EECS 595

Natural Language Processing

Lecture 1: Introduction

Instructor: Joyce Chai

What is NLP?

Dave Bowman: Open the pod bay doors, HAL.

What is NLP?

Dave Bowman: Open the pod bay doors, HAL.

HAL: I'm sorry Dave. I'm afraid I can't do that.

- Fundamental goal: *deep* understand of broad language
  - Not just string processing or keyword matching!
- Applications:

  information extraction, question answering, machine translation, conversational systems

What is NLP

- The study of human languages and how they can be represented computationally and analyzed and generated algorithmically
  - *The dog likes bacon.* --> like (dog, bacon)
  - like (dog, bacon) --> *The dog likes bacon*
- Studying NLP involves studying natural language, formal representations, and algorithms for their manipulation

NLP becomes increasingly important

- Play an important role in curbing information explosion on the internet
  - Information extraction
  - Text summarization
  - Social media analysis
- Used for building natural interfaces to databases, machine translations, etc.
  - Intelligent conversational agents

Why NLP is Difficult: Multidisciplinary

- **Linguistics**: how words, phrases, and sentences are formed.
- **Psycholinguistics**: how people understand and communicate using human language
- **Philosophy**: relates to the semantics of language; notation of meaning. NLP requires considerable knowledge about the world

Why NLP is Difficult: Multidisciplinary

- **Computer Science**: deals with model formation and implementation
- **Mathematics and Statistics**: deals with probabilities, statistical distribution and hypothesis testing of language phenomena
- **Artificial Intelligence**: relates to knowledge representation and reasoning

Language Ambiguities

I made her duck.

- How many different interpretations does the above sentence have?
- How can each ambiguous piece be resolved?

Language Ambiguities

- Lexical ambiguity: when a word has more than one part of speech

  Rice flies like sand.

- Structural ambiguity:

  John saw the boy with a telescope

  John saw the boy with a telescope

Basic levels of language processing

- Phonetics: how words are related to the sounds that realize them.
- Morphology: how words are constructed. *beauty, beautiful*
- Syntax: how words can be put together to form correct sentences, and the role of each plays in the sentence. *John likes Mary*

Basic levels of language processing

- Semantics: the meaning of words and sentences

  bass fishing, bass playing
- Discourse: how the meaning of words and sentences is affected by the surrounding text or utterances

  Mary bought a new computer yesterday. She likes it very much. (pronoun resolution)
- Pragmatics: how sentences are used in different situations (contexts)

  Mary grabbed her umbrella

  A) It is a cloudy day

  B) She was afraid of dogs

Representations and Algorithms for NLP

- Representations: formal models used to capture linguistic knowledge
- Algorithms manipulate representations to analyze or generate linguistic phenomena

Examples of NLP Applications

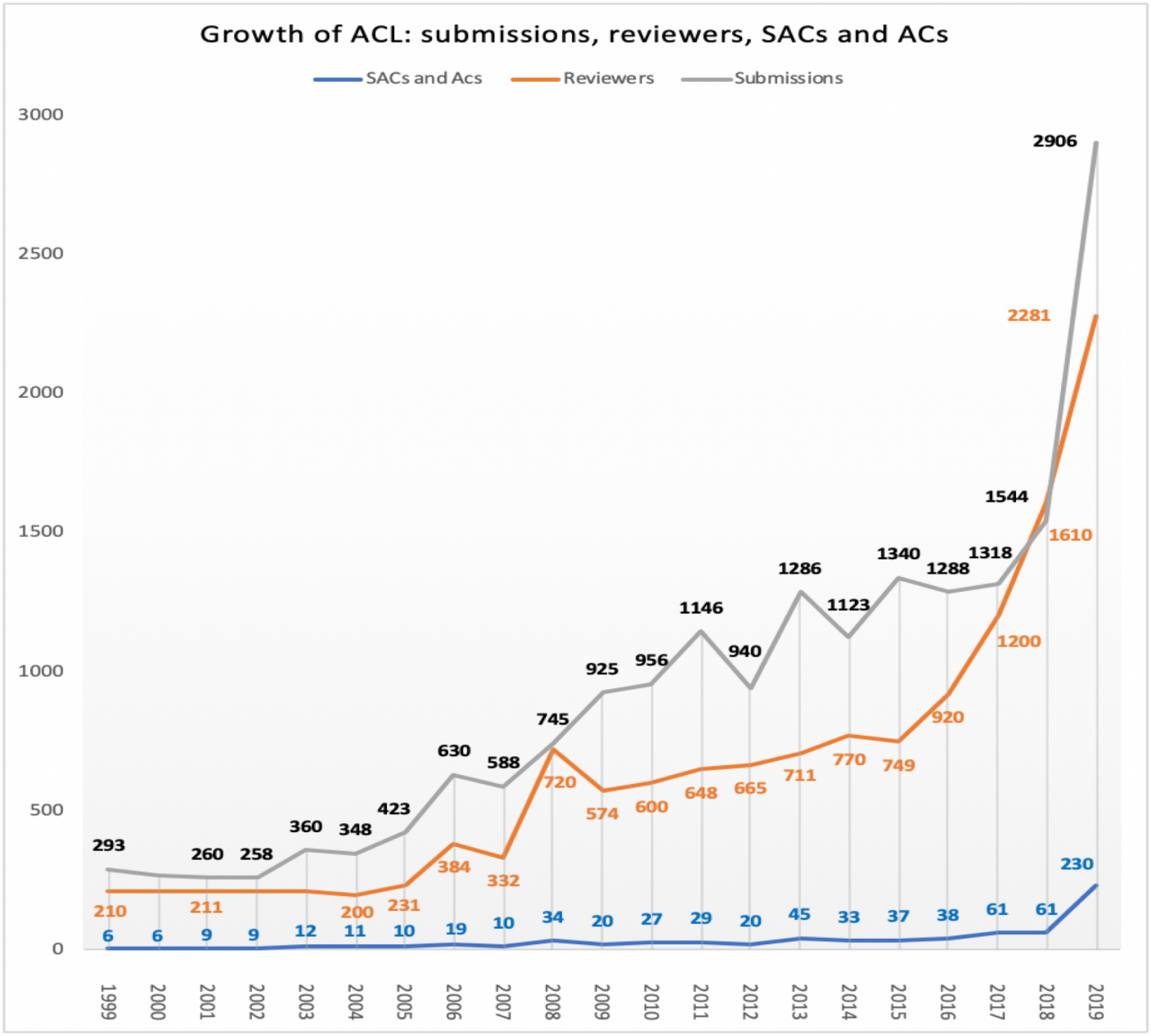
- dialogue systems
  - for automated service: communicator
  - for education and entertainment:
- speak to your appliances
- automated speech recognition and text-to-speech in vehicles.
- generate weather reports in two languages
- translate Web pages into different languages
  - http://translate.google.com/translate_t#
  - http://www.systransoft.com
- grade essays
- question answering
- Text analytics: mining textual documents (e.g., event tracking, political opinion tracking, social network analysis, etc.)

