Natural Language Processing CS 6320

Lecture 3

Language Modeling with N-Grams

Instructor: Sanda Harabagiu

The problem

❑ Using the notion of word prediction for processing language

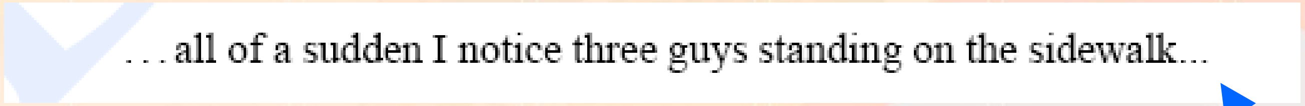
- Example:

  What word is most likely to follow:

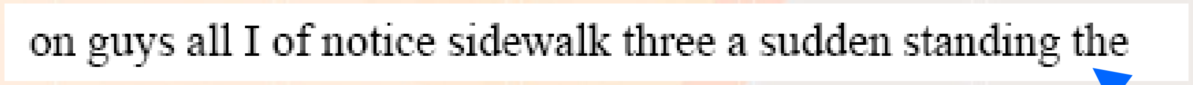
  I'd like to make a collect call

- Using probabilistic models called N-grams to predict the next word from the previous n-1 words.

❑ Computing the probability of the next word is related to computing *the probability of a sequence of words*.


. . . all of a sudden I notice three guys standing on the sidewalk . . .

Has a non-zero probability of appearing in a text


on guys all I of notice sidewalk three a sudden standing the

The same words in a different order have a very low probability

Today

- *Word prediction task*
- *Language modeling (N-grams)*
  - *N-gram intro*
  - *The chain rule*
  - *Model evaluation*
  - *Smoothing*

Word Prediction

- *Guess the next word...*
    - *... I notice three guys standing on the ???*
- *There are many sources of knowledge that can be used to inform this task, including arbitrary world knowledge.*
- *But it turns out that you can do pretty well by simply looking at the preceding words and keeping track of some fairly simple counts.*

Word Prediction

- *We can formalize this task using what are called N-gram models.*
- *N-grams are **token sequences of length N.***
- *Our earlier example contains the following 2-grams (aka bigrams)*
  - *(I notice), (notice three), (three guys), (guys standing), (standing on), (on the)*
- *Given knowledge of counts of N-grams such as these, we can guess likely next words in a sequence.*

N-Gram Models

- *More formally, we can use knowledge of the counts of N-grams to assess the conditional probability of candidate words as the next word in a sequence.*
- *Or, we can use them to assess the probability of an entire sequence of words.*
  - *Pretty much the same thing as we'll see...*

Applications

- *It turns out that being able to predict the next word (or any linguistic unit) in a sequence is an extremely useful thing to be able to do.*
- *As we'll see, it lies at the core of the following applications*
  - *Automatic speech recognition*
  - *Handwriting and character recognition*
  - *Spelling correction*
  - *Machine translation*
  - *And many more.*

Counting

- *Simple counting lies at the core of any probabilistic approach. So let's first take a look at what we're counting.*
  - *He stepped out into the hall, was delighted to encounter a water brother.*
    - *13 tokens, 15 if we include "," and "." as separate tokens.*
    - *Assuming we include the comma and period, how many bigrams are there?*

Counting

- *Not always that simple*
  - *I do uh main- mainly business data processing*
- *Spoken language poses various challenges.*
  - *Should we count "uh" and other fillers as tokens?*
  - *What about the repetition of "mainly"? Should such do-overs count twice or just once?*
  - *The answers depend on the application.*
    - *If we're focusing on something like ASR to support indexing for search, then "uh" isn't helpful (it's not likely to occur as a query).*
    - *But filled pauses are very useful in dialog management, so we might want them there.*

Counting: Types and Tokens

- *How about*
  - *They picnicked by the pool, then lay back on the grass and looked at the stars.*
    - *18 tokens (again counting punctuation)*
- *But we might also note that "the" is used 3 times, so there are only 16 unique types (as opposed to tokens).*
- *In going forward, we'll have occasion to focus on counting both types and tokens of both words and N-grams.*

Counting: Wordforms

- *Should "cats" and "cat" count as the same when we're counting?*
- *How about "geese" and "goose"?*
- *Some terminology:*
  - *Lemma: a set of lexical forms having the same stem, major part of speech, and rough word sense*
  - *Wordform: fully inflected surface form*
- *Again, we'll have occasion to count both lemmas and wordforms*

Counting: Corpora

- *So what happens when we look at large bodies of text instead of single utterances?*
- *Brown et al (1992) large corpus of English text*
  - *583 million wordform tokens*
  - *293,181 wordform types*
- *Google*
  - *Crawl of 1,024,908,267,229 English tokens*
  - *13,588,391 wordform types*
    - *That seems like a lot of types... After all, even large dictionaries of English have only around 500k types. Why so many here?*

•Numbers
•Misspellings
•Names
•Acronyms
•etc

Language Modeling

- *Back to word prediction*
- *We can model the word prediction task as the ability to assess the conditional probability of a word given the previous words in the sequence*
  - $P(w_n|w_1,w_2\ldots w_{n-1})$

- *We'll call a statistical model that can assess this probability a Language Model*

Language Modeling

- *How might we go about calculating such a conditional probability?*
  - *One way is to use the definition of conditional probabilities and look for counts. So to get*
  - *P(the | its water is so transparent that)*
- *By definition that’s*

$$\frac{P(\text{its water is so transparent that the})}{P(\text{its water is so transparent that})}$$

We can get each of those from counts in a large corpus!

Very Easy Estimate

- *How to estimate?*
  - *P(the | its water is so transparent that)*

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{\text{Count(its water is so transparent that the)}}{\text{Count(its water is so transparent that)}}$$

Very Easy Estimate (really???)

