Detection and Correction of Spelling Errors

Based on slides of Michael Collins, Dan Jurafsky, Dan Klein, Chris Manning, Luke Zettlemoyer

Broder Problems

- Non-word error detection: graffe and giraffe
- Isolated-word error correction
- Context –dependent error detection and correction
  - E.g. dessert for desert, or piece for peace

Minimum Edit Distance

Operations:

- insertion
- deletion
- substitution

$$D[i,j] = \min \begin{cases} D[i-1,j] + \text{del-cost}(source[i]) \\ D[i,j-1] + \text{ins-cost}(target[j]) \\ D[i-1,j-1] + \text{sub-cost}(source[i], target[j]) \end{cases}$$

Or

$$D[i,j] = \min \begin{cases} D[i-1,j] + 1 \\ D[i,j-1] + 1 \\ D[i-1,j-1] + \begin{cases} 2; & \text{if } source[i] \neq target[j] \\ 0; & \text{if } source[i] = target[j] \end{cases} \end{cases}$$

function MIN-EDIT-DISTANCE(*source*, *target*) **returns** *min-distance*

$n \leftarrow$ LENGTH(*source*)
$m \leftarrow$ LENGTH(*target*)
Create a distance matrix *D[n+1,m+1]*

\# *Initialization: the zeroth row and column is the distance from the empty string*
$D[0,0] = 0$
for each row i **from** 1 **to** n **do**
  $D[i,0] \leftarrow D[i\text{-}1,0] + del\text{-}cost(source[i])$
for each column j **from** 1 **to** m **do**
  $D[0,j] \leftarrow D[0,j\text{-}1] + ins\text{-}cost(target[j])$

\# *Recurrence relation:*
for each row i **from** 1 **to** n **do**
  for each column j **from** 1 **to** m **do**
    $D[i,j] \leftarrow$ MIN($D[i-1,j] + del\text{-}cost(source[i])$,
      $D[i-1,j-1] + sub\text{-}cost(source[i], target[j])$,
      $D[i,j-1] + ins\text{-}cost(target[j])$)
\# *Termination*
return $D[n,m]$

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1] + \begin{cases} 2; & \text{if } source[i] \neq target[j] \\ 0; & \text{if } source[i] = target[j] \end{cases} \end{cases}$$

Src\Tar	#	e	x	e	c	u	t	i	o	n
#	0	1	2	3	4	5	6	7	8	9
i	1	2	3	4	5	6	7	6	7	8
n	2	3	4	5	6	7	8	7	8	7
t	3	4	5	6	7	8	7	8	9	8
e	4	3	4	5	6	7	8	9	10	9
n	5	4	5	6	7	8	9	10	11	10
t	6	5	6	7	8	9	8	9	10	11
i	7	6	7	8	9	10	9	8	9	10
o	8	7	8	9	10	11	10	9	8	9
n	9	8	9	10	11	12	11	10	9	8

Alignment

	#	e	x	e	c	u	t	i	o	n
#	0	← 1	← 2	← 3	← 4	← 5	← 6	← 7	← 8	← 9
i	↑ **1**	↖←↑ 2	↖←↑ 3	↖←↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖ 6	← 7	← 8
n	↑ 2	↖←↑ **3**	↖←↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ 8	↑ 7	↖←↑ 8	↖ 7
t	↑ 3	↖←↑ 4	↖←↑ **5**	↖←↑ 6	↖←↑ 7	↖←↑ 8	↖ 7	←↑ 8	↖←↑ 9	↑ 8
e	↑ 4	↖ 3	← 4	↖← **5**	← **6**	← 7	←↑ 8	↖←↑ 9	↖←↑ 10	↑ 9
n	↑ 5	↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ **8**	↖←↑ 9	↖←↑ 10	↖←↑ 11	↖↑ 10
t	↑ 6	↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ 8	↖←↑ 9	↖ **8**	← 9	← 10	←↑ 11
i	↑ 7	↑ 6	↖←↑ 7	↖←↑ 8	↖←↑ 9	↖←↑ 10	↑ 9	↖ **8**	← 9	← 10
o	↑ 8	↑ 7	↖←↑ 8	↖←↑ 9	↖←↑ 10	↖←↑ 11	↑ 10	↑ 9	↖ **8**	← 9
n	↑ 9	↑ 8	↖←↑ 9	↖←↑ 10	↖←↑ 11	↖←↑ 12	↑ 11	↑ 10	↑ 9	↖ **8**

Language Modeling

Introduction to N-grams

Probabilistic Language Models

Today's goal: assign a probability to a sentence

- Machine Translation:
  - P(**high** winds tonite) > P(**large** winds tonite)
- Spell Correction
  - The office is about fifteen **minuets** from my house
  - P(about fifteen **minutes** from) > P(about fifteen **minuets** from)
- Speech Recognition
  - P(I saw a van) >> P(eyes awe of an)
- + Summarization, question-answering, etc., etc.!!

