

EECS 595

Natural Language Processing

Lecture 2: NGram

Instructor: Joyce Chai

Misc.

- Homework 1 will be released later today
- Due date: September 17

Minimum Edit Distance

- Much of NLP concern with how similar two strings are
 - Spell checking and correction
 - Word Error Rate for speech recognition
 - machine translation, etc.
- MED is the minimum number of editing operations needed to transform one into the other
 - Insertion
 - Deletion
 - Substitution

Minimum Edit Distance

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N
d	s	s		i	s				

- If each operation has cost of 1
 - Distance between these is 5
- If substitutions cost 2 (Levenshtein)
 - Distance between them is 8

Minimum Edit Distance

One possible path

i n t e n t i o n	← <i>delete i</i>
n t e n t i o n	← <i>substitute n by e</i>
e t e n t i o n	← <i>substitute t by x</i>
e x e n t i o n	← <i>insert u</i>
e x e n u t i o n	← <i>substitute n by c</i>
e x e c u t i o n	

There can be many different paths

The problem becomes the search problem to find the path with minimum cost

Defining Min Edit Distance

- For two strings S_1 of len n , S_2 of len m
 - $\text{distance}(i,j)$ or $D(i,j)$
 - means the edit distance of $S_1[1..i]$ and $S_2[1..j]$
 - i.e., the minimum number of edit operations need to transform the first i characters of S_1 into the first j characters of S_2
 - The edit distance of S_1, S_2 is $D(n,m)$
- We compute $D(n,m)$ by computing $D(i,j)$ for all i ($0 \leq i \leq n$) and j ($0 \leq j \leq m$)
- Note the index associated with the source/target string: first is source and second is the target

Defining Min Edit Distance

- Base conditions:
 - $D(i,0) = i \quad /* deletion cost */$
 - $D(0,j) = j \quad /* insertion cost */$
- Recurrence Relation:
$$- D(i,j) = \min \begin{cases} D(i-1,j) + 1 & /* cost for deletion */ \\ D(i,j-1) + 1 & /* cost for insertion */ \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} & /* cost for substitution */ \end{cases}$$

Dynamic Programming

- A tabular computation of $D(n,m)$
- Bottom-up
 - Compute $D(i,j)$ for smaller i,j
 - Increase i, j to computer $D(i,j)$ using previously computed values based on smaller indexes.

The Edit Distance Table

source

N	9										
O	8										
I	7										
T	6	$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$									
N	5										
E	4										
T	3										
N	2										
I	1										
#	0	1	2	3	4	5	6	7	8	9	
	#	E	X	E	C	U	T	I	O	N	

target

N	9										
O	8										
I	7										
T	6										
N	5										
E	4										
T	3										
N	2										
I	1										
#	0	1	2	3	4	5	6	7	8	9	
	#	E	X	E	C	U	T	I	O	N	

N	9										
O	8										
I	7										
T	6										
N	5										
E	4										
T	3										
insertion											
I	1	← 2									
#	0	↓ 1	deletion		4	5	6	7	8	9	
substitution											
	E	X	E	C	U	T	I	O	N		

N	9											
O	8											
I	7											
T	6											
N	5											
E	4											
T	3											
N	2											
I	1	2	3									
#	0		1	2	3	4	5	6	7	8	9	
	#	E	X	E	C	U	T	I	O	N		

$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$

N	9										
O	8										
I	7										
T	6										
N	5										
E	4										
T	3										
N	2										
I	1 ← 2 ← 3 ↓										
#	0 → 1 ↓ 2 ↓	3	4	5	6	7	8	9			
	#	E	X	E	C	U	T	I	O	N	

Acquiring the alignment

- We can keep track of a “backtrace”
- Every time we enter a cell, remember where we come from
- Then when we reach the end, we can trace back from the upper right corner to get an alignment

Adding Backtrace to MinEdit

- Base conditions:
 - $D(i,0) = i$
 - $D(0,j) = j$
- Recurrence Relation:

$$- D(i,j) = \min \begin{cases} D(i-1,j) + 1 & \text{DOWN} \\ D(i,j-1) + 1 & \text{LEFT} \\ D(i-1,j-1) + \begin{cases} 1; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} & \text{DIAGONAL} \end{cases}$$