Some brief history

- Foundational insights (40's and 50's): automaton (Turing), probabilities, information theory (Shannon), formal languages (Backus and Naur), noisy channel and decoding (Shannon), first systems (Davis et al., Bell Labs)
- Two camps (57-70): symbolic and stochastic.
  Transformation grammar (Harris, Chomsky), artificial intelligence (Minsky, McCarthy, Shannon, Rochester), automated theorem proving and problem solving (Newell and Simon)
  Bayesian reasoning (Mosteller and Wallace)
  Corpus work (Kučera and Francis)

Some brief history

- Four paradigms (70-83): stochastic (IBM), logic-based (Colmerauer, Pereira and Warren, Kay, Bresnan), NLU (Winograd, Schank, Fillmore), discourse modelling (Grosz and Sidner)
- Empiricism and finite-state models redux (83-93): Kaplan and Kay (phonology and morphology), Church (syntax)
- Late years (94-99): strong integration of different techniques, different areas (including speech and IR)
- More recent (00-present): rise of machine learning



(from ACL2019 PC chair blog)

Topics covered

- Three major parts:
  - Linguistic, mathematical, and computational background
  - Levels of linguistic processing: morphology, syntax, semantics, and discourse
  - Applications: sentiment analysis, information extraction, question answering, machine translation, dialogue systems

Goals

- Three major goals:
  - Learn the basic principles and theoretical issues underlying natural language processing
  - Learn techniques and tools used to develop practical, robust systems
  - Gain insight into many open research problems in natural language

Logistics

- Instructor: Joyce Chai
- Office: Beyster 3632
- Office Hours: Tuesday: noon-3pm or by appointment, some exceptions at the beginning
  - September 10: noon-1pm
  - September 17: noon-1:15pm
- TA: Cristian-Paul Bara (cpbara@umich.edu), Office hours: Thursdays: 10:00-noon, Beyster 1637 Table 01.
- CANVAS
  - Syllabus, lecture notes, assignments
  - submissions

Textbook and Lecture Notes

- *Speech and Language Processing, an introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, third edition (draft)* by Daniel Jurafsky and James Martin, Prentice Hall.

- Many lecture notes will also come from this website.

Programming

- In each of the first five lectures (before the fall break), Paul will give a one-hour tutorial on Python & Pytorch
- Programming assignments are excepted in Python

Grading

- Four programming assignments: 50
  - Submission through CANVAS
- In-class exam: 20%
  - Open notes
- Final project (30%): A group project with up to 3 people.
  - The scope of the project should be proportional to the number of people in the project group.
  - You can propose a project related to NLP or the application of NLP techniques for a different subject.
  - Project proposal (5%)
  - Presentation (10%) – Format TBD
  - Final paper (15%)

Today

- Review some of the simple representations and ask ourselves how we might use them to do interesting and useful things
  - Regular Expression and its use in chatbots
  - Minimum editing distance

The Turing Test

- Alan Turing: the *Turing test* (for intelligence)
- Three participants: a computer and two humans (one is an interrogator)
- Interrogator's goal: to tell the machine and human apart
- Machine's goal: to fool the interrogator into believing that a person is responding
- Other human's goal: convince the interrogator that the other participant is the machine