Why?

The Language Modeling Problem

- Setup: Assume a (finite) vocabulary of words

  V ={the, a, man, telescope, two, Madrid,…}

- We can construct an (infinite) set of strings

  V+ ={the, a, the man, the a, the man with the telescope,…}

- Data: given a training set of example sentences $x \in V^+$
- Problem: estimate a probability distribution

$$\sum_{x \in \mathcal{V}^\dagger} p(x) = 1$$

and $p(x) \geq 0$ for all $x \in \mathcal{V}^\dagger$

$p(\text{the}) = 10^{-12}$

$p(\text{a}) = 10^{-13}$

$p(\text{the fan}) = 10^{-12}$

$p(\text{the fan saw Beckham}) = 2 \times 10^{-8}$

$p(\text{the fan saw saw}) = 10^{-15}$

$\ldots$

The Noisy-Channel Model

- We want to predict a sentence given acoustics:

$$w^* = \arg\max_w P(w|a)$$

- The noisy channel approach:

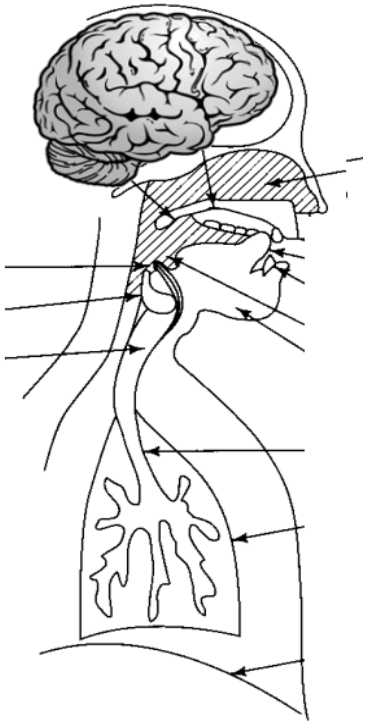
$$w^* = \arg\max_w P(w|a)$$

$$= \arg\max_w P(a|w)P(w)/P(a)$$

$$\propto \arg\max_w P(a|w)P(w)$$

Acoustic model: Distributions over acoustic waves given a sentence

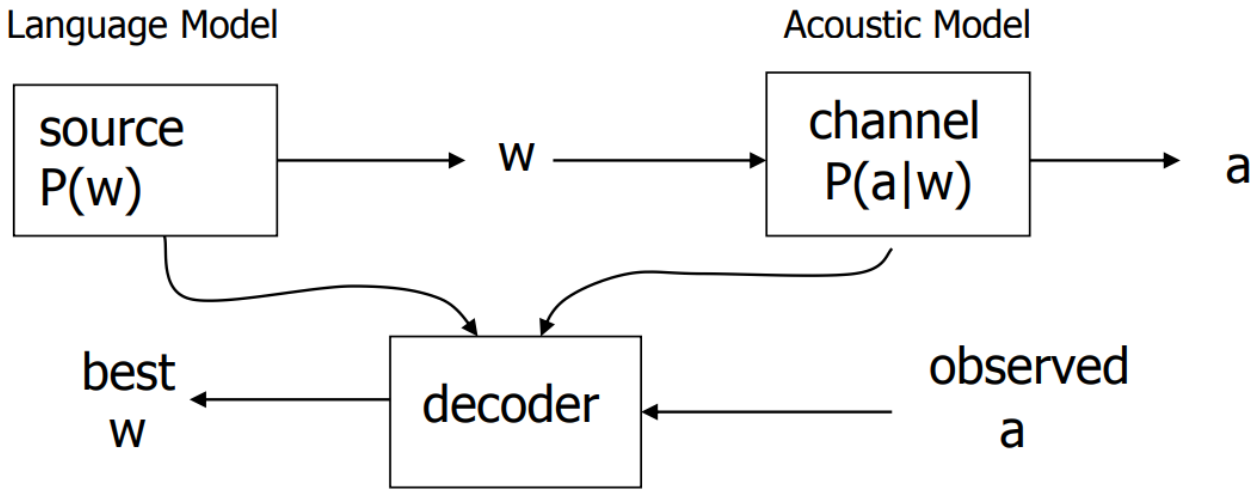
Language model: Distributions over sequences of words (sentences)



Acoustically Scored Hypotheses

the station signs are in deep in english	-14732
the stations signs are in deep in english	-14735
the station signs are in deep into english	-14739
the station 's signs are in deep in english	-14740
the station signs are in deep in the english	-14741
the station signs are indeed in english	-14757
the station 's signs are indeed in english	-14760
the station signs are indians in english	-14790
the station signs are indian in english	-14799
the stations signs are indians in english	-14807
the stations signs are indians and english	-14815