N	9	8	9	10	11	12	11	10	9	8
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

MinEdit with Backtrace

n	9	$\downarrow 8$	$\swarrow \leftarrow \downarrow 9$	$\swarrow \leftarrow \downarrow 10$	$\swarrow \leftarrow \downarrow 11$	$\swarrow \leftarrow \downarrow 12$	$\downarrow 11$	$\downarrow 10$	$\downarrow 9$	$\swarrow \mathbf{8}$	
o	8	$\downarrow 7$	$\swarrow \leftarrow \downarrow 8$	$\swarrow \leftarrow \downarrow 9$	$\swarrow \leftarrow \downarrow 10$	$\swarrow \leftarrow \downarrow 11$	$\downarrow 10$	$\downarrow 9$	$\swarrow \mathbf{8}$	$\leftarrow 9$	
i	7	$\downarrow 6$	$\swarrow \leftarrow \downarrow 7$	$\swarrow \leftarrow \downarrow 8$	$\swarrow \leftarrow \downarrow 9$	$\swarrow \leftarrow \downarrow 10$	$\downarrow 9$	$\swarrow \mathbf{8}$	$\leftarrow 9$	$\leftarrow 10$	
t	6	$\downarrow 5$	$\swarrow \leftarrow \downarrow 6$	$\swarrow \leftarrow \downarrow 7$	$\swarrow \leftarrow \downarrow 8$	$\swarrow \leftarrow \downarrow 9$	$\swarrow \mathbf{8}$	$\leftarrow 9$	$\leftarrow 10$	$\leftarrow 11$	
n	5	$\downarrow 4$	$\swarrow \leftarrow \downarrow 5$	$\swarrow \leftarrow \downarrow 6$	$\swarrow \leftarrow \downarrow 7$	$\swarrow \leftarrow \downarrow \mathbf{8}$	$\swarrow \leftarrow \downarrow 9$	$\swarrow \leftarrow \downarrow 10$	$\swarrow \leftarrow \downarrow 11$	$\swarrow \downarrow 10$	
e	4	$\swarrow 3$	$\leftarrow 4$	$\swarrow \leftarrow \mathbf{5}$	$\leftarrow \mathbf{6}$	$\leftarrow 7$	$\leftarrow \downarrow 8$	$\swarrow \leftarrow \downarrow 9$	$\swarrow \leftarrow \downarrow 10$	$\downarrow 9$	
t	3	$\swarrow \leftarrow \downarrow 4$	$\swarrow \leftarrow \downarrow \mathbf{5}$	$\swarrow \leftarrow \downarrow 6$	$\swarrow \leftarrow \downarrow 7$	$\swarrow \leftarrow \downarrow 8$	$\swarrow 7$	$\leftarrow \downarrow 8$	$\swarrow \leftarrow \downarrow 9$	$\downarrow 8$	
n	2	$\swarrow \leftarrow \downarrow \mathbf{3}$	$\swarrow \leftarrow \downarrow 4$	$\swarrow \leftarrow \downarrow 5$	$\swarrow \leftarrow \downarrow 6$	$\swarrow \leftarrow \downarrow 7$	$\swarrow \leftarrow \downarrow 8$	$\downarrow 7$	$\swarrow \leftarrow \downarrow 8$	$\swarrow \mathbf{7}$	
i	1	$\swarrow \leftarrow \downarrow 2$	$\swarrow \leftarrow \downarrow 3$	$\swarrow \leftarrow \downarrow 4$	$\swarrow \leftarrow \downarrow 5$	$\swarrow \leftarrow \downarrow 6$	$\swarrow \leftarrow \downarrow 7$	$\swarrow 6$	$\leftarrow 7$	$\leftarrow 8$	
#	0	1	2	3	4	5	6	7	8	9	
	#	e	x	e	c	u	t	i	o	n	