Q: Please write me a sonnet on the topic of the Forth Bridge.
A: Count me out on this one. I never could write poetry.
Q: Add 34957 to 70764.
A: 105621 (after a pause)

The Turing Test

- Alan Turing: the *Turing test* (for intelligence)
- Three participants: a computer and two humans (one is an interrogator)
- Interrogator's goal: to tell the machine and human apart
- Machine's goal: to fool the interrogator into believing that a person is responding
- Other human's goal: convince the interrogator that the other participant is the machine

Eliza: http://www.manifestation.com/neurotoys/eliza.php3

Regular expressions

- A formula in a special language that is used for specifying simple classes of strings
  - A string is a sequence of symbols
  - For text-based search, a string is a sequence of alphanumeric characters (letters, numbers, spaces, tabs, and punctuation)
- Can be used to specify search strings and define a language in a formal way.

Basic Regular Expression Patterns

- Case sensitive
- Perl-based syntax (slightly different from other notations for regular expressions)
- Disjunctions **`[abc]`**
- Ranges **`[A-Z]`**
- Negations **`[^Ss]`**
- Optional characters **`?`**, **`+`**, and **`*`**
- Wild cards **`.`**
- Anchors **`^`** and **`$`**, also **`\b`** and **`\B`**
- Disjunction, grouping, and precedence **`|`**

Regular expressions

- How can we search for any of these?
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks

Regular Expressions: Disjunctions

- Letters inside square brackets []

Pattern	Matches
`[wW]oodchuck`	Woodchuck, woodchuck
`[1234567890]`	Any digit

- Ranges `[A-Z]`

Pattern	Matches	
`[A-Z]`	An upper case letter	`Drenched Blossoms`
`[a-z]`	A lower case letter	`my beans were impatient`
`[0-9]`	A single digit	`Chapter 1: Down the Rabbit Hole`

Regular Expressions: Negation in Disjunction

- Negations `[^Ss]`
  - Carat means negation only when first in []

Pattern	Matches	
`[^A-Z]`	Not an upper case letter	`Oyfn pripetchik`
`[^Ss]`	Neither 'S' nor 's'	`I have no exquisite reason"`
`[^e^]`	Neither e nor ^	`ears`
`a^b`	The pattern a carat b	`Look up a^b now`

Regular Expressions: More Disjunction

- Woodchucks is another name for groundhog!
- The pipe | for disjunction

Pattern	Matches
`groundhog|woodchuck`	
`yours|mine`	yours mine
`a|b|c`	= [abc]
`[gG]roundhog|[Ww]ood chuck`	

Regular Expressions: ? * + .

Kleene *, Kleene +

Pattern	Matches	
`colou?r`	Optional previous char	`color` `colour`
`oo*h!`	0 or more of previous char	`oh!` `ooh!` `oooh!` `ooooh!`
`o+h!`	1 or more of previous char	`oh!` `ooh!` `oooh!` `ooooh!`
`baa+`		`baa` `baaa` `baaaa` `baaaaa`
`beg.n`		`begin` `begun` `begun` `beg3n`

Regular Expressions: Anchors ^ $

- ^ beginning of the string
- $ end of the string

Pattern	Matches
^[A-Z]	Palo Alto
^[^A-Za-z]	1 "Hello"
\.$	The end.
.$	The end? The end!

RE	Expansion	Match	First Matches
\d	[0-9]	any digit	Party␣of␣5
\D	[^0-9]	any non-digit	Blue␣moon
\w	[a-zA-Z0-9_]	any alphanumeric/underscore	Daiyu
\W	[^\w]	a non-alphanumeric	!!!!
\s	[␣\r\t\n\f]	whitespace (space, tab)	
\S	[^\s]	Non-whitespace	in␣Concord

Figure 2.7 Aliases for common sets of characters.

RE	Match
*	zero or more occurrences of the previous char or expression
+	one or more occurrences of the previous char or expression
?	exactly zero or one occurrence of the previous char or expression
{n}	*n* occurrences of the previous char or expression
{n,m}	from *n* to *m* occurrences of the previous char or expression
{n,}	at least *n* occurrences of the previous char or expression
{,m}	up to *m* occurrences of the previous char or expression

Figure 2.8 Regular expression operators for counting.

Example

- Find me all instances of the word “the” in a text.

`/the/` Misses capitalized examples

`/[tT]he/` Incorrectly returns `other` or `theology`

`/[^a-zA-Z][tT]he[^a-zA-Z]`

/\b[tT]he\b/

Errors

- The process we just went through was based on fixing two kinds of errors
  - Matching strings that we should not have matched (there, then, other)
    - False positives (Type I)
  - Not matching things that we should have matched (The)
    - False negatives (Type II)

Errors cont.

- In NLP we are always dealing with these kinds of errors.
- Reducing the error rate for an application often involves two antagonistic efforts:
  - Increasing accuracy or precision (minimizing false positives)
  - Increasing coverage or recall (minimizing false negatives).