ASR system components



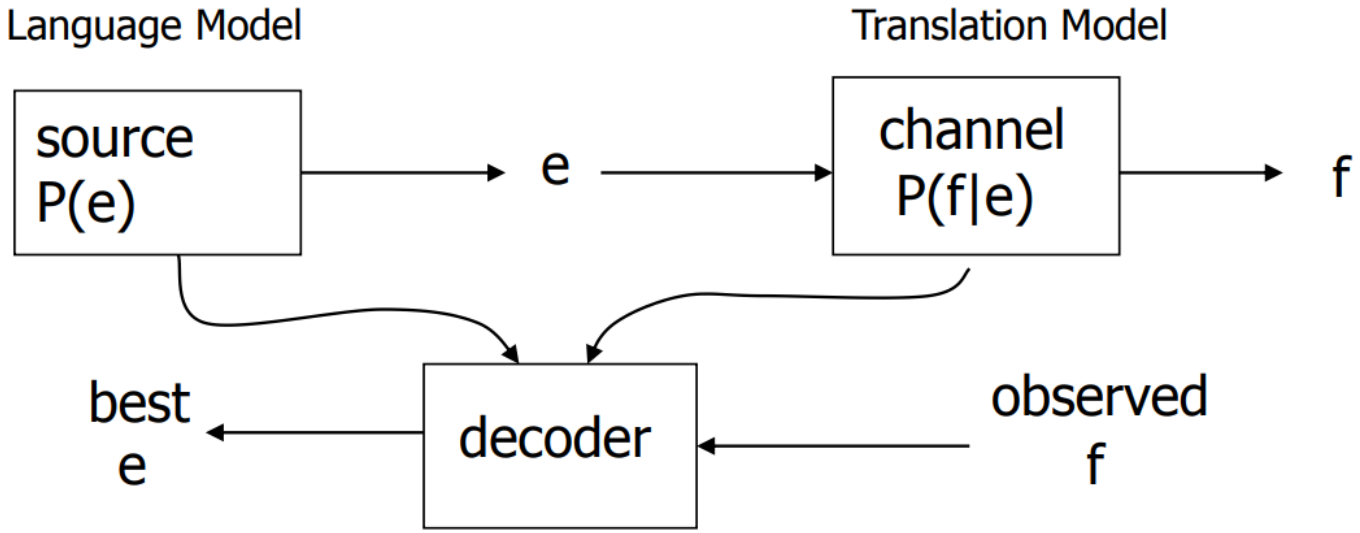
$$\underset{w}{\operatorname{argmax}}\, P(w|a) = \underset{w}{\operatorname{argmax}}\, P(a|w)P(w)$$

Translation: Codebreaking?

" Also knowing nothing official about, but having guessed and inferred considerable about, the powerful new mechanized methods in cryptography—methods which I believe succeed even when one does not know what language has been coded—one naturally wonders if the problem of translation could conceivably be treated as a problem in cryptography. When I look at an article in Russian, I say: 'This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode.' "

Warren Weaver (1955:18, quoting a letter he wrote in 1947)

MT System Components



$$\operatorname*{argmax}_{e} P(e|f) = \operatorname*{argmax}_{e} P(f|e)P(e)$$

Probabilistic Language Modeling

Goal: compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1,w_2,w_3,w_4,w_5 \ldots w_n)$$

Related task: probability of an upcoming word:

$$P(w_5 | w_1,w_2,w_3,w_4)$$

A model that computes either of these:

$P(W)$ or $P(w_n | w_1,w_2 \ldots w_{n-1})$ is called a **language model**.

Better: **the grammar** But **language model** or **LM** is standard

How to compute P(W)

How to compute this joint probability:

- P(its, water, is, so, transparent, that)

Intuition: let's rely on the Chain Rule of Probability

Reminder: The Chain Rule

Recall the definition of conditional probabilities

p(B|A) = P(A,B)/P(A) Rewriting: **P(A,B) = P(A)P(B|A)**

More variables:

$P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$

The Chain Rule 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 applied to compute joint probability of words in sentence

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_1 w_2 \ldots w_{i-1})$$

P("its water is so transparent") =

P(its) × P(water|its) × P(is|its water)

× P(so|its water is) × P(transparent|its water is so)

How to estimate these probabilities

Could we just count and divide?