- *According to Google those counts are 5/9.*
  - *Unfortunately... 2 of those were to these slides... So maybe it's really*
  - *3/7*
  - *In any case, that's not terribly convincing due to the small numbers involved.*

Language Modeling

- *Unfortunately, for most sequences and for most text collections we won't get good estimates from this method.*
  - *What we're likely to get is 0. Or worse 0/0.*
- *Clearly, we'll have to be a little more clever.*
  - ➢ *Let's use the chain rule of probability*
  - ➢ *And a particularly useful independence assumption.*

The Chain Rule

- *Recall the definition of conditional probabilities*

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

- *Rewriting:*

$$P(A \wedge B) = P(A|B)P(B)$$

- *For sequences...*
  - $P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$
- *In general*
  - $P(x_1,x_2,x_3,\ldots x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_1\ldots x_{n-1})$

The Chain Rule

$$
\begin{aligned}
P(w_1^n) &= P(w_1)P(w_2|w_1)P(w_3|w_1^2)\ldots P(w_n|w_1^{n-1}) \\
&= \prod_{k=1}^{n} P(w_k|w_1^{k-1})
\end{aligned}
$$

P(its water was so transparent)= P(its)× P(water|its))× P(was|its water))× P(so|its water was))× P(transparent|its water was so)

- *UNFORTUNATELY!!!*
- *In general, we'll never be able to get enough data to compute the statistics for those longer prefixes*
  - *Same problem we had for the strings themselves*

Independence Assumption

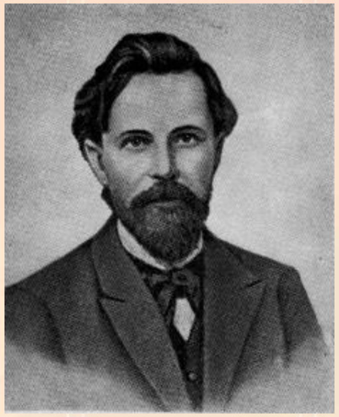
- ❑ *Make the* ***simplifying assumption***
  - *P(lizard|the,other,day,I,was,walking,along,and,saw,a) = P(lizard|a)*
- *Or maybe*
  - *P(lizard|the,other,day,I,was,walking,along,and,saw,a) = P(lizard|saw,a)*
- ✓ *That is, the probability in question is independent of its earlier history.*

- ➢ *When we use a bigram model to predict the conditional probability of the next word, we are thus making the following approximation:*

$$\boxed{P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})}$$

Independence Assumption

- *This particular kind of independence assumption is called a Markov assumption after the Russian mathematician Andrei Markov.*



The assumption that the probability of a word depends only on the previous word is called a **Markov assumption**.
Markov models are the class of probabilistic models that assume we can predict the probability of some future unit without looking too far into the past.

Markov Assumption

*Equation for n-gram **approximation** to the conditional probability of the next word in a sequence is :*

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-N+1}^{n-1})$$

Bigram version: $P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-1})$

But how do I compute these probabilities?

- ***Estimating Bigram Probabilities***

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

An Example

<s> I am Sam </s>

<s> Sam I am </s>

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

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

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

For the general case of MLE n-gram parameter estimation:

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

Maximum Likelihood Estimates

- *The maximum likelihood estimate of some parameter of a model M from a training set T*
  - *Is the estimate that maximizes the likelihood of the training set T given the model M*
- *Suppose the word Chinese occurs 400 times in a corpus of a million words (Brown corpus)*

What is the probability that a random word from some other text from the same distribution will be "Chinese"?

MLE estimate is 400/1000000 = .004

- *This may be a bad estimate for some other corpus*

*But it is the **estimate** that makes it **most likely** that "Chinese" will occur 400 times in a million word corpus.*

Counting words in corpora

- We count word forms not lemmas
  - *Word forms =words as they appear in the corpus*
  - *Lemmas= a set of lexical forms having the same stem, the same part-ofspeech, and the same wordsense.* Example: cat, cats.

- We distinguish **types** – the number of distinct words in a corpus of vocabulary size V, from **tokens**, the total number N of running words.

- Simple (unsmoothed) N-grams
  - Suppose we want to compute the probability of word *w* given its history *h* as *P(w|h)*. Suppose the word is *w* ="the" and its history is:

"*its water is so transparent that*"

This would be answering the question "Out of the times we saw the history h, how many times was it followed by the word w, as follows:

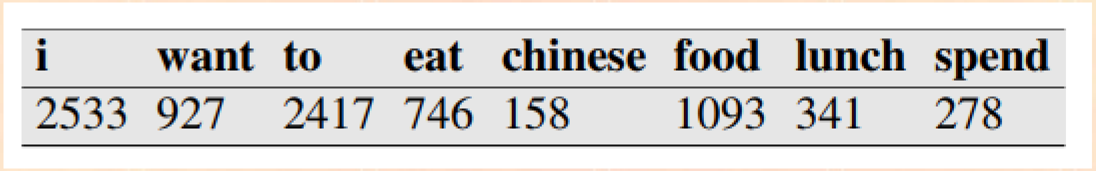
$$P(the|its\ water\ is\ so\ transparent\ that) = \frac{C(its\ water\ is\ so\ transparent\ that\ the)}{C(its\ water\ is\ so\ transparent\ that)}$$

Berkeley Restaurant Project Sentences

- *can you tell me about any good cantonese restaurants close by?*
- *mid priced thai food is what i'm looking for.*
- *tell me about Chez Panisse!*
- *can you give me a listing of the kinds of food that are available?*
- *i'm looking for a good place to eat breakfast.*
- *when is Caffe Venezia open during the day?*