MinEdit with Backtrace

n	9	$\downarrow 8$	$\swarrow \leftarrow 9$	$\swarrow \leftarrow 10$	$\swarrow \leftarrow 11$	$\swarrow \leftarrow 12$	$\downarrow 11$	$\downarrow 10$	$\downarrow 9$	$\swarrow 8$	
o	8	$\downarrow 7$	$\swarrow \leftarrow 8$	$\swarrow \leftarrow 9$	$\swarrow \leftarrow 10$	$\swarrow \leftarrow 11$	$\downarrow 10$	$\downarrow 9$	$\swarrow 8$	$\leftarrow 9$	
i	7	$\downarrow 6$	$\swarrow \leftarrow 7$	$\swarrow \leftarrow 8$	$\swarrow \leftarrow 9$	$\swarrow \leftarrow 10$	$\downarrow 9$	$\swarrow 8$	$\leftarrow 9$	$\leftarrow 10$	
t	6	$\downarrow 5$	$\swarrow \leftarrow 6$	$\swarrow \leftarrow 7$	$\swarrow \leftarrow 8$	$\swarrow \leftarrow 9$	$\swarrow 8$	$\leftarrow 9$	$\leftarrow 10$	$\leftarrow 11$	
n	5	$\downarrow 4$	$\swarrow \leftarrow 5$	$\swarrow \leftarrow 6$	$\swarrow \leftarrow 7$	$\swarrow \leftarrow 8$	$\swarrow \leftarrow 9$	$\swarrow \leftarrow 10$	$\swarrow \leftarrow 11$	$\swarrow 10$	
e	4	$\swarrow 3$	$\leftarrow 4$	$\swarrow \leftarrow 5$	$\leftarrow 6$	$\leftarrow 7$	$\leftarrow 8$	$\swarrow \leftarrow 9$	$\swarrow \leftarrow 10$	$\downarrow 9$	
t	3	$\swarrow \leftarrow 4$	$\swarrow \leftarrow 5$	$\swarrow \leftarrow 6$	ins	$\swarrow \leftarrow 8$	$\swarrow 7$	$\leftarrow 8$	$\swarrow \leftarrow 9$	$\downarrow 8$	
n	2	$\swarrow \leftarrow 3$	sub	$\swarrow \leftarrow 5$	$\swarrow \leftarrow 6$	$\swarrow \leftarrow 7$	$\swarrow \leftarrow 8$	$\downarrow 7$	$\swarrow \leftarrow 8$	$\swarrow 7$	
i	1	sub	$\swarrow \leftarrow 3$	$\swarrow \leftarrow 4$	$\swarrow \leftarrow 5$	$\swarrow \leftarrow 6$	$\swarrow \leftarrow 7$	$\swarrow 6$	$\leftarrow 7$	$\leftarrow 8$	
#	del	0	1	2	3	4	5	6	7	8	9
#		e	x	e	c	u	t	i	o	n	

Performance

- Time: $O(nm)$
- Space: $O(nm)$
- Backtrace: $O(n+m)$

Language Modeling - NGram

Next Word Prediction

From a NY Times story...

- Stocks plunged this

Next Word Prediction

From a NY Times story...

- Stocks plunged this
- Stocks plunged this morning, despite a cut in interest rates

Next Word Prediction

From a NY Times story...

- Stocks plunged this
- Stocks plunged this morning, despite a cut in interest rates
- Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall ...

Next Word Prediction

From a NY Times story...

- Stocks plunged this
- Stocks plunged this morning, despite a cut in interest rates
- Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall ...
- Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall Street began

- Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall Street began trading for the first time since last

- Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall Street began trading for the first time since last *Tuesday's terrorist attacks.*
- …
- Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall Street began trading for the first time since last *Tuesday's terrorist attacks.*

Human Word Prediction

- Clearly, at least some of us have the ability to predict future words in an utterance.
- How?
 - Domain knowledge
 - Syntactic knowledge
 - Lexical knowledge

Claim

- A useful part of the knowledge needed to allow Word Prediction (guessing the next word) can be captured using simple statistical techniques.
- In particular, we'll rely on the notion of the probability of a sequence (e.g., sentence) and the likelihood of words co-occurring

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.

Why is this useful?

Example applications that employ language models:

- Speech recognition
- Handwriting recognition
- Spelling correction
- Machine translation systems
- Optical character recognizers

Terminology

- Sentence: unit of written language
- Utterance: unit of spoken language
- Word Form: the inflected form that appears in the corpus
- Lemma: lexical forms having the same stem, part of speech, and word sense
- Types: number of distinct words in a corpus (vocabulary size)
- Tokens: total number of running words

Words and Corpora

How to count words?