A more complex example

- Exercise: Write a regular expression that will match "any PC with more than 500MHz and 32 Gb of disk space for less than $1000":

```
/$[0-9]+/
/$[0-9]+\.[0-9][0-9]/
/\b$[0-9]+(\.[0-9][0-9])?\b/
/\b[0-9]+ *(MHz|[Mm]egahertz|Ghz|
    [Gg]igahertz)\b/
/\b[0-9]+ *(Mb|[Mm]egabytes?)\b/
/\b[0-9](\.[0-9]+) *(Gb|[Gg]igabytes?)\b/
```

Substitutions and memory

- Substitutions (Transduction)

```
s/colour/color/
```

- Memory (`\1`, `\2`, etc. refer back to matches)

35 boxes => <35> boxes:

```
s/([0-9]+)/<\1>/
```

s/regexp1/pattern/
s/I am feeling (.+)/You are feeling \1?/
s/I gave (.+) to (.+)/Why would you give \2 \1?/

Eliza [Weizenbaum, 1966]

User: Men are all alike

ELIZA: IN WHAT WAY

User: They're always bugging us about something or other

ELIZA: CAN YOU THINK OF A SPECIFIC EXAMPLE?

User: Well, my boyfriend made me come here

ELIZA: YOUR BOYFRIEND MADE YOU COME HERE

User: He says I'm depressed much of the time

ELIZA: I AM SORRY TO HEAR THAT YOU ARE DEPRESSED

Eliza-style regular expressions

Step 1: replace first person references with second person references
Step 2: use additional regular expressions to generate replies
Step 3: rank possible transformations

```
s/.* YOU ARE (depressed|sad) .*/I AM SORRY TO
  HEAR YOU ARE \1/
s/.* YOU ARE (depressed|sad) .*/WHY DO YOU
  THINK YOU ARE \1/
s/.* all .*/IN WHAT WAY/
s/.* always .*/CAN YOU THINK OF A SPECIFIC
  EXAMPLE/
```

Uses of Regular Expressions in NLP

- Regular expressions play a surprisingly large role
  - Sophisticated sequences of regular expressions are often the first model for any text processing text
- For many hard tasks, we use machine learning classifiers
  - But regular expressions are used as features in the classifiers
  - Can be very useful in capturing generalizations

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

How many words?

N = number of tokens

V = vocabulary = set of types

$|V|$ is the size of the vocabulary

	Tokens = N	Types = |V|
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
Google N-grams	1 trillion	13 million

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 s the minimum number of editing operations needed to transform one into the other
  - Insertion
  - Deletion
  - Substitution