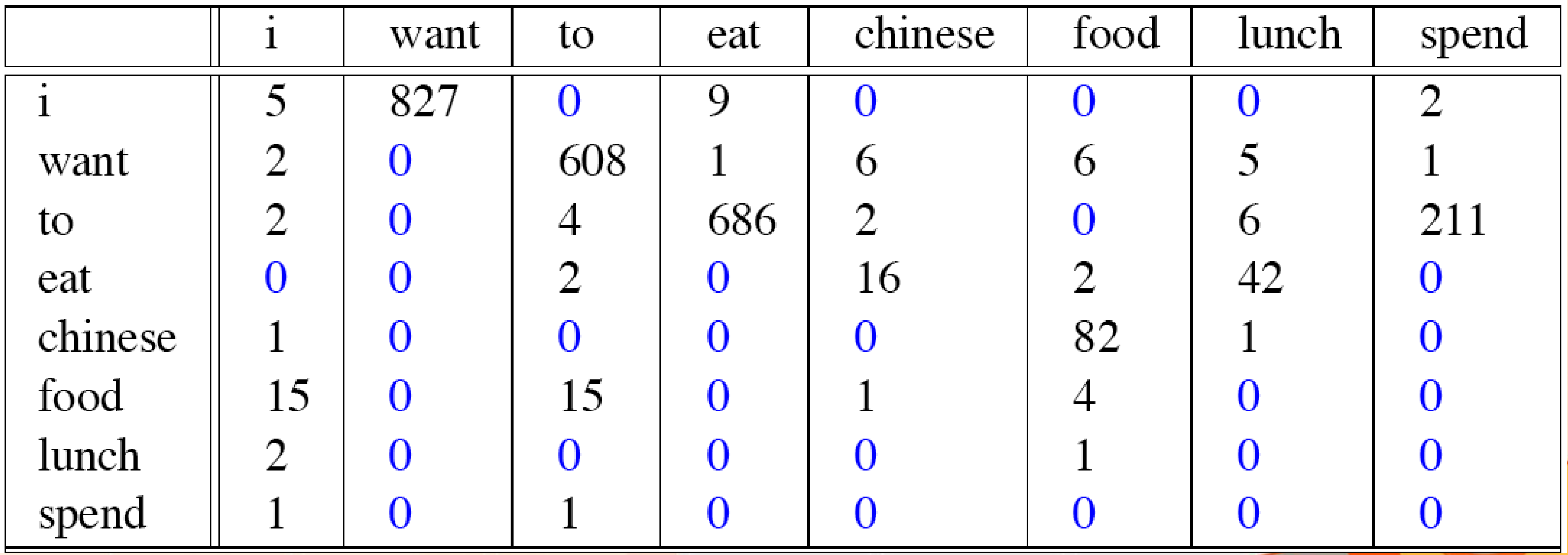
Bigram Counts

- *Out of 9222 sentences, the unigram counts are:*



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

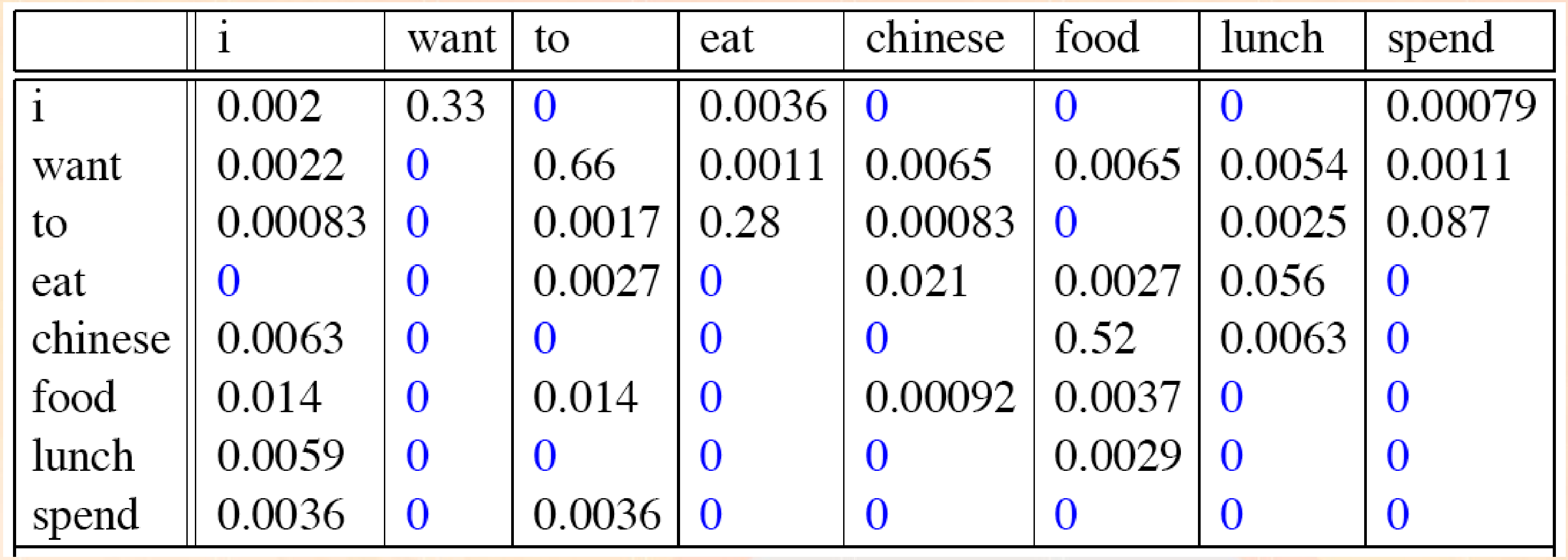
- *And the bigram counts are:*



	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

Bigram Probabilities

- *Divide bigram counts by prefix unigram counts to get probabilities.*



	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

Bigram Estimates of Sentence Probabilities

- *P(<s> I want english food </s>) =*
  *P(i|<s>)**
  *P(want|I)**
  *P(english|want)**
  *P(food|english)**
  *P(</s>|food)**
  =.000031

Kinds of Knowledge

- As crude as they are, *N*-gram probabilities capture a range of interesting facts about language.

- *P(english|want) = .0011*
- *P(chinese|want) = .0065*
- *P(to|want) = .66*
- *P(eat | to) = .28*
- *P(food | to) = 0*
- *P(want | spend) = 0*
- *P (i | <s>) = .25*

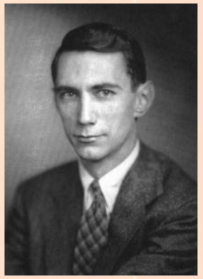
World knowledge

Syntax

Discourse

Shannon's Method

Assigning probabilities to sentences is all well and good, but it's not terribly illuminating . A more interesting task is to turn the model around and use it to generate random sentences that are like the sentences from which the model was derived.



- Generally attributed to Claude Shannon.

1. *Sample a random bigram (<s>, w) according to its probability*
2. *Now sample a random bigram (w, x) according to its probability*
   - *Where the prefix w matches the suffix of the first.*
3. *And so on until we randomly choose a (y, </s>).Then string the words together.*

<s> I

I want

want to

to eat

eat Chinese

Chinese food

food </s>

Some practical issues:

- *In practice it's more common to use* ***trigram models****, which condition on the previous two words rather than the previous word, or* ***4-gram*** *or even* ***5-gram*** *models, when there is sufficient training data.*
  - For example, to compute trigram probabilities at the very beginning of the sentence, we can use two pseudo-words for the first trigram (i.e., P(I|<s><s>)
- **Log Probabilities:** We always represent and compute language model probabilities in log format as log probabilities. *Why????*
- Since probabilities are (by definition) less than or equal to 1, the more probabilities we multiply together, the smaller the product becomes. Multiplying enough n-grams together would result in **numerical underflow**.
  - *By using log probabilities instead of raw probabilities, we get numbers that are not as small.*
  - *if we need to report them at the end; then we can just take the exponential of the logprob:*

$$p_1 \times p_2 \times p_3 \times p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4)$$

Evaluating *N*-Gram Models

- *Best evaluation for a language model*
  - *Put model A into an application*
    - *For example, a speech recognizer*
  - *Evaluate the performance of the application with model A*
  - *Put model B into the application and evaluate*
  - *Compare performance of the application with the two models*
  - ***Extrinsic evaluation***

Difficulty of extrinsic (in-vivo) evaluation of N-gram models

- *Extrinsic evaluation*
  - *This is really time-consuming*
  - *Can take days to run an experiment*
- *So*
  - *As a temporary solution, in order to run experiments*
  - *To evaluate N-grams we often use an **intrinsic** evaluation, an approximation called **perplexity***
  - *But perplexity is a poor approximation unless the test data looks **just** like the training data*
  - *So is **generally only useful in pilot experiments (generally is not sufficient to publish)***
  - ***But is helpful to think about a signal of the quality of the language model.***