They picnicked by the pool, then lay back on the grass and looked at the stars

- **Type:** an element of the vocabulary.
- **Token:** an instance of that type in running text.
- How many? (ignore punctuation)
 - 16 tokens
 - 14 types

Words and Corpora

- I do uh main- mainly business data processing
 - Fragments, filled pauses
- Seuss's **cat** in the hat is different from other **cats!**
 - **Lemma:** same stem, part of speech, rough word sense
 - **cat** and **cats** = same lemma
 - **Wordform:** the full inflected surface form
 - **cat** and **cats** = different wordforms

Earlier Corpora

- Corpora are (generally online) collections of text and speech
 - Linguistic Data Consortium (LDC)
 - Brown Corpus
 - Wall Street Journal and AP News corpora
 - ATIS, Broadcast News (speech)
 - TDT (text and speech)
 - Switchboard, Call Home (speech)
 - AQUAINT
- 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 (only include those appearing 40 times or more)
 - 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

- 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
 - $P(w_n | w_1, w_2 \dots w_{n-1})$
- We'll call a statistical model that can assess this a *Language Model*

Language Modeling

- How might we go about calculating such a conditional probability?

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

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.

Chain Rule

conditional probability:

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

So:

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

“the dog”:

$$P(\text{The} \wedge \text{dog}) = P(\text{dog} | \text{the})P(\text{the})$$

“the dog bites”:

$$P(\text{The} \wedge \text{dog} \wedge \text{bites}) = P(\text{The})P(\text{dog} | \text{The})P(\text{bites} | \text{The} \wedge \text{dog})$$

Chain Rule

The probability of a word sequence is the probability of a conjunctive event.

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

Unfortunately, that's really not helpful in general.
Why?

Markov Assumption

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

- $P(w_n)$ can be approximated using only $N-1$ previous words of context
- This lets us collect statistics in practice
- Markov models are the class of probabilistic models that assume that we can predict the probability of some future unit without looking too far into the past
- Order of a Markov model: length of prior context

Simple N-Grams

- An **N-gram model** uses the previous N-1 words to predict the next one:
 - $P(w_n | w_{n-1})$
 - We'll pretty much always be dealing with $P(<\text{word}> | <\text{some prefix}>)$
- unigrams: $P(\text{dog})$
- bigrams: $P(\text{dog} | \text{big})$
- trigrams: $P(\text{dog} | \text{the big})$
- quadrigrams: $P(\text{dog} | \text{the big dopey})$

How do we get the N-gram probabilities? Use MLE

N-gram models can be trained by **counting** and **normalization**

Bigram:
$$P(w_n | w_{n-1}) = \frac{C(w_{n-1} w_n)}{C(w_{n-1})}$$

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

Using N-Grams

- Recall that
 - $P(w_n | w_{1..n-1}) \approx P(w_n | w_{n-N+1..n-1})$
- For a bigram grammar
 - $P(\text{sentence})$ can be approximated by multiplying all the bigram probabilities in the sequence
 - $P(\text{I want to eat Chinese food}) = P(\text{I} | \langle\text{start}\rangle) P(\text{want} | \text{I}) P(\text{to} | \text{want}) P(\text{eat} | \text{to}) P(\text{Chinese} | \text{eat}) P(\text{food} | \text{Chinese})$

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*

A Bigram Grammar Fragment from BERP

Eat on	.16	Eat Thai	.03
Eat some	.06	Eat breakfast	.03
Eat lunch	.06	Eat in	.02
Eat dinner	.05	Eat Chinese	.02
Eat at	.04	Eat Mexican	.02
Eat a	.04	Eat tomorrow	.01
Eat Indian	.04	Eat dessert	.007
Eat today	.03	Eat British	.001

<start> I	.25	Want some	.04
<start> I'd	.06	Want Thai	.01
<start> Tell	.04	To eat	.26
<start> I'm	.02	To have	.14
I want	.32	To spend	.09
I would	.29	To be	.02
I don't	.08	British food	.60
I have	.04	British restaurant	.15
Want to	.65	British cuisine	.01
Want a	.05	British lunch	.01