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

We'll never see enough data for estimating these

Markov Assumption


Andrei Markov

Simplifying assumption:

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

Or maybe

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

Markov Assumption

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i \mid w_{i-k} \ldots w_{i-1})$$

In other words, we approximate each component in the product

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-k} \ldots w_{i-1})$$

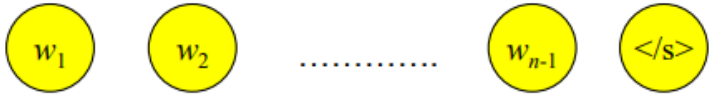
Simplest case: Unigram model

Simplest case: unigrams

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

Generative process: pick a word, pick a word, … until you pick </s>

Graphical model:



Examples:

- `fifth, an, of, futures, the, an, incorporated, a, a, the, inflation, most, dollars, quarter, in, is, mass`
- `thrift, did, eighty, said, hard, 'm, july, bullish`
- `that, or, limited, the`

Big problem with unigrams: P(the the the the) >> P(I like ice cream)!

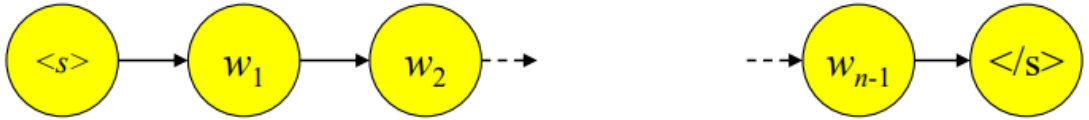
Bigram Model

Conditioned on previous single word

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

Generative process: pick <s>, pick a word conditioned on previous one, repeat until to pick </s>

Graphical model:



Examples:

- texaco, rose, one, in, this, issue, is, pursuing, growth, in, a, boiler, house, said, mr., gurria, mexico, 's, motion, control, proposal, without, permission, from, five, hundred, fifty, five, yen
- outside, new, car, parking, lot, of, the, agreement, reached
- this, would, be, a, record, november

N-gram models

We can extend to trigrams, 4-grams, 5-grams

In general this is an insufficient model of language

- because language has **long-distance dependencies**:

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

But we can often get away with N-gram models

Language Modeling

Estimating N-gram Probabilities

Estimating bigram probabilities

The Maximum Likelihood Estimate

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

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

An 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>

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

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

More examples:
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

Raw bigram counts

Out of 9222 sentences

	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

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

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

What kinds of knowledge?

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

Practical Issues

We do everything in log space

- Avoid underflow
- (also adding is faster than multiplying)

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

Language Modeling Toolkits

SRILM

- http://www.speech.sri.com/projects/srilm/

KenLM

- https://kheafield.com/code/kenlm/

Google N-Gram Release, August 2006

AUG
3

All Our N-gram are Belong to You

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,

...

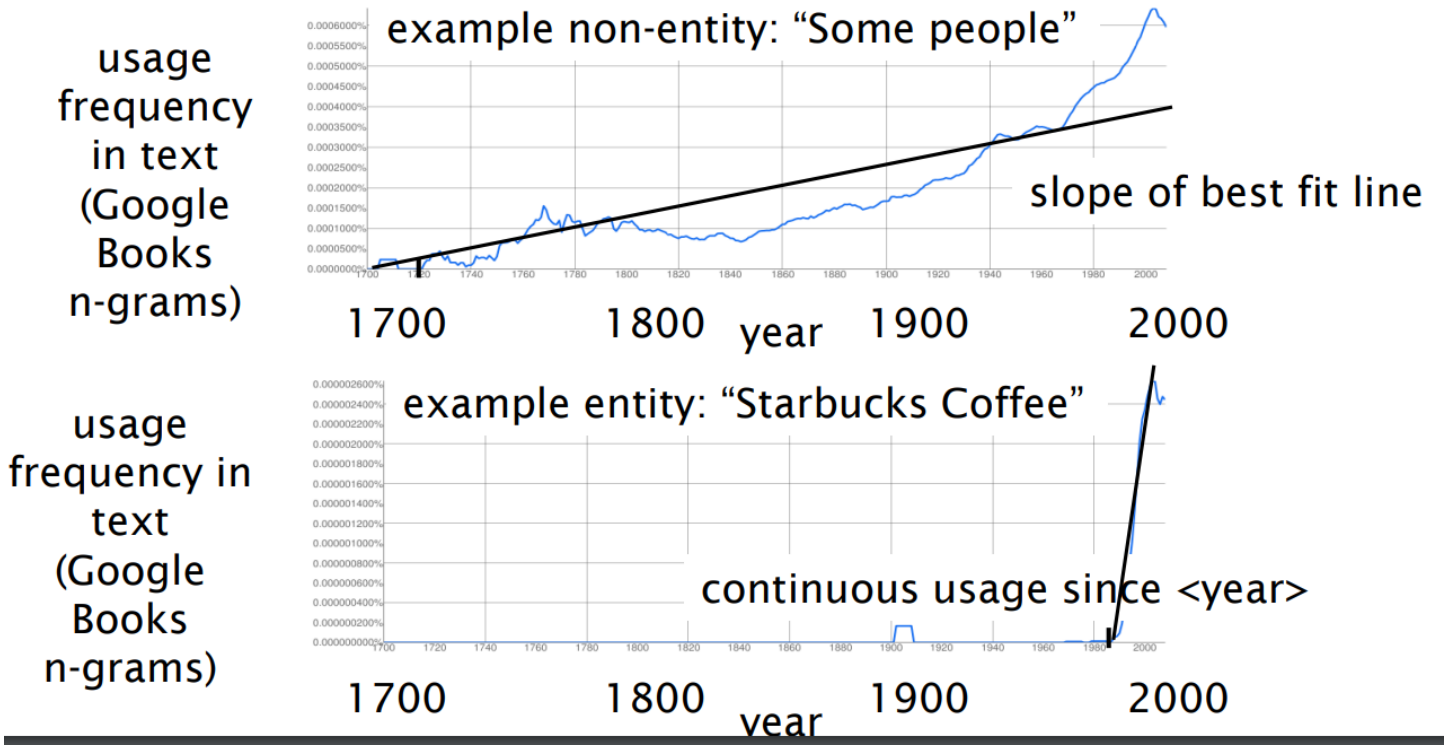
That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than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