Evaluation

- *How do we know if our models are any good?*
  - *And in particular, how do we know if one model is better than another.*
- *Well Shannon's game gives us an intuition.*
  - *The generated texts from the higher order models sure look better. That is, they sound more like the text the model was obtained from.*
  - *But what does that mean? Can we make that notion operational?*

Evaluation Methodology

- *Standard method*
  - *Train parameters of our model on a **training set**.*
  - *Look at the models performance on some held-out data*
    - *This is exactly what happens in the real world; we want to know how our model performs on data we haven't seen*
  - *So use a **test set**. A dataset which is different than our training set, but is drawn from the same source*
  - *Then we need an **evaluation metric** to tell us how well our model is doing on the test set.*
    - *One such metric is **perplexity** (to be introduced a bit later)*

Training and Test Sets

- The probabilities of the N-gram come from the corpus it is trained on.
- The parameters of any statistical model are trained on some data and then they are tested on a different data.
  - ❑ Training corpus
  - ❑ Testing corpus

❑ How do we create a training corpus and a testing corpus?

- ❖ Take a corpus – and divide it into a training set and a testing set.
- ❖ This training-and-testing paradigm can be used to evaluate different N-gram architectures.
  - ❖ Suppose we train two different N-gram models on the training set (e.g. bigram and trigram). We want to see which one performs better on the test set!
  - ❖ Which one better <u>models the test set</u>???

Held-out Sets and Development Sets

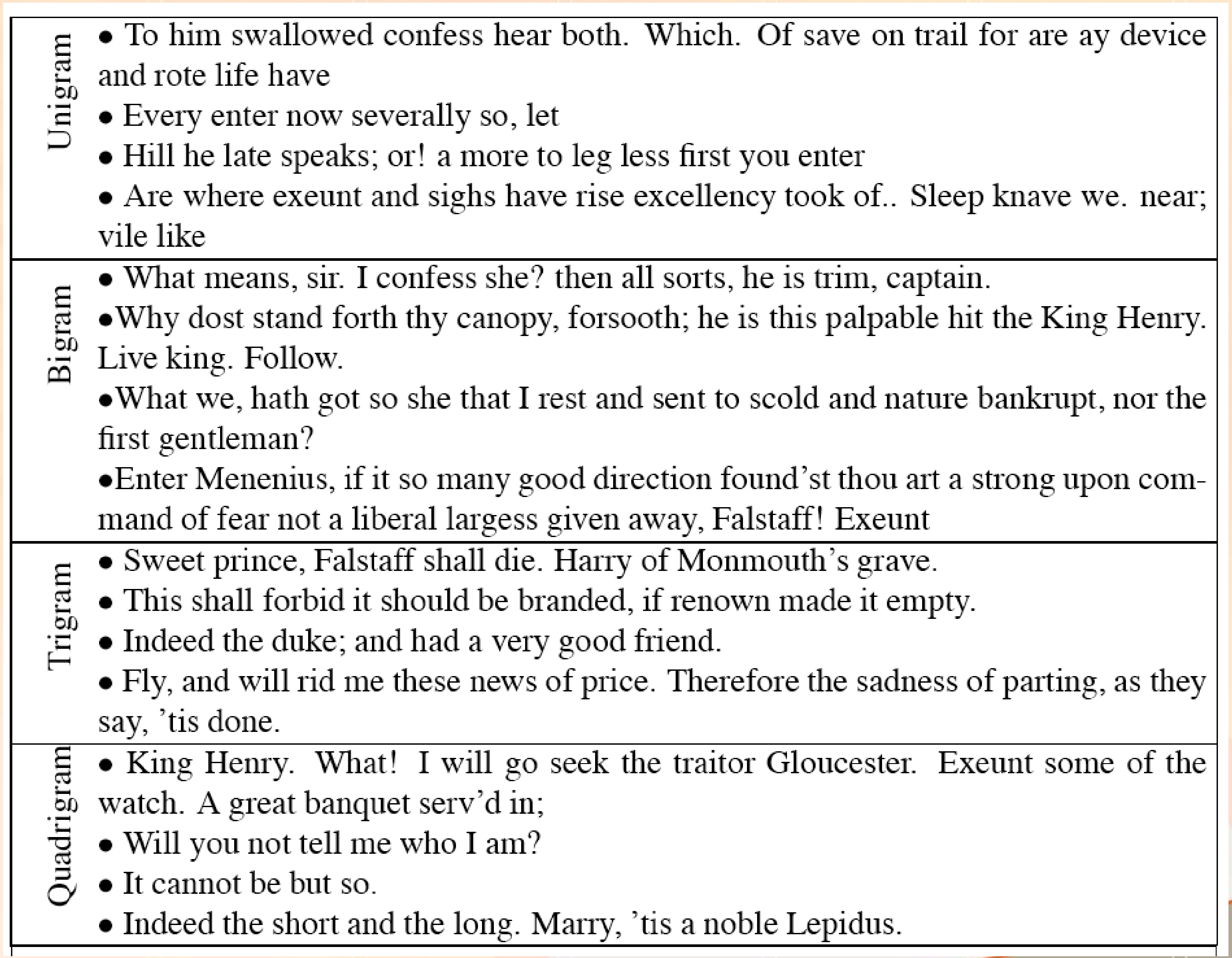
- To measure how well a statistical model matches a test corpus, we have a measure called **perplexity.**
- It is important not to let the test data (sentences) in the training data (sentences). Why? It the test data is mistakenly part of the training data- we shall assign it an artificially high probability. This is called **training on the test data.** Training on test data introduces a bias that makes probabilities look too high, and causes huge inaccuracies in perplexity.

- Sometimes we need an extra source of data to augment the training data. Such data is called **held-out set** – because it is hold out from the training data. It is used to set some parameters – e.g. interpolation.
- Sometimes we need to have multiple test sets. If we use a test set very often, we implicitly tune its charactersitics.
- Solution: We get a second test set. The first test set is called **development set or devset**.