- $P(\text{I want to eat British food}) = P(\text{I}|\text{<start>})$
 $P(\text{want}|\text{I}) P(\text{to}|\text{want}) P(\text{eat}|\text{to}) P(\text{British}|\text{eat})$
 $P(\text{food}|\text{British}) = .25 * .32 * .65 * .26 * .001 * .60$
 $= .000080$
- vs. $I \text{ want to eat Chinese food} = .00015$
- Probabilities seem to capture ``syntactic'' facts, ``world knowledge''
 - **eat** is often followed by a NP
 - British food is not too popular

Log Probability

- Individual probabilities can be small, and the product of many probabilities can be VERY small.
- It is usually best to compute N-gram probabilities in log space, e.g.,

$$\begin{aligned}\log_2 P(w_1 \dots w_n) &= \log_2 \prod P(w_i | w_{i-1}) \\ &= \sum \log_2 P(w_i | w_{i-1})\end{aligned}$$

- In the end, the actual probability can be restored by taking the anti-log: $2^{\log(X)} = X$

Some Useful Empirical Observations

- A small number of events occur with high frequency
- A large number of events occur with low frequency
- You can quickly collect statistics on the high frequency events
- You might have to wait an arbitrarily long time to get valid statistics on low frequency events
- Some of the zeroes in the table are really zeroes. But others are simply low frequency events you haven't seen yet.

Problem with MLE estimate - Example

Suppose in a corpus, 10 instances with “come across ..”

Come across as: 8

Come across more: 1

Come across a: 1

Using MLE estimation:

$$P(\text{as}|\text{come across}) = 0.8$$

$$P(\text{more}|\text{come across}) = 0.1$$

$$P(\text{a} \mid \text{come across}) = 0.1$$

Any problem?

Problem with MLE estimate - Example

Suppose in a corpus, 10 instances with “come across ..”

Come across as: 8

Come across more: 1

Come across a: 1

Using MLE estimation:

$P(\text{as}|\text{come across}) = 0.8$

$P(\text{more}|\text{come across}) = 0.1$

$P(\text{a} \mid \text{come across}) = 0.1$

$P(x \mid \text{come across}) = 0$ for x not among the above 3 words

The MLE does not capture the fact that there are other words which can follow “come across”, e.g., “the”, “some”, etc, but just do not appear in the training set

Problem with MLE estimate

While there are a limited number of frequent events in language, there is a seemingly never ending tail to the probability distribution of rarer events, and we can never collect enough data to get to the end of the tail.

Devise better estimators that allow for the possibility that we will see events that we didn't see in the training text

How to address this problem? Smoothing!

The intuition of smoothing (from Dan Klein)

- When we have sparse statistics:

$P(w | \text{denied the})$

3 allegations

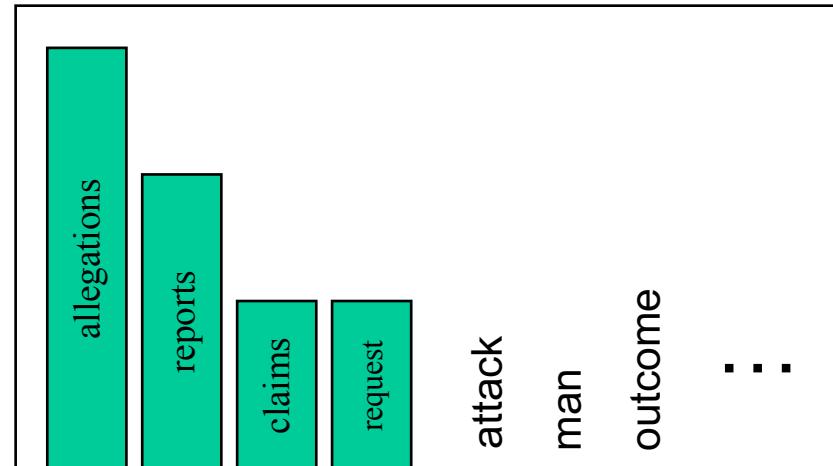
2 reports

1 claims

1 request

7 total

- Steal probability mass to generalize better



$P(w | \text{denied the})$

2.5 allegations

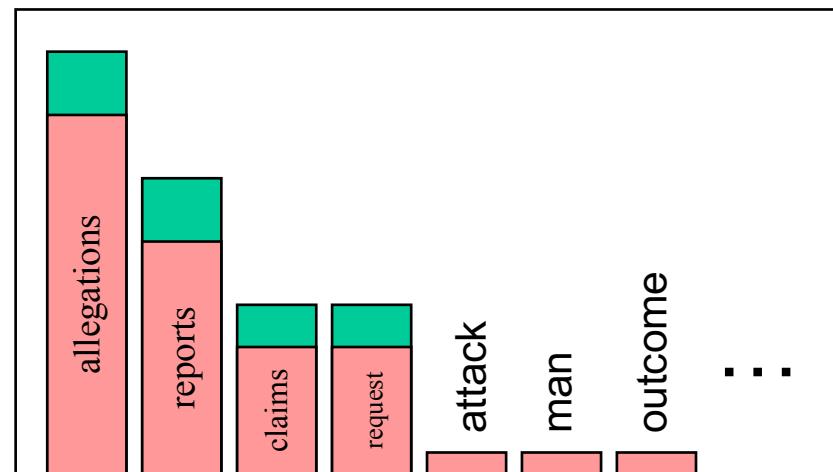
1.5 reports

0.5 claims

0.5 request

2 other

7 total



Discounting Method

Suppose we've seeing the following counts

X	$count(x)$	$P_{ML}(w_i w_{i-1}) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})}$
the	50	
the flower	16	16/50
the dog	14	14/50
the park	8	8/50
the tree	6	6/50
the woman	3	3/50
The path	1	1/50
The pond	1	1/50
The afternoon	1	1/50

Discounting Method

Now define “discounted” counts, $\text{Count}^*(x) = \text{Count}(x) - 0.5$

New estimates:

X	$\text{count}(x)$	$\text{count}^*(x)$	$P_{ML}(w_i w_{i-1}) = \frac{C^*(w_{i-1}w_i)}{C(w_{i-1})}$
the	50		
the flower	16	15.5	15.5/50
the dog	14	13.5	13.5/50
the park	8	7.5	7.5/50
the tree	6	5.5	5.5/50
the woman	3	2.5	2.5/50
The path	1	0.5	0.5/50
The pond	1	0.5	0.5/50
The afternoon	1	0.5	0.5/50

Discounting Method

- We now have some “miss probability mass”

$$\alpha(w_{i-1}) = 1 - \sum_w \frac{\text{Count}^*(w_{i-1}, w)}{\text{Count}(w_{i-1})}$$

E.g., in our example $\alpha(w_{i-1}) = 4/50$

- Divide the remaining probability mass between word w for which $\text{Count}(w_{i-1}, w) = 0$

Smoothing Techniques

Smoothing: re-evaluate some zero-probability and low-probability N-grams, and assign them non-zero values.

- Laplace smoothing
- Backoff
- Interpolation
- Kneser-Ney smoothing

Laplace Smoothing

- Also add-one smoothing:
 - Add 1 to the count
 - Original Prob: $P_i = \frac{c_i}{N}$
 - Smoothed Prob: $P_i^* = \frac{c_i + 1}{N + V}$
 - Discount: Lowering some non-zero counts

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

Laplace Smoothing

Given a corpus, suppose vocabulary size $V = 15,000$

“....was...”: 3000 times; “...was not....”: 608 times

Original: $P(\text{not} \mid \text{was}) = \frac{C(\text{"was not"})}{C(\text{"was"})} = \frac{608}{3000} = 0.203$

After add one smoothing:

$$P(\text{not} \mid \text{was}) = \frac{C(\text{"was not"}) + 1}{C(\text{"was"}) + V} = \frac{608 + 1}{3000 + 15000} = 0.034$$

Any problem?

Laplace Smoothing

Original 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

Discounted counts

	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

Laplace Smoothing

- In general, add-one smoothing is a poor method of smoothing
 - Add k (a fraction) also works poorly
- We'd like to find methods that don't change the original counts/probabilities so drastically

Backoff and Interpolation

- If we are estimating:
 - trigram $p(z|x,y)$
 - but $\text{count}(xyz)$ is zero
- Use info from:
 - Bigram $p(z|y)$
- Or even:
 - Unigram $p(z)$
- How to combine this trigram, bigram, unigram info in a valid fashion?