Google N-Gram Release

```
serve as the incoming 92
serve as the incubator 99
serve as the independent 794
serve as the index 223
serve as the indication 72
serve as the indicator 120
serve as the indicators 45
serve as the indispensable 111
serve as the indispensible 40
serve as the individual 234
```

http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html

Temporally-Aware Features



Google Book N-grams

http://ngrams.googlelabs.com/

Language Modeling

Evaluation and Perplexity

Growth of parameters

	Model	Parameters if vocab 20,000
n=1	unigram	20000
n=2	bigram	$20000^2 = 400$ million
n=3	trigram	$20000^3 = 8$ billion
n=4	4-gram, fourgram	1.6×10^{17}

Evaluation: How good is our model?

Does our language model prefer good sentences to bad ones?

- Assign higher probability to “real” or “frequently observed” sentences
  - Than “ungrammatical” or “rarely observed” sentences?

We train parameters of our model on a **training set**.

We test the model’s performance on data we haven’t seen.

- A **test set** is an unseen dataset that is different from our training set, totally unused.
- An **evaluation metric** tells us how well our model does on the test set.

Extrinsic evaluation of N-gram models

Best evaluation for comparing models A and B

- Put each model in a task
  - spelling corrector, speech recognizer, MT system
- Run the task, get an accuracy for A and for B
  - How many misspelled words corrected properly
  - How many words translated correctly
- Compare accuracy for A and B

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

Extrinsic evaluation

- Time-consuming; can take days or weeks

So

- Sometimes use **intrinsic** evaluation: **perplexity**
- Bad approximation
  - unless the test data looks **just** like the training data
  - So **generally only useful in pilot experiments**
- But is helpful to think about.

Intuition of Perplexity

The **Shannon Game**:

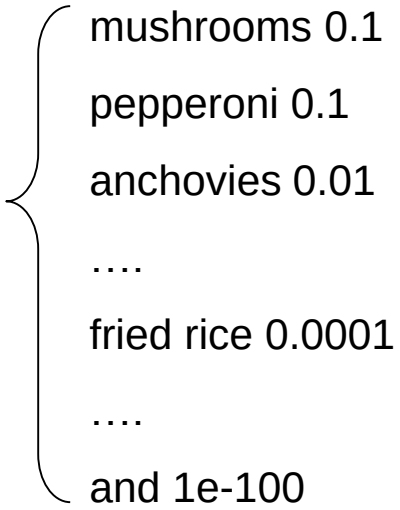
- How well can we predict the next word?

  I always order pizza with cheese and ____

  The 33rd President of the US was ____

  I saw a ____

- Unigrams are terrible at this game. (Why?)