Shakespeare as a Corpus

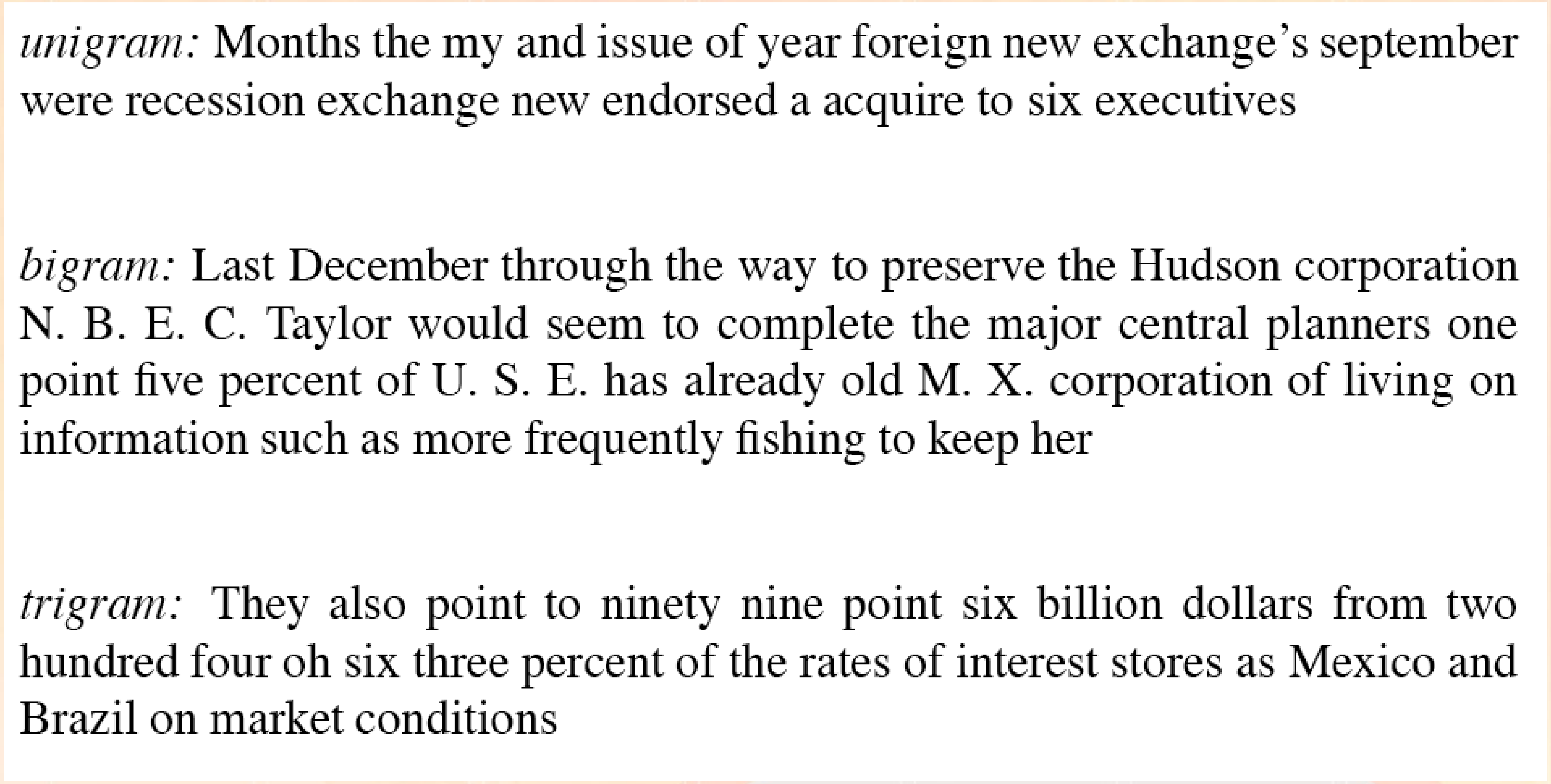
- *N=884,647 tokens, V=29,066*
- *Shakespeare produced 300,000 bigram types out of V^2= 844 million possible bigrams...*
    - *So, 99.96% of the possible bigrams were never seen (have zero entries in the table)*
    - *This is the biggest problem in language modeling; we'll come back to it.*
- *Quadrigrams are worse: What's coming out looks like Shakespeare because it* ***is*** *Shakespeare*

Shakespeare



Unigram	• To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have • Every enter now severally so, let • Hill he late speaks; or! a more to leg less first you enter • Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like
Bigram	• What means, sir. I confess she? then all sorts, he is trim, captain. •Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. •What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman? •Enter Menenius, if it so many good direction found'st thou art a strong upon command of fear not a liberal largess given away, Falstaff! Exeunt
Trigram	• Sweet prince, Falstaff shall die. Harry of Monmouth's grave. • This shall forbid it should be branded, if renown made it empty. • Indeed the duke; and had a very good friend. • Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.
Quadrigram	• King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; • Will you not tell me who I am? • It cannot be but so. • Indeed the short and the long. Marry, 'tis a noble Lepidus.

The Wall Street Journal is Not Shakespeare



unigram: Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

bigram: Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her

trigram: They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Perplexity Definition

- *Perplexity (PP) of a **LM** on a test set is the inverse probability of the test set, normalized by the number of words.*

$$\mathrm{PP}(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

- *Chain rule:*

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$$

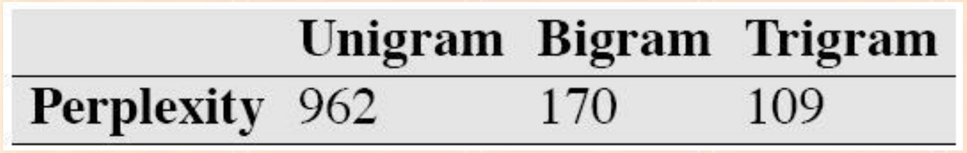
For bigrams:

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

- Minimizing perplexity is the same as maximizing probability
  - *The best language model is one that best predicts an unseen test set*

Lower perplexity means a better model

- *Training 38 million words, test 1.5 million words, WSJ*



	Unigram	Bigram	Trigram
Perplexity	962	170	109

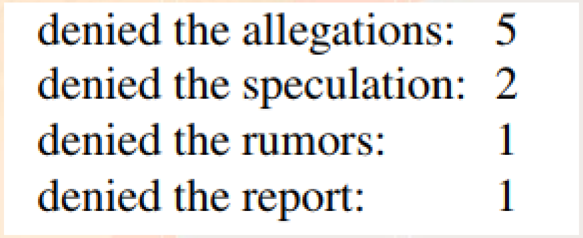
There is another way to think about perplexity: as the weighted average ***branching factor of a language****. The branching factor of a language is the number of possible next words that can follow any word.*

Unknown Words = Zero probabilities

- *Language models may still be subject to the problem of* ***sparsity****.*
  - *For any n-gram that occurred a sufficient number of times, we might have a good estimate of its probability. But because any corpus is limited, some perfectly acceptable English word sequences are bound to be missing from it.*
  - *That is, we'll have many cases of putative "zero probability n-grams" that should really have some non-zero probability.*