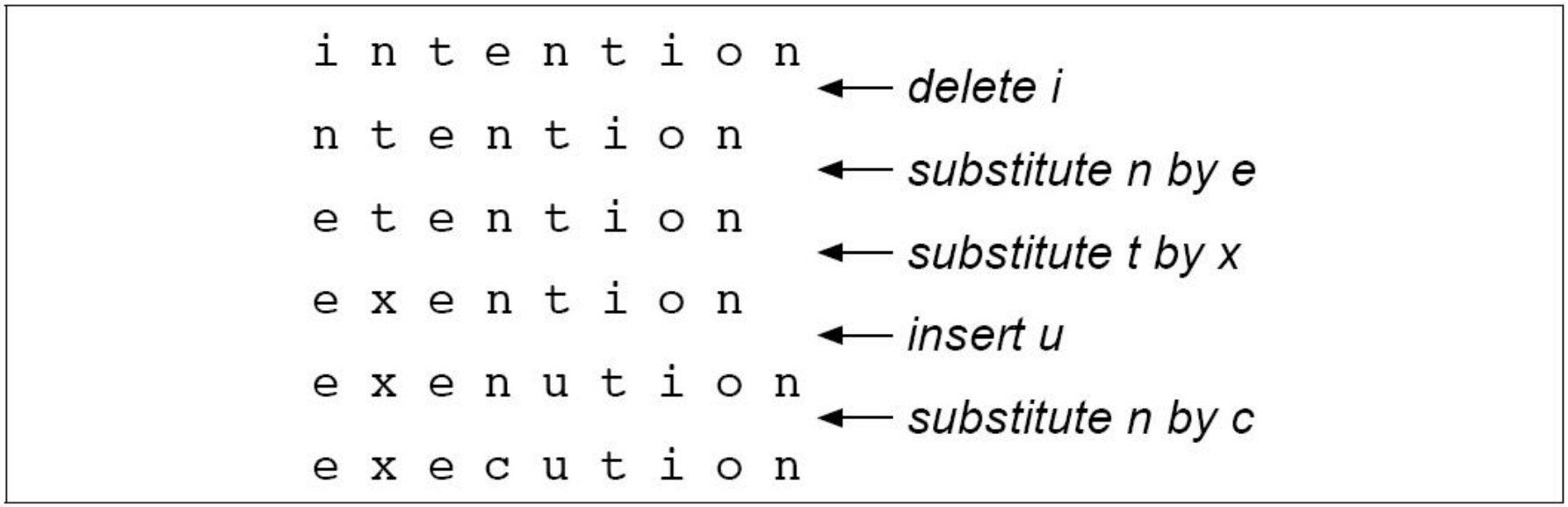
Minimum Edit Distance

```
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



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
  - 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* /* *deletion cost**/
  - D(0,*j*) = *j* /* *insertion cost**/

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

    /* cost for substitution */

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									
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

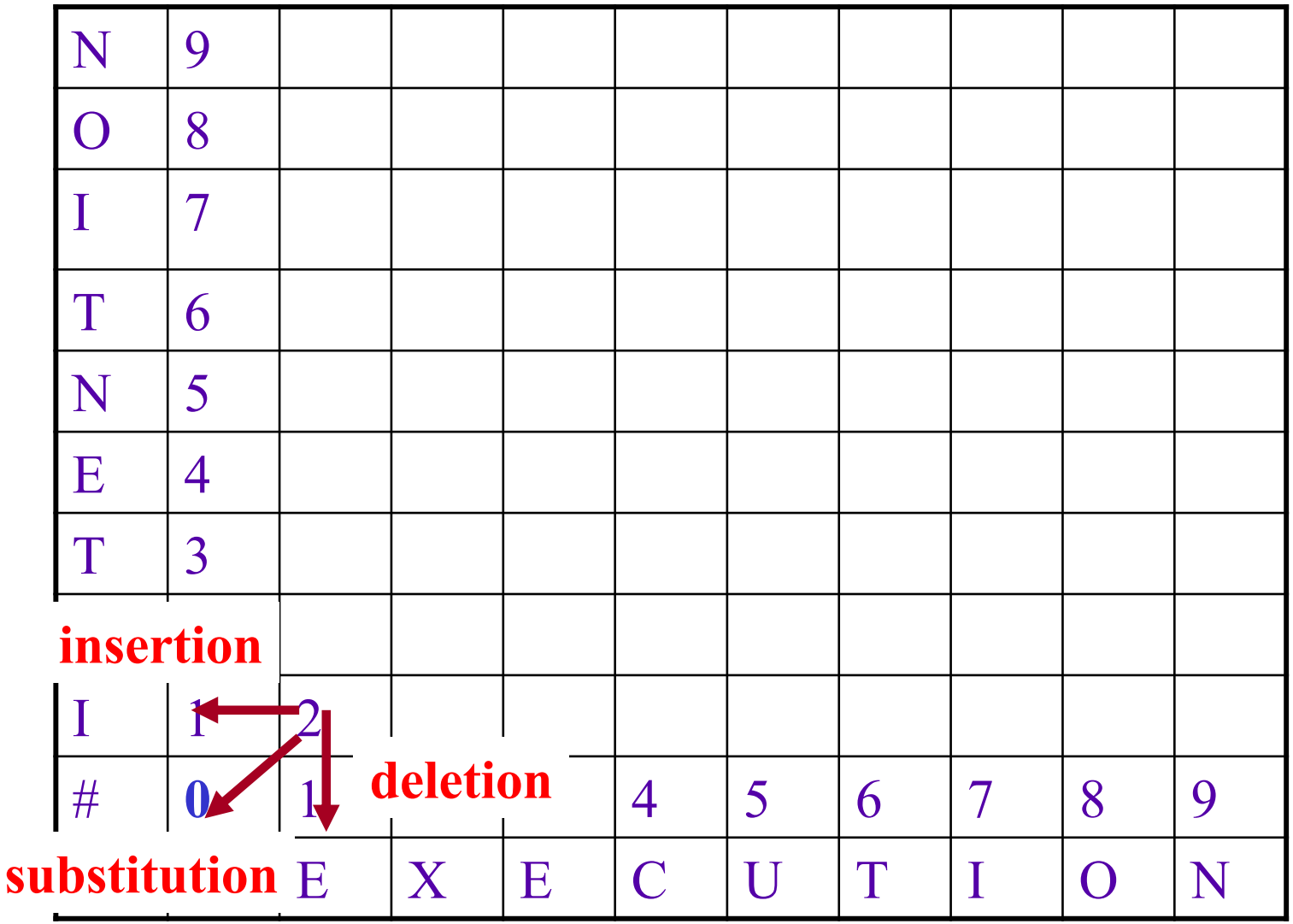
$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

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

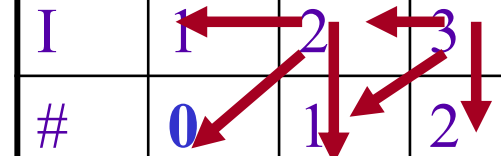
- Base conditions:
  - $D(i,0) = i$
  - $D(0,j) = j$
- Recurrence Relation:
  - $$D(i,j) = \min \begin{cases} D(i-1,j) + 1 & \text{DOWN} \\ D(i,j-1) + 1 & \text{LEFT} \\ D(i-1,j-1) + \begin{cases} 1; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} & \text{DIAGONAL} \end{cases}$$


N 9
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



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	↓ 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	↓ 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	ins	↙←↓ 8	↙ 7	←↓ 8	↙←↓ 9	↓ 8
n	2	↙←↓ **3**	↙	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↓ 7	↙←↓ 8	↙ 7
i	**1**	↙	↙←↓	↙←↓ 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**

sub

sub

del

Performance

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

Language Modeling

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.

Human Word Prediction

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

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

Handwriting Recognition

- Assume a note is given to a bank teller, which the teller reads as I have a gub.
- NLP to the rescue ….
  - gub is not a word
  - gun, gum, Gus, and gull are words, but gun has a higher probability in the context of a bank

Review of 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

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
  - $P(w_n|w_1,w_2 \ldots w_{n-1})$
- 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: $P(A \mid B) = \frac{P(A \wedge B)}{P(B)}$

So: $P(A \wedge B) = P(B \mid A)P(A)$

"the dog": $P(The \wedge dog) = P(dog \mid the)P(the)$

"the dog bites":

$P(The \wedge dog \wedge bites) = P(The)P(dog \mid The)P(bites \mid The \wedge dog)$

Chain Rule

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

$$P(w_1^n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2)...P(w_n \mid w_1^{n-1})$$