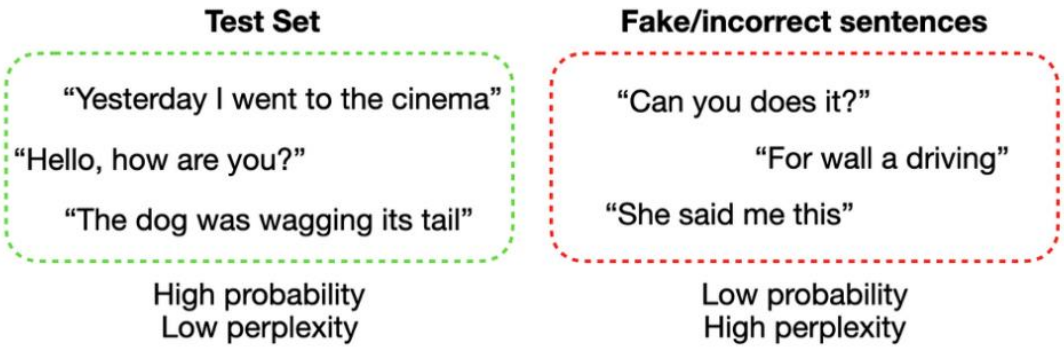


A better model of a text

- is one which assigns a higher probability to the word that actually occurs

Intuition of Perplexity

Intuitively, if a model assigns a high probability to the test set, it means that it is **not surprised** to see it (it's not *perplexed* by it), which means that it has a good understanding of how the language works.



Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest P(sentence)

Perplexity is the inverse probability of the test set, normalized by the number of words:

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

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

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

Normalization

- Datasets can have **varying numbers of sentences**, and sentences can have varying numbers of words.
- Adding more sentences introduces more uncertainty, so if other things are being equal- a larger test set is likely to have a lower probability than a smaller one.
- Ideally, we'd like to have a metric that is independent of the size of the dataset.
- We could obtain this by **normalising** the probability of the test set **by the total number of words**, which would give us a **per-word measure**.

Normalization

$$P(W) = P(w_1, w_2, \ldots, w_N) = P(w_1)P(w_2)\ldots P(w_N) = \prod_{i=1}^{N} P(w_i)$$

$$\ln(P(W)) = \ln\left(\prod_{i=1}^{N} P(w_i)\right) = \sum_{i=1}^{N} \ln P(w_i)$$

Sum: divide by N

Product:??

$$\frac{\ln(P(W))}{N} = \frac{\sum_{i=1}^{N} \ln P(w_i)}{N}$$

$$e^{\frac{\ln(P(W))}{N}} = e^{\frac{\sum_{i=1}^{N} \ln P(w_i)}{N}}$$

$$\left(e^{\ln(P(W))}\right)^{\frac{1}{N}} = \left(e^{\sum_{i=1}^{N} \ln P(w_i)}\right)^{\frac{1}{N}}$$

$$P(W)^{\frac{1}{N}} = \left(\prod_{i=1}^{N} P(w_i)\right)^{\frac{1}{N}}$$

Test dataset

W = (<s>, This, is, the, first, sentence, . ,<\s>,<s>,This, is, the, second, one, . ,<\s>)

N=16

The Shannon Game intuition for perplexity

From Josh Goodman

Perplexity is weighted equivalent branching factor

How hard is the task of recognizing digits ‘0,1,2,3,4,5,6,7,8,9’

- Perplexity 10

How hard is recognizing (30,000) names

- Perplexity = 30,000

Let's imagine a call-routing phone system gets 120K calls and has to recognize

- "Operator" (let's say this occurs 1 in 4 calls)
- "Sales" (1in 4)
- "Technical Support" (1 in 4)
- 30,000 different names (each name occurring 1 time in the 120K calls)
- What is the perplexity?

Perplexity as (weighted) branching factor

Let's suppose a sentence consisting of random digits

What is the perplexity of this sentence according to a model that assign P=1/10 to each digit?

$$\begin{aligned} \mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \left(\frac{1}{10}^{N}\right)^{-\frac{1}{N}} \\ &= \frac{1}{10}^{-1} \\ &= 10 \end{aligned}$$

Weighted branching factor: rolling a die

- A regular die has 6 sides, so the **branching factor** of the die is 6. The branching factor simply indicates **how many possible outcomes** there are whenever we roll.

A fair die

Outcome	1	2	3	4	5	6
Probability	1/6	1/6	1/6	1/6	1/6	1/6

Branching factor: **6**

Weighted branching factor: rolling a die

- Let's say we train our model on this fair die, and the model learns that each time we roll there is a 1/6 probability of getting any side.
- Then let's say we create a test set by rolling the die 10 more times and we obtain the (highly unimaginative) sequence of outcomes T = {1, 2, 3, 4, 5, 6, 1, 2, 3, 4}. What's the perplexity of our model on this test set?

$$PP(T) = \frac{1}{\left(\left(\frac{1}{6}\right)^{10}\right)^{\frac{1}{10}}} = 6$$

Perplexity matches branching factor

Weighted branching factor: rolling a die

- Let's say we have an **unfair die**, which rolls a 6 with a probability of 7/12, and all the other sides with a probability of 1/12 each.
- We again train a model on a training set created with this unfair die so that it will learn these probabilities.
- We then create a new test set T by rolling the die 12 times: we get a 6 on 7 of the rolls, and other numbers on the remaining 5 rolls. What's the perplexity now?

$$PP(T) = \frac{1}{\left(\left(\frac{7}{12}\right)^7 \cdot \left(\frac{1}{12}\right)^5\right)^{\frac{1}{12}}} = 3.9 \approx 4$$

- **The perplexity is lower**.
- This is because our model now knows that rolling a 6 is more probable than any other number, so it's less "surprised" to see one, and since there are more 6s in the test set than other numbers, the overall "surprise" associated with the test set is lower.
- The *branching factor* is still 6, because all 6 numbers are still possible options at any roll.