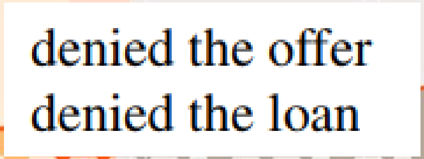
Zero Counts

- *Back to Shakespeare*
  - *Recall that Shakespeare produced 300,000 bigram types out of V^2= 844 million possible bigrams...*
  - *So, 99.96% of the possible bigrams were never seen (have zero entries in the table)*
  - *Does that mean that any sentence that contains one of those bigrams should have a probability of 0?*
- *Consider a trigram LM on WSJ Treebank, which words follow the bigram "denied the"?* 3-grams and their counts:



denied the allegations:	5
denied the speculation:	2
denied the rumors:	1
denied the report:	1

- *But 0-zounts for:*



denied the offer
denied the loan

Unknown words

- *Sometimes we have a language processing task in which we know all the words that can occur. In such a* ***closed vocabulary*** *system the test set can only contain words from this lexicon, and there will be no unknown words.*
  - *Reasonable assumption for speech recognition or machine translation, where we have a pronunciation dictionary or a phrase table that are fixed in advance, and so the language model can only use the words in that dictionary or phrase table.*
- *Othertimes we have to deal with words we haven't seen before, which we'll* ***unknown words****, or* ***out of vocabulary (OOV) words****.*
  - *How are we going to tackle them in the Language Model?*
  - *SOLUTION: smoothing!!!!*

Smoothing 1/2

- **IDEA:** *To keep a language model from assigning zero probability to unseen words/n-grams, we'll have to shave off a bit of probability mass from some more frequent words/n-grams and give it to those we've never seen.*

- **Smoothing** is the task of re-evaluating some of the zero-probability and low-probability estimations.

- **Solution**: modify the MLE for computing N-grams to always assign them nonzero values.

1. Laplace Smoothing
2. Good-Turing Discounting

Laplace (Add-One) Smoothing

The simplest way to do smoothing is to add 1 to all the n-gram counts, before we normalize them into probabilities. All the counts that used to be zero will now have a count of 1, the counts of 1 will be 2, and so on.

- Let's apply it to the unigram model:

$$P(w_x) = \frac{C(w_x)}{N}$$

C *– count (number) of words in a type*
N *– no. of word tokens*

- Add-one smoothing adds one to each count. Since there are V words in the vocabulary, and each one got incremented, we also need to adjust the denominator to take into account the extra V observations

$$P_{addone}(w_x) = \frac{C(w_x)+1}{N+V}$$

Adjusted Counts

$$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

- *IDEA: Instead of changing both the numerator and denominator, it is convenient to describe how a smoothing algorithm <u>affects the numerator</u>, by defining an adjusted count c $*$.*
- *Why??? This adjusted count is easier to compare directly with the MLE counts and can be turned into a probability like an MLE count by normalizing by N.*
- The **adjusted count** is:

$$c_i^* = (c+1)\frac{N}{N+V}$$

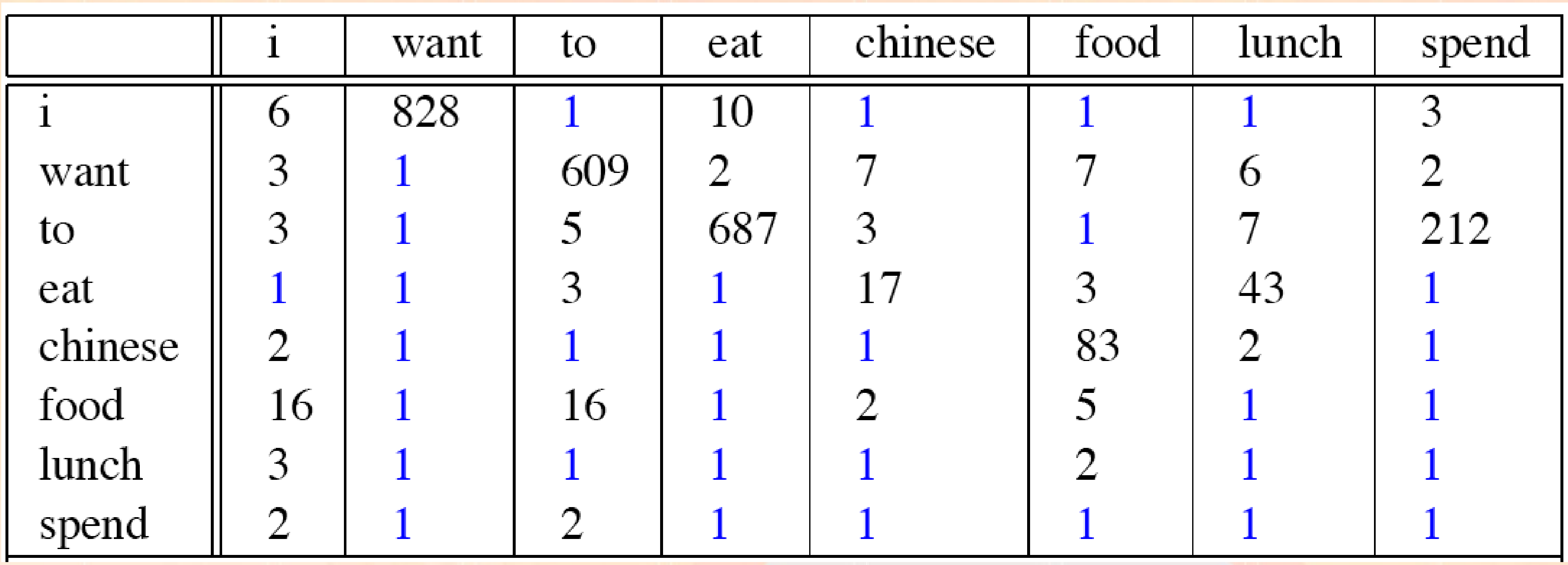
We can turn c into a probability p* by normalizing by N*

Discounting

- ***A related way to view smoothing :***
  - *as discounting (lowering) some non-zero counts in order to get the probability mass that will be assigned to the zero counts.*
  - *we can describe a smoothing algorithm in terms of a* ***relative discount*** d_c*, the ratio of the discounted counts to the original counts:*

$$d_c = \frac{c^*}{c}$$

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

Berkeley Restaurant Project

Laplace-Smoothed Bigram Probabilities

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

	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

Reconstituted Counts

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

	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

Big Change to the Counts!