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

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:
  - $P(w_n \mid w_{n-1})$
  - We'll pretty much always be dealing with P(<word> | <some prefix>)
- unigrams: P(dog)
- bigrams: P(dog | big)
- trigrams: P(dog | the big)
- quadrigrams: P(dog | the big dopey)

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

N-gram models can be trained by counting and normalization

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

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

Review: MLE

Toss a coin: head or tail.

Suppose we know: P(head) = 0.5

Let's toss the coin 9 times, what is the probability of having 4 heads and 5 tails?

Review: MLE

Toss a coin: head or tail.

Suppose we know: P(head) = 0.5

Let's toss the coin 9 times, what is the probability of having 4 heads?

$$\frac{9!}{4!(9-4)!}0.5^4 \times (1-0.5)^{(9-4)} = 0.246$$

Review: Binomial Distribution

Toss a coin: head or tail.

Suppose we know: P(head) = p

Let's toss the coin n times, the probability of having h heads:

$$\frac{n!}{h!(n-h)!} p^h (1-p)^{n-h}$$

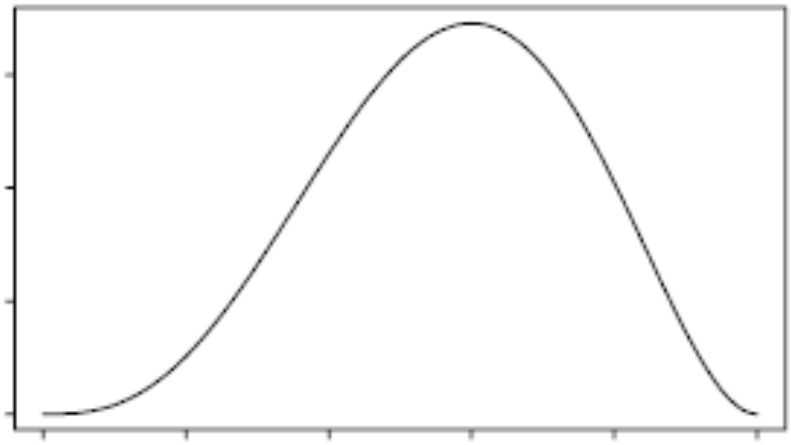
If p (i.e, the model parameter) is known,
we can get the probability of a certain observation (i.e., P(O|p))

Toss a coin: head or tail.

Suppose we know: P(head) = p

Let's toss the coin *5* times, the probability of having *3* heads:

$$L(p, 3) \propto p^3(1-p)^2$$



Maximum Likelihood Estimation

What if we only see the observation, we don't know p,

Then we want to find: $\arg\max_p P(p \mid O)$

Because: $P(p \mid O) \propto P(O \mid p)$ (why?)

Therefore we need to find: $\arg\max_p P(O \mid p)$

Find the model parameter that makes the observation data most likely

MLE Cont.

$$P(p \mid x_1,....x_n) \propto P(x_1,....x_n \mid p)$$

$$= p^{x_1}(1-p)^{1-x_1}....p^{x_n}(1-p)^{1-x_n} = p^{\sum x_i}(1-p)^{\sum(1-x_i)}$$

$$= p^{\sum x_i}(1-p)^{n-\sum x_i}$$

$$x_i = 0 \text{ or } 1, \text{ and } i = 1,...,n$$

$$\ln P = \sum x_i \ln p + (n - \sum x_i)\ln(1-p)$$

$$\frac{d(\ln P)}{dp} = \frac{\sum x_i}{p} - \frac{n - \sum x_i}{1-p} = 0$$

$$\sum x_i - p\sum x_i = np - p\sum x_i$$

$$\hat{p} = \frac{\sum x_i}{n}$$

MLE for N-gram

N-gram models can be trained by counting and normalization

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

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

Using N-Grams

- Recall that
  - $P(w_n \mid w_{1..n-1}) \approx P(w_n \mid w_{n-N+1..n-1})$
- For a bigram grammar