Backoff Vs. Interpolation

- **Backoff:** use trigram if you have it,
otherwise bigram, otherwise unigram
- **Interpolation:** mix all three

Backoff

- Use N-gram “hierarchy” to “back off” to a lowered order N-gram if there is zero evidence for a higher-order N-gram
- Backoff methods (e.g. Katz)
 - Where trigram unavailable **back off** to bigram if available, otherwise, unigram probability

Backoff

Is the following right?

$$P_{backoff}(w_i \mid w_{i-2}w_{i-1}) = \begin{cases} P_{ML}(w_i \mid w_{i-2}w_{i-1}), & \text{if } C(w_{i-2}w_{i-1}w_i) > 0 \\ P_{ML}(w_i \mid w_{i-1}), & \text{if } C(w_{i-2}w_{i-1}w_i) = 0 \\ & \quad \text{and } C(w_{i-1}w_i) > 0 \\ P_{ML}(w_i), & \text{otherwise} \end{cases}$$

Backoff

Is the following right?

$$P_{backoff}(w_i \mid w_{i-2}w_{i-1}) = \begin{cases} P_{ML}(w_i \mid w_{i-2}w_{i-1}), & \text{if } C(w_{i-2}w_{i-1}w_i) > 0 \\ P_{ML}(w_i \mid w_{i-1}), & \text{if } C(w_{i-2}w_{i-1}w_i) = 0 \\ & \quad \text{and } C(w_{i-1}w_i) > 0 \\ P_{ML}(w_i), & \text{otherwise} \end{cases}$$

(Brants et al. 2007) – stupid backoff, no discounting

To make a true probability distribution, any backoff language model must also be discounted

Katz Backoff for Trigram

$$P_{Katz}(w_i \mid w_{i-2}w_{i-1}) = \begin{cases} P^*(w_i \mid w_{i-2}w_{i-1}), & \text{if } C(w_{i-2}w_{i-1}w_i) > 0 \\ \alpha(w_{i-2}w_{i-1})P_{Katz}(w_i \mid w_{i-1}), & \text{otherwise} \end{cases}$$

$$P_{Katz}(w_i \mid w_{i-1}) = \begin{cases} P^*(w_i \mid w_{i-1}), & \text{if } C(w_{i-1}w_i) > 0 \\ \alpha(w_{i-1})P(w_i), & \text{otherwise} \end{cases}$$

Katz Backoff for Trigram

With the following:

$$\alpha(w_{i-2}w_{i-1}) = \frac{1 - \sum_{\substack{w_i: C(w_{i-2}, w_{i-1}, w_i) > 0}} P^*(w_i | w_{i-2}, w_{i-1})}{\sum_{\substack{w_i: C(w_{i-2}, w_{i-1}, w_i) = 0}} P_{katz}(w_i | w_{i-1})}$$

$$\alpha(w_{i-1}) = \frac{1 - \sum_{\substack{w_i: C(w_{i-1}, w_i) > 0}} P^*(w_i | w_{i-1})}{\sum_{\substack{w_i: C(w_{i-1}, w_i) = 0}} P(w_i)}$$

Linear Interpolation

$$\begin{aligned} P_{LI}(w_i \mid w_{i-2}, w_{i-1}) &= \lambda_1 \times P_{ML}(w_i \mid w_{i-2}, w_{i-1}) \\ &\quad + \lambda_2 \times P_{ML}(w_i \mid w_{i-1}) \\ &\quad + \lambda_3 \times P_{ML}(w_i) \end{aligned}$$

Where $\lambda_1 + \lambda_2 + \lambda_3 = 1$, and $\lambda_i \geq 0$ for all i.

Linear Interpolation

Why does this estimation correctly define a distribution?

$$\begin{aligned} & \sum_{w_i \in V} P_{LI}(w_i | w_{i-2}, w_{i-1}) \\ &= \sum_{w_i \in V} [\lambda_1 \times P_{ML}(w_i | w_{i-2}, w_{i-1}) + \lambda_2 \times P_{ML}(w_i | w_{i-1}) + \lambda_3 \times P_{ML}(w_i)] \\ &= \lambda_1 \sum_{w_i \in V} P_{ML}(w_i | w_{i-2}, w_{i-1}) + \lambda_2 \sum_{w_i \in V} P_{ML}(w_i | w_{i-1}) + \lambda_3 \sum_{w_i \in V} P_{ML}(w_i) \\ &= \lambda_1 + \lambda_2 + \lambda_3 \\ &= 1 \end{aligned}$$