- *C(count to) went from 608 to 238!*
- *P(to|want) from .66 to .26!*
- *Discount d= c*/c*
  - *d for "chinese food" =.10!!! A 10x reduction*
  - *Could use more fine-grained method (add-k):*

$$P^*_{\text{Add-k}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+k}{C(w_{n-1})+kV}$$

- *Despite its flaws Laplace (add-k) is however still used to smooth other probabilistic models in NLP, especially*
  - *For pilot studies*
  - *in domains where the number of zeros isn't so huge.*

Better Smoothing

- *The Laplace discounting/smoothing helps solve the problem of zero frequency n-grams. But there is an additional source of knowledge we can draw on.*
- ***IDEA**: If we are trying to compute $P(w_n|w_{n-2}w_{n-1})$ but we have no examples of a particular trigram $w_{n-2}w_{n-1}w_n$, we can instead estimate its probability by using the bigram probability $P(w_n|w_{n-1})$. Similarly, if we don't have counts to compute $P(w_n|w_{n-1})$, we can look to the unigram $P(w_n)$.*

➢ *sometimes using less context is a good thing, helping to generalize more for contexts that the model hasn't learned much about.*

Backoff and interpolation

There are two ways to use this n-gram "hierarchy":

- ❑ *Backoff*
- ❑ *Interpolation*

$$\begin{tikzcd}
w_{n-2}w_{n-1}w_n \arrow[d] \\
w_{n-1}w_n \arrow[d] \\
w_n
\end{tikzcd}$$

- *In backoff, we use the trigram if the evidence is sufficient, otherwise we use the bigram, otherwise the unigram. In other words, we only "<u>back off</u>" to a lower-order n-gram if we have zero evidence for a higher-order interpolation n-gram.*

- *By contrast, in interpolation, we always mix the probability estimates from all the n-gram estimators, weighing and combining the trigram, bigram, and unigram counts.*

Linear Interpolation

- *In simple linear interpolation, we combine different order n-grams by linearly interpolating all the models.*
  - *E.g. we estimate the trigram probability $P(w_n|w_{n-2}w_{n-1})$ by mixing together the unigram, bigram, and trigram probabilities, each weighted by a λ:*

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

$$\sum_i \lambda_i = 1$$

- *More sophisticated: each λ weight is computed by conditioning on the context:*

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

How to Set the Lambdas?

- *Use a **held-out, or development,** corpus*
- *Choose lambdas which maximize the probability of some held-out data*
  - *I.e. fix the N-gram probabilities*

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

  - *Then search for lambda values that when plugged into the equation give largest probability for held-out set*

- *Can use EM to do this search*

Backoff IDEA

- *In a backoff n-gram model, if the n-gram we need has zero counts, we approximate it by backing off to the (n-1)-gram. We continue backing off until we reach a history that has some counts.*

- *In order for a backoff model to give a correct probability distribution, we have to <u>discount the higher-order n-grams</u> to save some probability mass for the lower order n-grams. If the higher-order n-grams aren't discounted, then as soon as we replaced an n-gram which has zero probability with a lower-order n-gram, we would be adding probability mass, and the total probability assigned to all possible strings by the language model would be greater than 1!*

Katz Backoff

- Katz backoff: discounting is implemented by using discounted probabilities P* rather than MLE probabilities!

$$P_{katz}(w_n \mid w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n \mid w_{n-N+1}^{n-1}) & \text{if } C(w_{n-N+1}^{n-1}) > 0 \\ \alpha(w_{n-N+1}^{n-1}) P_{katz}(w_n \mid w_{n-N+2}^{n-1}) & \text{otherwise} \end{cases}$$

The discounted probability $\longrightarrow$

$$P^*(w_n \mid w_{n-N+1}^{n-1}) = \frac{c^*(w_{n-N+1}^{n})}{c(w_{n-N+1}^{n-1})}$$

Katz Backoff

$$P_{\text{katz}}(w_n|w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n|w_{n-N+1}^{n-1}), & \text{if } C(w_{n-N+1}^{n}) > 0 \\ \alpha(w_{n-N+1}^{n-1})P_{\text{katz}}(w_n|w_{n-N+2}^{n-1}), & \text{otherwise.} \end{cases}$$

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

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

Why discounts P* and alpha?

- *MLE probabilities sum to 1*

$$\boxed{\sum_i P(w_i \mid w_j w_k) = 1}$$

- *So if we used MLE probabilities but backed off to lower order model when MLE prob is zero*
  - ❑*We would be adding extra probability mass*
  - ❑*And total probability would be greater than 1*

$$\boxed{P^*(w_n \mid w_{n-N+1}^{n-1}) = \frac{c^*(w_{n-N+1}^{n})}{c(w_{n-N+1}^{n-1})}}$$

Discussion

- Why do we need discounts and the α values?

$$P_{katz}(w_n \mid w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n \mid w_{n-N+1}^{n-1}) & \text{if } \mathrm{C}(\mathrm{w}_{\text{n-N+1}}^{\text{n-1}}) > 0 \\ \alpha(w_{n-N+1}^{n-1}) P_{katz}(w_n \mid w_{n-N+2}^{n-1}) & \text{otherwise} \end{cases}$$

- Why couldn't we just have MLE probabilities and no weights?

Answer: without discounts and the α weights, the result of P_{katz} would not be a probability.

$$\sum_i P(w_i \mid w_j w_k) = 1$$

But if we use MLE probabilities and backoff to a lower order model when the MLE is zero – we add extra probability mass into the equation – and the total probability is >1 !!!

$$P^*(w_n \mid w_{n-N+1}^{n-1}) = \frac{c^*(w_{n-N+1}^{n})}{c(w_{n-N+1}^{n-1})} < \frac{c(w_{n-N+1}^{n})}{c(w_{n-N+1}^{n-1})}$$

Computing α 1/3

- *We still need to know how to compute α*

$$P_{katz}(w_n \mid w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n \mid w_{n-N+1}^{n-1}) & \text{if } C(w_{n-N+1}^{n-1}) > 0 \\ \alpha(w_{n-N+1}^{n-1}) P_{katz}(w_n \mid w_{n-N+2}^{n-1}) & \text{otherwise} \end{cases}$$

What is the role of α ?

It passes *the left-over probability mass* to the lower-order N-grams.