  - P(sentence) can be approximated by multiplying all the bigram probabilities in the sequence
  - P(I want to eat Chinese food) = P(I | <start>) P(want | I) P(to | want) P(eat | to) P(Chinese | eat) P(food | 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(I want to eat British food) = P(I|<start>) P(want|I) P(to|want) P(eat|to) P(British|eat) P(food|British) = .25*.32*.65*.26*.001*.60 = .000080
- vs. I 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.,

$$\log_2 P(w_1 ....w_n) = \log_2 \prod P(w_i \mid w_{i-1})$$
$$= \sum \log_2 P(w_i \mid w_{i-1})$$

- 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(as|come across) = 0.8
P(more|come across) = 0.1
P(a | 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(as|come across) = 0.8
P(more|come across) = 0.1
P(a | come across) = 0.1
P(x | 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.

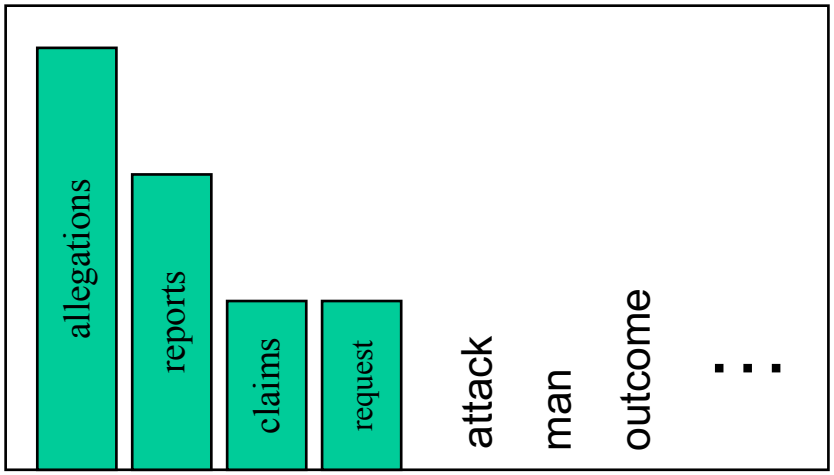
Devise better estimators that allow for the possibility that we will see events that we didn't see in the training text

How to address this problem? Smoothing!

The intuition of smoothing (from Dan Klein)

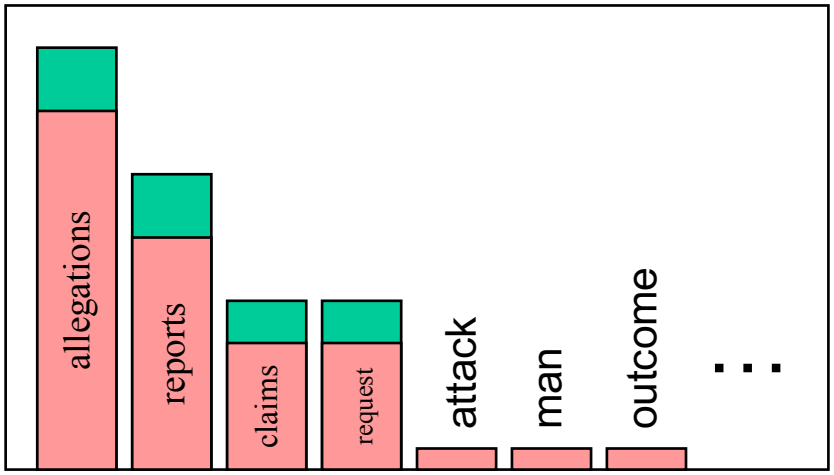
- When we have sparse statistics:

P(w | 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



Discounting Method

Suppose we've seeing the following counts

X	*count (x)*	$P_{ML}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})}$
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, Count*(x) = Count(x)-0.5

New estimates:

X	*count (x)*	*count*(x)*	$P_{ML}(w_i \mid w_{i-1}) = \frac{C^*(w_{i-1}w_i)}{C(w_{i-1})}$
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"

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

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

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

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

After add one smoothing:

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

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 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?

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

Backoff

Is the following right?

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

(Brants et al. 2007) – stupid backoff, no discounting -> very large language model.

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 :

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

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

Linear Interpolation

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

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