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, search for lambda values that when plugged into previous equation, give largest probability for held-out set
 - use expectation-maximization to do this search (check it out if you are interested)

Kneser-Ney Smoothing

Bigram count in training set	Bigram count in heldout set
0	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

- Observations: except for 0 and 1, all other re-estimated counts is about 0.75 less than the MLE count.
- Absolute discounting

Knerson-Ney Smoothing

- Intuition: words that have appeared in more contexts are more likely to appear in some new context

$$P_{KN}(w_i \mid w_{i-1}) = \begin{cases} \frac{C(w_{i-1}w_i) - D}{C(w_{i-1})} & \text{if } C(w_{i-1}w_i) > 0 \\ \alpha(w_{i-1}) \frac{\mathbf{C}(\bullet w_i)}{\sum\limits_{w_i} \mathbf{C}(\bullet w_i)} & \text{otherwise} \end{cases}$$

where $\mathbf{C}(\bullet w_i)$ is the number of unique words preceding w_i

Class N-grams

- Define classes for words that exhibit similar semantic or grammatical behaviors. Another effective way to handle the data sparsity problem.
- Simple assumption: a word can be uniquely mapped to only one class

$$P(w_i | c_{i-n+1}, \dots, c_{i-1}) = P(w_i | c_i)P(c_i | c_{i-n+1}, \dots, c_{i-1})$$

Class Trigram

$$P(W) = \prod_i P(w_i | c_i)P(c_i | c_{i-2}, c_{i-1})$$

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.



Shannon's Method

- Sample a random bigram ($\langle s \rangle$, w) according to its probability
- Now sample a random bigram (w, x) according to its probability
 - Where the prefix w matches the suffix of the first.
- And so on until we randomly choose a (y, $\langle /s \rangle$)
- Then string the words together
- $\langle s \rangle$ I

I want

want to

to eat

eat Chinese

Chinese food

food $\langle /s \rangle$

Approximating Shakespeare

- As we increase the value of N, the accuracy of the n-gram model increases
- Generating sentences with random unigrams...
 - Every enter now severally so, let
 - Hill he late speaks; or! a more to leg less first you enter
- With bigrams...
 - 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.

- Trigrams
 - Sweet prince, Falstaff shall die.
 - This shall forbid it should be branded, if renown made it empty.
- Quadrigrams
 - What! I will go seek the traitor Gloucester.
 - Will you not tell me who I am?

- There are 884,647 tokens, with 29,066 word form types, in about a one million word Shakespeare corpus
- Shakespeare produced 300,000 bigram types out of 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.

N-Gram Training Sensitivity

- If we repeated the Shakespeare experiment but trained on a Wall Street Journal corpus, there would be little overlap in the output
- This has major implications for corpus selection or design

Unigram

Months the my and issue of year foreign new exchange's september were recession ex-change 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

Google N-Gram Release

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

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 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

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.
- More: <https://books.google.com/ngrams/info>
- <https://pypi.org/project/google-ngram-downloader/>

Unknown Words

- But once we start looking at test data, we'll run into words that we haven't seen before (pretty much regardless of how much training data you have).
- With an *Open Vocabulary* task
 - Create an unknown word token <UNK>
 - Training of <UNK> probabilities
 - Create a fixed lexicon L , of size V
 - From a dictionary or
 - A subset of terms from the training set
 - At text normalization phase, any training word not in L changed to <UNK>
 - Now we count that like a normal word
 - At test time
 - Use UNK counts for any word not in training

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? How do we quantify that?

Evaluation

- Standard method
 - Train parameters of our model on a **training set**.
 - Look at the models performance on some new 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.
- What could be an intuitive but expensive way?
 - Extrinsic evaluation: speech recognition
 - Expensive
- Intrinsic evaluation: Perplexity

Perplexity

- Perplexity is the probability of the test set (assigned by the language model), normalized by the number of words:

$$\begin{aligned} PP(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \end{aligned}$$

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

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

N -gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Take away

- N-Grams played a crucial role in NLP for a long time.
 - The idea and concepts (e.g., smoothing) etc. are still important for similar problems in other fields.
- In practice, N-Gram has been replaced by neural language models recently