Let us represent the total amount of the left-over probability mass by a function β which is a function of the N-1-gram context:

$$\beta(w_{n-N+1}^{n-1}) = 1 - \sum_{w_n : c(w_{n-N+1}^{n}) > 0} P^*(w_n \mid w_{n-N+1}^{n-1})$$

Computing α 3/3

- *The trigram version of the backoff*

$$P_{katz}(w_i \mid w_{i-2}w_{i-1}) = \begin{cases} P^*(w_i \mid w_{i-2}w_{i-1}) & \text{if } \mathrm{C}(\mathrm{w}_{\text{i-2}}w_{i-1}w_i) > 0 \\ \alpha(w_{i-1}w_i)P^*(w_i \mid w_{i-1}) & \text{else if } \mathrm{C}(\mathrm{w}_{\text{i-1}}w_i) > 0 \\ \alpha(w_i)P^*(w_i) & \textit{otherwise} \end{cases}$$

Computing α 2/3

$$\beta(w_{n-N+1}^{n-1}) = 1 - \sum_{w_n : c(w_{n-N+1}^{n})>0} P^*(w_n \mid w_{n-N+1}^{n-1})$$

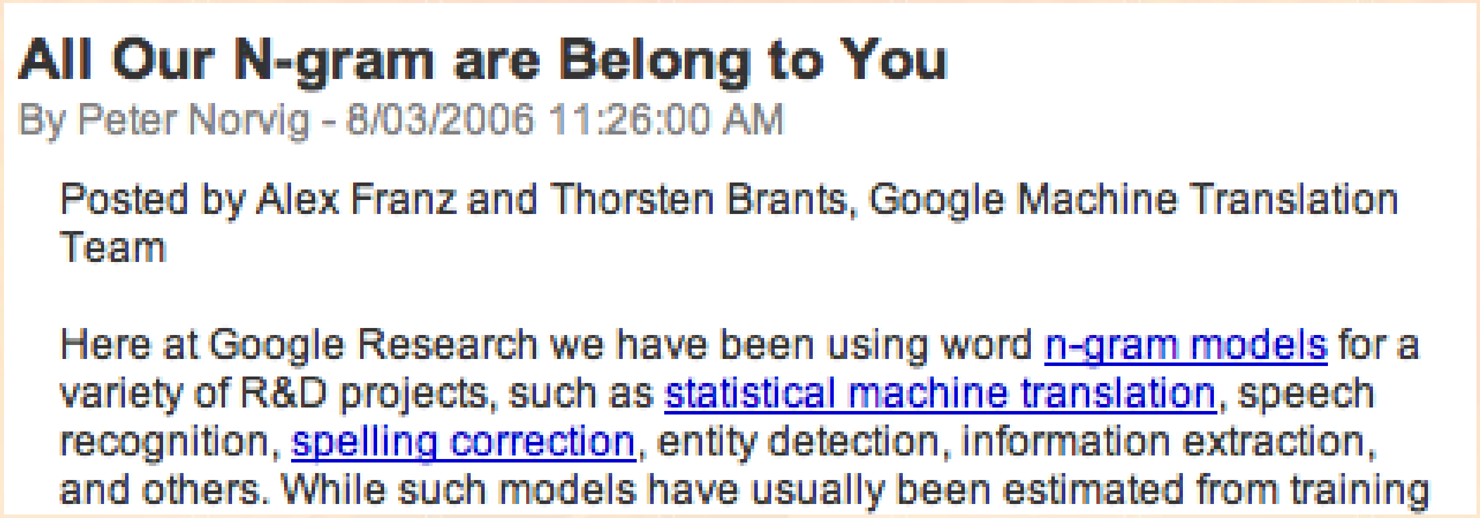
- *This is the total probability mass that we can distribute to all the N-1 gram. Thus:*

$$\alpha(w_{n-N+1}^{n-1}) = \frac{\beta(w_{n-N+1}^{n-1})}{\sum_{w_n : c(w_{n-N+1}^{n})=0} P(w_n \mid w_{n-N+2}^{n-1})}$$
$$= \frac{1 - \sum_{w_n : c(w_{n-N+1}^{n})>0} P^*(w_n \mid w_{n-N+1}^{n-1})}{1 - \sum_{w_n : c(w_{n-N+1}^{n})>0} P^*(w_n \mid w_{n-N+2}^{n-1})}$$

Note: α is a function of the preceding word. The probability mass which is reassigned to lower n-grams is recomputed for each N-1 gram that occur in each N-gram.

$$\sum_{w_n : c(w_{n-N+1}^{n})>0} P^*(w_n \mid w_{n-N+2}^{n-1}) + \sum_{w_n : c(w_{n-N+1}^{n})=0} P(w_n \mid w_{n-N+2}^{n-1}) = 1$$

Google N-Gram Release

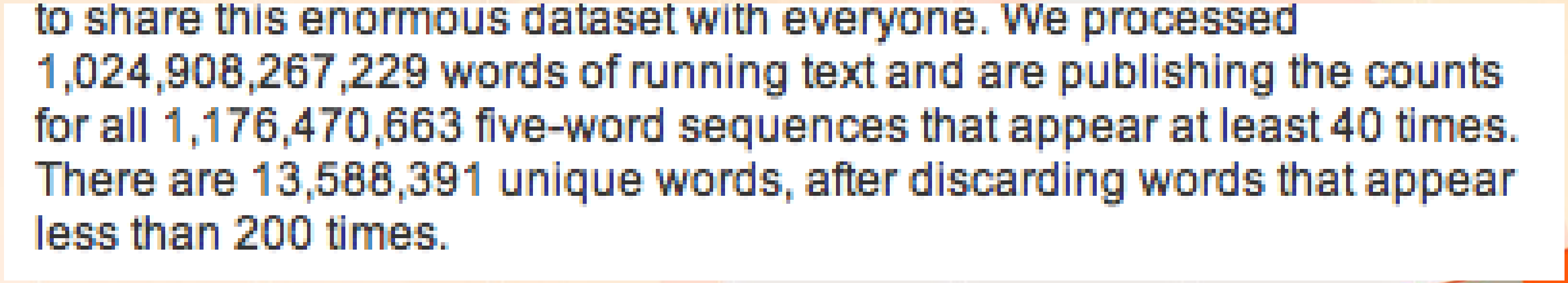


All Our N-gram are Belong to You

By Peter Norvig - 8/03/2006 11:26:00 AM

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects, such as statistical machine translation, speech recognition, spelling correction, entity detection, information extraction, and others. While such models have usually been estimated from training



to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

Google Caveat

- *Remember the lesson about test sets and training sets... Test sets should be similar to the training set (drawn from the same distribution) for the probabilities to be meaningful.*
- *So... The Google corpus is fine if your application deals with arbitrary English text on the Web.*
- *If not then a smaller domain specific corpus is likely to yield better results.*

Summary

❑ *Introduced language modeling and the n-gram*

- *Language models offer a way to assign a probability to a sentence or other sequence of words, and to predict a word from preceding words.*
- *n-grams are Markov models that estimate words from a fixed window of previous words. n-gram probabilities can be estimated by counting in a corpus and normalizing (the maximum likelihood estimate).*
- *n-gram language models are evaluated extrinsically in some task, or intrinsically using perplexity.*
- *Smoothing algorithms provide a more sophisticated way to estimate the probability of n-grams.*