Linear Interpolation

Why does this estimation correctly define a distribution?

$$\sum_{w_i \in V} P_{LI}(w_i \mid w_{i-2}, w_{i-1})$$

$$= \sum_{w_i \in V} [\lambda_1 \times P_{ML}(w_i \mid w_{i-2}, w_{i-1}) + \lambda_2 \times P_{ML}(w_i \mid w_{i-1}) + \lambda_3 \times P_{ML}(w_i)]$$

$$= \lambda_1 \sum_{w_i \in V} P_{ML}(w_i \mid w_{i-2}, w_{i-1}) + \lambda_2 \sum_{w_i \in V} P_{ML}(w_i \mid w_{i-1}) + \lambda_3 \sum_{w_i \in V} P_{ML}(w_i)$$

$$= \lambda_1 + \lambda_2 + \lambda_3$$

$$= 1$$

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)

Knerser-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

Knerser-Ney Smoothing

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

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

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

$$P(w_i \mid c_{i-n+1....}c_{i-1}) = P(w_i \mid c_i)P(c_i \mid c_{i-n+1....}c_{i-1})$$

Class Trigram

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

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 (<s>, 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, </s>)
- Then string the words together
- <s> I
  I want
  want to
  to eat
  eat Chinese
  Chinese food
  food </s>

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 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

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:

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

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

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

Minimizing perplexity is the same as maximizing probability

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

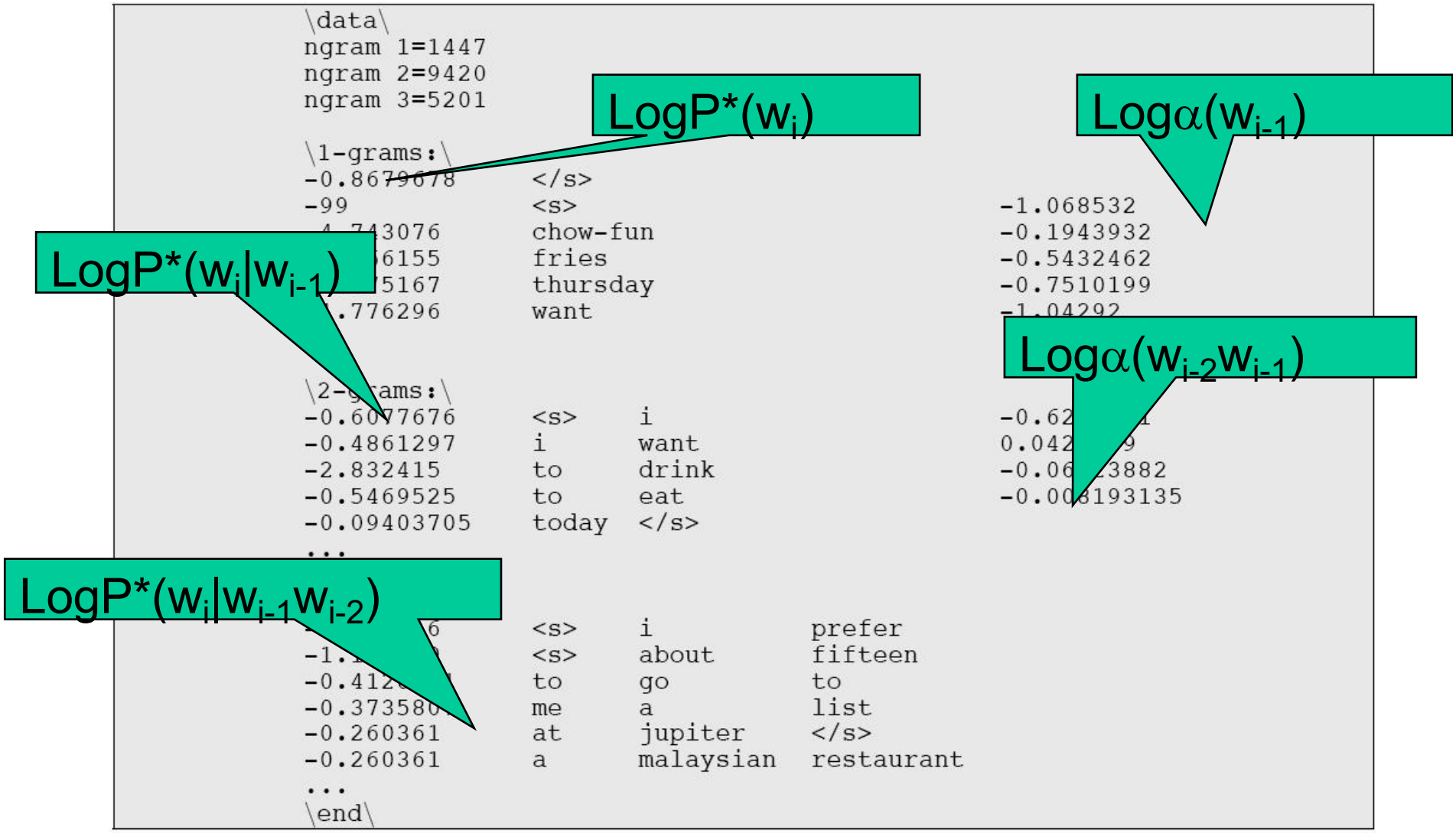
Lower perplexity means a better model

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

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Toolkits and Data Formats

- SRI Language Model Toolkit
  - http://www.speech.sri.com/projects/srilm/
  - ARPA Format for N-grams


\data\
ngram 1=1447
ngram 2=9420
ngram 3=5201
\1-grams:\
-0.8679678 </s>
-99 <s> -1.068532
-0.743076 chow-fun -0.1943932
-0.56155 fries -0.5432462
-0.75167 thursday -0.7510199
-1.776296 want -1.04292
\2-grams:\
-0.6077676 <s> i -0.62
-0.4861297 i want 0.042
-2.832415 to drink -0.06 3882
-0.5469525 to eat -0.008193135
-0.09403705 today </s>
...
-0.6 <s> i prefer
-1.1 <s> about fifteen
-0.412 to go to
-0.373580 me a list
-0.260361 at jupiter </s>
-0.260361 a malaysian restaurant
...
\end\
LogP*(wi)
Logα(wi-1)
LogP*(wi|wi-1)
Logα(wi-2wi-1)
LogP*(wi|wi-1wi-2)