m2} and can combine them to make a new estimate P_m using:

$$P_m(w_1 \cdots w_n) = .5P_{m1}(w_1 \cdots w_n) + .5P_{m2}(w_1 \cdots w_n)$$

- Of course, we need not divide the corpus exactly into half. Experience has shown that, assuming the corpus is large enough, it is better to use more data for the initial estimate, saving only about 10% for the held-out data. This suggests a generalization of the above strategy. We divide the initial corpus into ten parts, and build ten estimates each one using a different part as the held out data. The final probability distribution would then be constructed using the average over the ten estimations obtained.

Cross Validation

- Note we can use the same technique for test data as well as for evaluation. Say we have one corpus and need to select part of it as the test data. Once again, typically people choose test data to be 10% of the size of the training data. Rather than doing this once, we could do a more extensive evaluation of a technique by do ten experiments, each one using a different subset of the corpora as the test set and the rest as the training.
- This is called the cross-validation technique and is an effective way to perform more extensive testing of a technique without requiring more data.
- Of course, if we were evaluating a technique that requires held-out data, we'd need to combine both approaches. In this case, we'd select one subset as test data, one as held-out data and train on the rest. There are now a hundred combinations that perform different experiments using the same corpora.
- The final evaluation result would be the average of the individual experiments performed.

Experiment 1

Experiment 2

Etc…

Good-Turing Estimation

- Reminder: $C(w_1 \cdots w_n) = r$

  $N_r = \text{number of n-grams which appear r times.}$

  $(\text{frequency of frequency})$

- Good and Turing estimated that text has a binomial distribution (on r and N_r)

$$r^* = (r+1)\frac{E(N_{r+1})}{E(N_r)}$$

$E(N_r)$ = "expected value"

$E(N_{r+1}) < E(N_r)$

$$P_{GT} = r^* / N$$

- Works bad for big r's, mainly used for small r's (N_r is big, due to Zipf's Law).
- There are ways of normalizing probabilities to integrate GT and MLE.

Linear Interpolation

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

Where: $\sum_i \lambda_i = 1$

- We combine several n-grams
- Learn proper values for λ_i by training to (approximately) maximize the likelihood of an independent ***development*** (a.k.a. ***tuning***) corpus.

Backoff

- Only use lower-order model when data for higher-order model is unavailable (i.e. count is zero).
- Recursively back-off to weaker models until data is available.

$$P_{katz}(w_n \mid w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n \mid w_{n-N+1}^{n-1}) & \text{if } C(w_{n-N+1}^{n-1}) > 1 \\ \alpha(w_{n-N+1}^{n-1}) P_{katz}(w_n \mid w_{n-N+2}^{n-1}) & \text{otherwise} \end{cases}$$

- Where P* is a discounted probability estimate to reserve mass for unseen events and α's are back-off weights (see text for details).

Entropy

- Suppose you want to maximize the amount of data you can transmit.
- Example: random number between 1-8. Binary encoding – need 3 bits.
- Entropy: $$H(X) = -\sum_{x \in X} p(x) \ln(p(x))$$
- What if each random number has a different probability?
- Simplified Polynesian (letter frequency):

p	t	k	a	i	u
1/8	1/4	1/8	1/4	1/8	1/8

$$H(P) = -\sum_{i \in \{p,t,k,a,i,u\}} P(i) \log(P(i)) = -\left[4 \times \frac{1}{8} \log \frac{1}{8} + 2 \times \frac{1}{4} \log \frac{1}{4}\right] = 2\frac{1}{2}$$

Entropy

- Example of encoding “exploiting” new letter frequency information:

p	t	k	a	i	u
100	00	101	01	110	111

(more frequent words take up 2 bits and start with 0)

- The joint entropy of a pair of discrete random variables X, Y is the amount of information needed on average to specify both their values. It is defined as:

$$H(X,Y) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) \log p(x.y)$$

- There is also a chain rule for entropy:

$$H(X,Y) = H(X) + H(Y \mid X) = H(Y) + H(X \mid Y)$$

- Therefore:

$$H(X) - H(X \mid Y) = H(Y) - H(Y \mid X) = I(X,Y)$$

I(X,Y) is called *mutual information*: the decrease in the uncertainty of X, given Y.

Simplified Polynesian - revisited

- Let's say we know more about Simplified Polynesian. Further fieldwork has revealed that Simplified Polynesian has syllable structure. It turns out that all words consist of sequences of CV (consonant-vowel) syllables. So the marginal distribution is as follows:

	p	t	k	
a	1/16	3/8	1/16	½
i	1/16	3/16	0	¼
u	0	3/16	1/16	¼
	1/8	**3/4**	**1/8**	

- A quick calculation will reveal that:

$$H(C) = 1.061$$
$$H(V \mid C) = 1.375$$
$$H(C,V) = 2.44$$

- Keep in mind these values are for whole syllables, i.e. **two** letters. In Comparison, we got 2.5 for **one** letter in the previous example (the actual result would be 5 bits on average for two letters)
- So, given the right encoding, we can pass data through even less bandwidth.

Entropy and model estimation

- So far we have examined the notion of entropy, and seen roughly how it is a guide to determining efficient codes for sending messages, but how does this relate to understanding language?
- Entropy is a measure of our uncertainty. The more we know about something (the language model), the lower the entropy will be, because we are less surprised by the outcome of a trial.

- If we make certain assumptions that a language is “nice”, then the cross entropy for a language L using a model *m* can be calculated as:

$$H(L,m) \approx -\frac{1}{n} \sum_{x_i \in sentences} \log(m(x_i))$$

- Our goal – to minimize this number as much as possible.
- If we were able to predict all words perfectly, the entropy would be 0.

Perplexity

- Another method of assessing a model.

$$perplexity(x_{1n}, m) = 2^{H(x_{1n}, m)} = m(x_{1n})^{-\frac{1}{n}}$$

- Same idea, simply non-logarithmic (according to Manning-Schutze: "*We suspect that speech recognition people prefer to report the larger non-logarithmic numbers given by perplexity mainly because it is much easier to impress funding bodies by saying that 'we've managed to reduce perplexity from 950 to only 540' than by saying 'we've reduced cross entropy from 9.9 to 9.1 bits'" (p. 78)*

References

- Foundations of Statistical Natural Language Processing, *Christopher D. Manning and Hinrich Schütze, The MIT Press, 1999*
- NLP Course 2010, TAU, Prof. Gideon Dror
- http://www.cs.rochester.edu/u/james/CSC248/
- http://homepages.inf.ed.ac.uk/lzhang10/slm.html
- http://www-nlp.stanford.edu/fsnlp/statest/
- Wikipedia

Interesting links

- **IBM and the Jeopardy Challenge:** http://www.youtube.com/watch?v=FC3IryWr4c8
- http://ngrams.googlelabs.com/
- http://web-ngram.research.microsoft.com/spellerchallenge/

Thank You !

Questions?

TEL AVIV UNIVERSITY אוניברסיטת תל-אביב