Lower perplexity = better model

Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Language Modeling

Generalization and zeros

The Shannon Visualization Method

Choose a random bigram

(<s>, w) according to its probability

Now choose a random bigram
(w, x) according to its probability

And so on until we choose </s>

Then string the words together

<s> I
I want
want to
to eat
eat Chinese
Chinese food
food </s>

I want to eat Chinese food

Approximating Shakespeare

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; –It cannot be but so.

Shakespeare as 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)

Quadrigrams worse: What's coming out looks like Shakespeare because it ***is*** Shakespeare

The Wall Street Journal is not Shakespeare (no offense)

1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	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
3 gram	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

Can you guess the training set author of the LM that generated these random 3-gram sentences?

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 gram Brazil on market conditions

This shall forbid it should be branded, if renown made it empty.

“You are uniformly charming!” cried he, with a smile of associating and now and then I bowed and they perceived a chaise and four to wish for.

The perils of overfitting

N-grams only work well for word prediction if the test corpus looks like the training corpus

- In real life, it often doesn't
- We need to train robust models that generalize!
- One kind of generalization: Zeros!
  - Things that don't ever occur in the training set
    - But occur in the test set

Zeros

Training set:

… denied the allegations
… denied the reports
… denied the claims
… denied the request

P(“offer” | denied the) = 0

- Test set
  … denied the offer
  … denied the loan

Zero probability bigrams

Bigrams with zero probability

- mean that we will assign 0 probability to the test set!

And hence we cannot compute perplexity (can't divide by 0)!

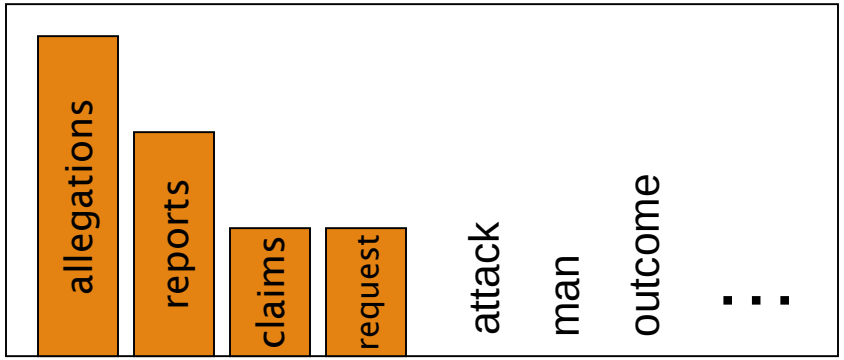
Language Modeling

Smoothing: Add-one (Laplace) smoothing

The intuition of smoothing (from Dan Klein)

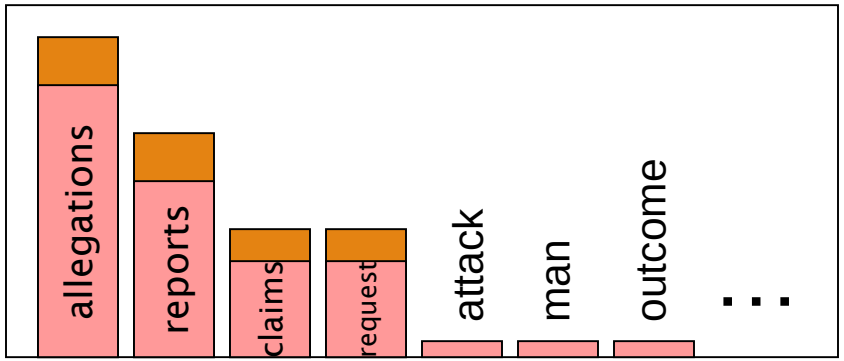
When we have sparse statistics:

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:

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

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

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

	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

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 isn't 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

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:

- use trigram if you have good evidence,
- otherwise bigram, otherwise unigram

Interpolation:

- mix unigram, bigram, trigram

Interpolation works better

Linear Interpolation

Simple interpolation

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

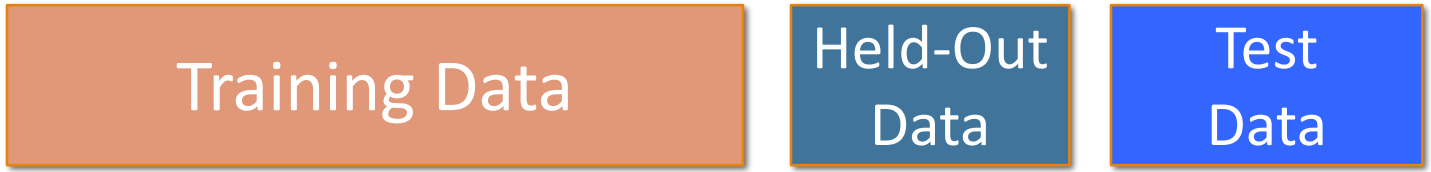
$$\sum_i \lambda_i = 1$$

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:

$$\log P(w_1 \ldots w_n \mid M(l_1 \ldots l_k)) = \sum_i \log P_{M(l_1 \ldots l_k)}(w_i \mid w_{i-1})$$

Unknown words: Open versus closed vocabulary tasks

If we know all the words in advanced

- Vocabulary V is fixed
- Closed vocabulary task

Often we don't know this

- **Out Of Vocabulary** = OOV words
- Open vocabulary task

Instead: create an unknown word token <UNK>

- Training of <UNK> probabilities
  - Create a fixed lexicon L of size V
  - At text normalization phase, any training word not in L changed to <UNK>
  - Now we train its probabilities like a normal word
- At decoding time
  - If text input: Use UNK probabilities for any word not in training

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 like tries
- Bloom filters: approximate language models
- 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

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

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

Advanced Language Modeling

Discriminative models:

- choose n-gram weights to improve a task, not to fit the training set

Parsing-based models

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 turned out to perform very poorly for speech recognition (why?)

Language Modeling

Advanced:

Kneser-Ney Smoothing

Absolute discounting: just subtract a little from each count

Suppose we wanted to subtract a little from a count of 4 to save probability mass for the zeros

How much to subtract ?

Church and Gale (1991)'s clever idea

Divide up 22 million words of AP Newswire

- Training and held-out set
- for each bigram in the training set
- see the actual count in the held-out set!

It sure looks like c* = (c - .75)

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

Absolute Discounting Interpolation

Save ourselves some time and just subtract 0.75 (or some d)!

discounted bigram

Interpolation weight

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1})P(w)$$

unigram

- (Maybe keeping a couple extra values of d for counts 1 and 2)

But should we really just use the regular unigram P(w)?

Kneser-Ney Smoothing I

Better estimate for probabilities of lower-order unigrams!

- Shannon game: *I can't see without my reading*_____? *Kong / glasses*
- "Kong" turns out to be more common than "glasses"
- … but "Kong" always follows "Hong"

The unigram is useful exactly when we haven't seen this bigram!

Instead of P(w): "How likely is w"

$P_{continuation}$(w): "How likely is w to appear as a novel continuation?

- For each word, count the number of bigram types it completes
- Every bigram type was a novel continuation the first time it was seen

$$P_{CONTINUATION}(w) \mu \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

Kneser-Ney Smoothing II

How many times does w appear as a novel continuation:

$$P_{CONTINUATION}(w) \mu \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

Normalized by the total number of word bigram types

$$\left|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}\right|$$

$$P_{CONTINUATION}(w) = \frac{\left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|}{\left|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}\right|}$$

Kneser-Ney Smoothing III

Alternative metaphor: The number of # of word types seen to precede w

$$|\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

normalized by the # of words preceding all words:

$$P_{CONTINUATION}(w) = \frac{\left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|}{\sum_{w'} \left|\{w'_{i-1} : c(w'_{i-1}, w') > 0\}\right|}$$

A frequent word (Kong) occurring in only one context (Hong) will have a low continuation probability

Kneser-Ney Smoothing IV

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{CONTINUATION}(w_i)$$

λ is a normalizing constant; the probability mass we've discounted

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \left| \{w : c(w_{i-1}, w) > 0\} \right|$$

the normalized discount

The number of word types that can follow w_{i-1}
= # of word types we discounted
= # of times we applied normalized discount

Kneser-Ney Smoothing: Recursive formulation

$$P_{KN}(w_i \mid w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^{i}) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1}) P_{KN}(w_i \mid w_{i-n+2}^{i-1})$$

$$c_{KN}(\cdot) = \begin{cases} count(\cdot) & \text{for the highest order} \\ continuationcount(\cdot) & \text{for lower order} \end{cases}$$

Continuation count = Number of unique single word contexts for •

Language Modeling

Advanced:

Kneser-Ney Smoothing